Bayesian Optimisation for Automated Machine Learning

Binxin Ru

St Hugh's College

University of Oxford

Submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

Michaelmas Term 2022

Abstract

In this thesis, we develop a rich family of efficient and performant Bayesian optimisation (BO) methods to tackle various AutoML tasks. We first introduce a fast information-theoretic BO method, FITBO, that overcomes the computation bottleneck of information-theoretic acquisition functions while maintaining their competitiveness on the noisy optimisation problems frequently encountered in AutoML. We then improve on the idea of local penalisation and develop an asynchronous batch BO solution, PLAyBOOK, to enable more efficient use of parallel computing resources when evaluation runtime varies across configurations. In view of the fact that many practical AutoML problems involve a mixture of multiple continuous and multiple categorical variables, we propose a new framework, named Continuous and Categorical BO (CoCaBO) to handle such mixed-type input spaces. CoCaBO merges the strengths of multi-armed bandits on categorical inputs and that of BO on continuous space, and uses a tailored kernel to permit information sharing across different categorical variables. We also extend CoCaBO by harnessing the concept of local trust region to achieve competitive performance on high-dimensional optimisation problems with mixed input types.

Beyond hyper-parameter tuning, we also investigate the novel use of BO on two important AutoML applications: black-box adversarial attack and neural architecture search. For the former (adversarial attack), we introduce the first BO-based attacks on image and graph classifiers; by actively querying the unknown victim classifier, our BO attacks can successfully find adversarial perturbations with many fewer attempts than competing baselines. They can thus serve as efficient tools for assessing the robustness of models suggested by AutoML. For the latter (neural architecture search), we leverage the Weisfeiler-Lehamn graph kernel to empower our BO search strategy, NAS-BOWL, to naturally handle the directed acyclic graph representation of architectures. Besides achieving superior query efficiency, our NAS-BOWL also returns interpretable sub-features that help explain the architecture performance, thus marking the first step towards interpretable neural architecture search. Finally, we examine the most computation-intense step in AutoML pipeline: generalisation performance evaluation for a new configuration. We propose a cheap yet reliable test performance estimator based on a simple measure of training speed. It consistently outperforms various existing estimators on on a wide range of architecture search spaces and and can be easily incorporated into different search strategies, including BO, to improve the cost efficiency.

Statement of Originality

I, Binxin Ru, hereby declare that, except where made explicitly clear, the contents of this dissertation are my own original work, and to the best of my knowledge do not contain materials submitted in whole, or in part, for consideration for any other degree or qualification at the University of Oxford or any other academic or professional institution.

Binxin Ru
September 17, 2021

Acknowledgements

First and foremost, I want to thank my supervisors. Thank Michael Osborne and Stephen Roberts, for offering me the opportunity to pursue this memorable DPhil journey, granting me enormous freedom to explore interesting research topics and being always super supportive of my decisions. Specifically, Michael introduced to me the concept of Bayesian machine learning, especially Bayesian optimisation (BO), and has steadily guided me on this exciting research direction. Besides being an excellent academic supervisor, Michael has also been my life mentor, especially on parenting for which he is much more experienced. Stephen is a perfect co-supervisor who is very responsive and inspiring. Stephen provided many ideas and comments with his impressive breath and depth of knowledge on machine learning. I am also deeply grateful to Yarin Gal, my informal supervisor, who kindly accepted me into his OATML group when Michael was on parental leave in 2019 and has been actively supervising me since then. Yarin extensively stretched my understanding on Bayesian deep learning and sharpened my research thinking with many thought-provoking discussions. Also his energetic working style, impressive time management and capability to master and leverage a remarkably broad spectrum of research directions also influenced me heavily. In addition, I would also like to thank Fabio Carlucci, Frank Hutter, Jan-Peter Callies, Jun Wang, Mark van der Wilk, Nathan Koren, Pedro Esperanca, Philip Torr, Stefan Zohren, Xiaowen Dong, Zhenguo Li, for their enriching guidance and helpful comments during the course of my DPhil.

Second, I have been very lucky and honoured to work with and learn from a long list of excellent collaborators and colleagues. The list includes members of MLRG (Adam Cobb, Arno Blaas, Ivan Kiskin, Ahsan Alvi, Diego Granziol, Ed Wagstaff, Favour Mandanji Nyikosa, Gabriele Abbati, Henry Kenlay, Jack Fitzsimons, Logan Graham, Mark McLeod, Michael Cohen, Olga Isupova, Paul Duckworth, Pengfei Zhang, Saad Hamid, Sebastien Schulze, Sin-Chi Kuok, Vu Nguyen, Xingyue Pu, Yunpeng Li, Zihao Zhang), members of OATML (Aidan Gomez, Andreas Kirsch, Angelos Filos, Challenger Mishra, Clare Lyle, Joost van Amersfoort, Lewis Smith, Lisa Schut, Milad Alizadeh, Sebastian Farquhar, Tim G. J. Rudner, Tom Rainforth, Zac Kenton) as well as researchers outside Oxford (Arber Zela, Colin White, Fengwei Zhou, Jiashun Feng, Kaichen Zhou, Lanqing Hong, Roy Eyono, Yang Liu). Special thanks first go to Diego Granziol, with whom I shared the same office at Eagle

House, watched Dragon Ball series, had numerous stimulating discussions, and experienced several sleepless nights fighting for deadlines. Beyond his academic strength, Diego is also the most charismatic person I've ever met in Oxford, and has become my lifelong friend involved in most of my happiest memories during this long DPhil journey. I also want to thank Xingchen Wan, a direct junior of mine as we went to the same high school in Singapore, did the same undergraduate degree in Oxford as well as finally joined the same research lab for DPhil. Xingchen is a very active and quick learner with strong capability to fix problems and get things working, reminding me of a younger version of myself but a better version. Collaborating with you is such a smooth and amazing experience and I really enjoy the synergy and efficiency when we teamed up. Moreover, it is my great pleasure to work with Ahsan Alvi and Vu Nguyen, both of whom broadened the scope of my understanding on BO and improved my research skills/efficiency with your professional well-organised working style. Vu also provided me with many helpful advices on research in general. Finally, I want to thank Zihao Zhang, Xingyu Pu and Sin-Chi Kuok for bringing lots of fun to my life in Oxford.

Third, I would like to take this opportunity to thank my family members. My deepest gratitude goes to Yang Yuan, my beloved wife, who has accompanied me through all the ups and downs over the years and has empowered me with enormous understanding and encouragement. Without your support, I would hardly start the DPhil journey, not to mention completing it with such a satisfactory/fulfilling fashion. I also owe the huge debt of gratitude to my parents for your unwavering love and support which make me who I am today. A final thank-you note goes to my daughter, Yichen Ru. Despite many chaotic and stressful moments, you are the greatest source of joy, happiness and motivation for me during the final lapse of my DPhil study. In fact, you made the endless stay at home due to covid-19 a positive experience for me.

Lastly, I would like to thank the Clarendon Fund for supporting three years of my PhD with the Clarendon Scholarship, and the Oxford-Man Institute of Quantitative Finance for providing a user-friendly server and a nice working environment.

The main chapters of this thesis are based on a number of joint-authored publications. These are listed below, along with a description of my contribution to them.

- **Chapter 3**

  Binxin Ru, Mark McLeod, Diego Granziol, and Michael A. Osborne. Fast information-theoretic Bayesian optimisation. In *International Conference on Machine Learning (ICML 2018)*, 2018.

I was responsible for all the theory, algorithmic development and experiments. Granziol helped with experiments on real world problems and McLeod contributed to the discussion and implementation of other information-theoretic baselines. Osborne was key in suggesting the initial research idea and provided general advice on theory and experiments.

- **Chapter 4**

  > Ahsan Alvi*, Binxin Ru*[1], Jan-Peter Calliess, Stephen Roberts, and Michael A. Osborne. Asynchronous Batch Bayesian Optimisation with Improved Local Penalisation. In *International Conference on Machine Learning (ICML 2019)*, 2019.

  Alvi and I contributed equally to the theory, research idea discussions and experiments. Specifically, I led the design of the local penalisation function, created all the illustrative plots in the paper and performed the real-world experiment on tuning CNN hyper-parameters. Alvi led the asynchronous design and method implementation, and performed all other experiments. Calliess provided theoretical insights on Lipschitz constants. Roberts and Osborne provided problem motivation as well as general advice on theory and experiments.

- **Chapter 5**

  > Binxin Ru*, Ahsan Alvi*, Vu Nguyen, Michael A. Osborne, and Stephen Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning (ICML 2020)*, 2020.

  Alvi and I contributed equally to the theory, algorithmic development and experiments. Specifically, I was responsible for the multi-armed bandits part of the algorithm and its regret analysis while Alvi focused on the continuous BO part. We jointly designed the CoCaBO kernel and the batch algorithm. All the experiments are performed jointly by both of us. Nguyen and Roberts provided general advice on the problem motivation and kernel design respectively. Osborne provided general guidance.

  > Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A. Osborne. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. In *International Conference on Machine Learning (ICML 2021)*, 2021.

[1] * denotes equal contribution.

Wan was the main author of the paper who led the methodology development and experiments. Nguyen provided the problem motivation and proposed the initial research idea with Wan. Ha provided all the theoretical analyses. I contributed to the discussion on how to extend the categorical kernel and combined kernel in CoCaBO (Ru et al., 2020a) for the proposed new method. I also contributed to running the baselines for mixed-input-type experiments as well as helped set up the black-box adversarial attack experiment. Osborne provided general guidance.

- **Chapter 6**

  Binxin Ru, Adam Cobb, Arno Blaas, and Yarin Gal. BayesOpt adversarial attack. In *International Conference on Learning Representations (ICLR 2020)*. 2020.

I contributed to the majority of the theory, algorithmic development and experiments. Cobb contributed to the research idea discussions. Blaas contributed to the discussions on dimensionality reduction techniques, related attack approaches and experiment design. Gal provided the problem motivation and important guidance on all theory and experiments.

  Xingchen Wan, Henry Kenlay, Binxin Ru, Arno Blaas, Michael A. Osborne, and Xiaowen Dong. Attacking graph classification via Bayesian optimisation. In *International Conference on Machine Learning Workshop on Adversarial Machine Learning*, 2021.

Wan was the main author of the paper who drove the theory, algorithmic development and experiments. Kenlay contributed to the graph theory, the implementation of existing graph attack approaches and attack result analyses. I contributed to the discussions about the problem motivation, the BO-based attack framework and the surrogate model of the BO attack. Blaas contributed to background information on adversarial attacks and general discussions about algorithmic development. Osborne provided general advices, especially on experiments. Dong provided important guidance on the problem motivation, graph theory, graph attacks and experiments.

- **Chapter 7** NAS-BOWL

  Binxin Ru*, Xingchen Wan*, Xiaowen Dong, and Michael A. Osborne. Interpretable neural architecture search via Bayesian optimisation with Weisfeiler-Lehman kernels. In *International Conference on Learning Representations (ICLR 2021)*, 2021.

Wan and I contributed equally to the theory, algorithmic development and experiments. Specifically, I provided the problem motivation and background on neural architecture search. I was also responsible for illustrating the Weisfeiler-Lehman procedure, processing NAS-Bench datasets and running baseline methods for all experiments. Wan was responsible for implementing and running our method for all experiments and generating useful plots in the interpretability section. We jointly developed the search strategy and its interpretability aspect. Dong and Osborne provided valuable guidance on graph kernel and BO respectively.

- **Chapter 8** TSE

  Binxin Ru*, Clare Lyle*, Lisa Schut, Mark van der Wilk, and Yarin Gal. Speedy Performance Estimation for Neural Architecture Search. In *arXiv preprint arXiv:2006.04492 (Under review of NeurIPS 2021)*, 2021.

  Lyle and I contributed equally to the theory and algorithmic development. Specifically, I provided the problem motivation, background on neural architecture search as well as was responsible for all the experiment design and empirical analyses. Lyle drove the theoretical connection to training speed and Bayesian marginal likelihood. Schut contributed to the theoretical connection and helped with the experiments. van der Wilk and Gal provided crucial guidance on the theoretical motivations, all the experiment design as well as result analyses.

Contents

List of Figures **xiv**

List of Tables **xxv**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Contribution . . . 3
1.3 Thesis Structure . . . 9

2 Background on Bayesian Optimisation **11**
2.1 Gaussian Process . . . 13
2.1.1 Kernel Functions . . . 16
2.1.2 GP Hyper-parameter Treatment . . . 17
2.2 Acquisition Functions . . . 19
2.2.1 Improvement-based Acquisition Functions . . . 19
2.2.2 Optimistic Acquisition Functions . . . 22
2.2.3 Information-based Acquisition Functions . . . 23
2.3 Extension Settings . . . 25
2.3.1 Batch BO . . . 25
2.3.2 High-dimensional BO . . . 29
2.3.3 BO on Categorical or Mixed Inputs . . . 31

3 Fast Information-theoretic Bayesian Optimisation **33**
3.1 Introduction . . . 34
3.2 Fast Information-theoretic Bayesian Optimisation . . . 36
3.2.1 Parabolic Transformation and Predictive Posterior Distribution 37
3.2.2 Hyper-parameter Treatment . . . 39
3.2.3 Approximation for the Gaussian Mixture Entropy . . . 40
3.2.4 The Algorithm . . . 43
3.3 Experiments . . . 43
3.3.1 The Acquisition Function Obtained by Numerical Integration and Moment-matching . . . 45
3.3.2 Runtime Tests . . . 45

3.3.3 Tests with Benchmark Functions 49
3.3.4 Test with Real-world Problems 51
3.4 Conclusion 53

4 Asynchronous Bayesian Optimisation with Improved Local Penalisation 55
4.1 Introduction 56
4.2 Related Work 57
4.3 Preliminaries 58
4.4 Asynchronous vs Synchronous BO 59
4.5 Penalisation-based Asynchronous BO 61
4.5.1 Hard Local Penaliser 62
4.5.2 Local Estimated Lipschitz Constants 64
4.6 Experiments 66
4.6.1 Implementation Details 67
4.6.2 Synchronous vs Asynchronous BO 68
4.6.3 Asynchronous Parallel BO 71
4.7 Conclusions 74

5 Bayesian Optimisation over Multiple Continuous and Categorical Inputs 76
5.1 Introduction 77
5.2 Preliminaries 79
5.3 Related Work 80
5.4 Continuous and Categorical Bayesian Optimisation 82
5.4.1 CoCaBO Bandit Algorithm 82
5.4.2 CoCaBO Kernel Design 85
5.4.3 Batch CoCaBO 88
5.5 Experiments 90
5.5.1 Predictive Performance of the CoCaBO Posterior 93
5.5.2 Performance of CoCaBO in Sequential Setting 94
5.5.3 Performance of CoCaBO in Batch Setting 95
5.6 Extension Work 97
5.6.1 Modified Kernel Design 98
5.6.2 Trust Region 99
5.6.3 Acquisition Function Optimisation 100
5.6.4 Experiments 101
5.7 Conclusion 105

6 Bayesian Optimisation for Adversarial Attack 107
6.1 Introduction . . . 108
6.2 Related Work . . . 111
6.3 Preliminaries . . . 113
6.3.1 Problem Definition . . . 113
6.3.2 Bayesian Optimisation . . . 114
6.4 BayesOpt Attack . . . 114
6.4.1 BayesOpt Attack Objective . . . 114
6.4.2 GP-based BayesOpt Attack . . . 115
6.4.3 Additive GP-based BayesOpt Attack . . . 116
6.4.4 Learning The Optimal d^r . . . 117
6.5 Experiments . . . 119
6.5.1 Effect of d^r . . . 120
6.5.2 Performance Under a Fixed Query Budget . . . 121
6.5.3 Query Efficiency Comparison . . . 125
6.5.4 Ablation Studies . . . 126
6.6 Extension Work . . . 128
6.6.1 Problem Setup . . . 128
6.6.2 Surrogate Model in GRABNEL . . . 130
6.6.3 Sequential Perturbation Selection . . . 131
6.6.4 Experiments . . . 133
6.6.5 Atack Analysis . . . 136
6.7 Conclusion . . . 138

7 Bayesian Optimisation for Neural Architecture Search 140
7.1 Introduction . . . 141
7.2 Preliminaries . . . 143
7.3 Bayesian Optimisation with Weisfeiler-Lehman Graph Kernel . . . 145
7.3.1 Surrogate Design and Graph Kernel . . . 146
7.3.2 Acquisition Function Optimisation over Graph Inputs . . . 149
7.3.3 Interpretable Motifs . . . 150
7.4 Related Work . . . 155
7.5 Experiments . . . 157
7.5.1 Surrogate Regression Performance . . . 157
7.5.2 Architecture Search on NAS-Bench Datasets . . . 162
7.5.3 Architecture Search on Open-domain Space . . . 164
7.5.4 Ablation Studies . . . 166
7.6 Conclusion . . . 169

8 Speedy Performance Estimation for Neural Architecture Search 170
8.1 Introduction . . . 171
8.2 Training Speed Estimation . . . 172
8.2.1 Theoretical Motivations . . . 172
8.2.2 Method Description . . . 175
8.3 Related Work . . . 177
8.4 Experiments . . . 179
8.4.1 Hyper-parameters of TSE Estimators . . . 181
8.4.2 Comparison of Performance Estimation Quality . . . 182
8.4.3 Architecture Generator Selection . . . 189
8.4.4 Speed up Query-based NAS . . . 189
8.4.5 Improving One-shot and Gradient-based NAS . . . 192
8.4.6 Ablation Studies . . . 197
8.5 Conclusion . . . 201

9 Conclusions and Future Directions 203
9.1 Conclusion . . . 203
9.2 Future Directions . . . 206

Appendices

A Additional Experiments 212
A.1 Compare Methods for Approximation a Gaussian Mixture Entropy 212
A.2 Compute a Gaussian Mixture Entropy in FITBO using Maximum Entropy Algorithm . . . 216
A.2.1 Moments of a Gaussian Mixture . . . 216
A.2.2 Maximum Entropy Distribution . . . 216
A.2.3 Approximate Gaussian Mixture Entropy with MEMe . . . 218
A.2.4 Experiments . . . 219
A.3 Additional Experiments for PLAyBOOK . . . 223
A.3.1 Asynchronous vs. Synchronous Parallel BOs . . . 223
A.3.2 Asynchronous Parallel BOs . . . 223
A.4 Additional Experiments for BayesOpt Attack . . . 229
A.5 Combining Different Kernels to Improve GPWL . . . 229
A.6 More Motif Discovery on NB101 and Other NB201 Tasks . . . 231
A.7 Additional NAS-BOWL Experiments . . . 233
A.8 Additional Experiments for TSE . . . 234
A.8.1 Overfitting on CIFAR10 and CIFAR100 . . . 234

B Additional Informations **237**
B.1 Kriging Believer 237
B.2 Algorithmic Description of the WL kernel 238
B.3 Other Implementation Details of NAS-BOWL and Competing Baselines 238
B.4 Experiment Set-up for the DARTS Space 241
B.5 NAS Datasets Description 242

C Additional Derivatives **247**
C.1 Compute the derivatives of the FITBO Acquisition Functions 247
C.1.1 Derivative of FITBO 248
C.1.2 Derivative of FITBO-MM acquisition function 250
C.2 Learn the Hyper-parameters of the CoCaBO Kernel 250
C.3 Theoretical Analysis for CASMOPOLITAN 251
C.3.1 Lemma on Validity of Proposed Kernels 252
C.3.2 Theorem on Maximum Information Gains of Proposed Kernels 252
C.3.3 Theorem on Local Convergence 257
C.3.4 Theorem on Global Convergence of Categorical Setting . . . 260
C.3.5 Proof of Theorem 3 260
C.3.6 Theorem on Global Convergence of Mixed Setting 262
C.3.7 Proof of Theorem 4 262

Bibliography **266**

List of Figures

2.1 Demonstration of BO on a 1D objective function. The red dashed line represents true objective function.The solid black line and blue shade represents the mean and ± standard deviation of the prediction by the surrogate model (GP in this case) that approximates the objective function. The green line denotes the acquisition function whose value is high at locations where the surrogate model gives a low predictive mean (exploitation) and/or high predictive variance (exploration). This figure is inspired by a similar figure in (Brochu et al., 2010b). 12

2.2 Random samples generated by a GP prior with the SE-ARD kernel function and different characteristic length scales. The smaller length scale leads to more wiggly function. 16

3.1 BO for a 1D objective function using FITBO method at the 1st, 3rd, 5th evaluations. In each subfigure, the top plot shows the objective function (red dotted line), the posterior mean (black solid line) and the 95% confidence interval (blue shaded area) estimated by the Gaussian process model as well as the observation points (black dot) and the next query point (red dot). The middle plot shows the acquisition function. The bottom plot is the histogram of η samples as well as its relation to the minimum observation (black vertical line) and the true global minimum (red vertical line). 44

3.2 Acquisition functions obtained using numerical integration and moment-matching approximation methods. The top plot shows the objective function (red dotted line), the posterior mean (black solid line) and the 95% confidence interval (blue shaded area) estimated by the Gaussian process model as well as the observation points (black dot). The following three plots show the acquisition function value of FITBO-MM and those of FITBO with different tolerance level (1e-3, 1e-4, 1e-6) used for numerical integration.The next query points are recommended by maximising respective acquisition functions: FITBO-1e-3 (green dot), FITBO-1e-4 (red dot), FITBO-1e-6 (pink dot) and FITBO-MM (blue triangle). The acquisition functions are computed using 600 hyperparameter samples. 46

3.3 The runtime of evaluating 7 different acquisition functions (PI, EI, GP-UCB, PES, MES, FITBO and FITBO-MM) at 100 test inputs. The left plot shows the runtime of evaluating the acquisition functions using M hyper-parameter samples for 2D input data and M tested are 100, 300, 500, 700, 900. The right plot shows the runtime of evaluating the acquisition functions using 400 hyper-parameter samples for input data of dimension d where d are 2, 4, 6, 8, 10. The y-axes are the evaluation runtime expressed in the logarithm to the base 10. 47
3.4 Optimisation performance of EI, PES, MES, FITBO and FITBO-MM for three benchmark test functions. 49
3.5 Evaluations taken by FITBO and EI in the Branin-2D problem. The white crosses in the top plots indicate the first 50 query points recommended by the two algorithms. The yellow triangles in the bottom plots indicate the guesses of the global minimiser recommended by the algorithms (i.e. $\hat{\mathbf{x}}_n$) after each evaluation. FITBO, which is more explorative in taking evaluations, successfully identifys all three global minimisers (red circle) but EI misses out one of the global minimisers. 50
3.6 Performance on tuning hyper-parameters for (a) training a neural network on the Boston housing dataset, (b) training an SVM on the MNIST dataset and (c) training a classification neural network on the breast cancer dataset. 52

4.1 Illustration showing the difference between synchronous and asynchronous batch BO in the case of $k = 3$ parallel workers. The blue bar indicates the processing time taken for a worker to evaluate its assigned task and the red bar indicates the waiting time for a worker between completing its previous task and beginning a new task. It is clear that asynchronous batch BO, which makes better use of the computing resources, can complete a greater number of evaluations than its synchronous counterpart within the same duration. 59
4.2 Illustration of asynchronous batch selection by naïve LP and HLP. The top left plot shows the acquisition function $\alpha(\mathbf{x})$ and the locations (i.e. $\mathbf{x}_{b1}$ and $\mathbf{x}_{b2}$ denoted in black dots) under evaluation by busy workers. The top right plot shows the shapes of two penalisers at the busy location $\mathbf{x}_{b1}$. Their respective penalisation effects on $\alpha(\mathbf{x})$ at $\mathbf{x}_{b1}$ and $\mathbf{x}_{b2}$ as well as the new batch point $\mathbf{x}_3$ to be assigned to the available worker are shown in the subplots that follow, LP on the left and HLP on the right. 64

4.3 Different penalisation effects on $\alpha(x)$ of using a single global Lipschitz constant compared to local Lipschitz constants. The top plots in both (a) and (b) show the true objective function (red line), six observed points (black crosses), the GP posterior mean (black line) and variance (blue shade), the two busy locations (black dots) and the next query point (red dot) selected by using the HLP with global and local Lipschitz constants respectively. The plots in (a) show the penalisation effect on busy locations using the same global Lipschitz constant while those in (b) show the effect of using local Lipschitz constants. It is clear that penalising the busy locations based on local Lipschitz constants allows the algorithm to capture the informative peak at the central region while selection based on the single global Lipschitz constant leads us to revisit the flat region near the boundary due to insufficient penalisation at $\mathbf{x}_1$. 65

4.4 A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO method over *evaluation time, t*. The median (solid line) and quartiles (shaded region) of the regret for optimising ack-5D for 30 random initialisations are shown. As expected, asynchronous methods clearly outperform their synchronous counterparts in terms of evaluation time. 69

4.5 A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO method over *number of evaluations, N*. The median (solid line) and quartiles (shaded region) of the regret for optimising ack-5D for 30 random initialisations are shown. Surprisingly, asynchronous methods also outperform their synchronous counterparts in terms of sample efficiency. 70

4.6 The median (solid line) and quartiles (shaded region) of the regret for different asynchronous BO methods on the global optimisation test functions for 30 random intialisations is shown. We can see that our proposed PLAyBOOK methods perform competitively, especially when we start choosing larger batch sizes. 71

4.7 Asynchronous optimisation of 9 hyper-parameters of a 6-layer CNN for image classification on the CIFAR10 dataset. The network is trained on half of the training set and evaluated on the second half. The objective being minimised is the classification accuracy on the validation set. PLAyBOOK outperforms both KB and TS in this expensive optimisation task. 73

5.1 Optimisation procedure in CoCaBO. 83

5.2 CoCaBO correctly optimises the two categorical inputs h_1 (Red) and h_2 (Blue) of the Func-2C test function over 200 iterations. The *best* category is $h_i = 2$ for both h_1 ($N_1 = 3$) and h_2 ($N_2 = 5$), and is highlighted in all plots. The left subplot shows the selections made by CoCaBO, showing how the both categorical inputs increasingly focus on the best categories as the algorithm progresses. The second left subplot shows the histogram of categories selected, with the best category being chosen the most frequently. The right subplots show the reward for each categorical value for h_1 and h_2 across iterations. Again, we see the correct category being identified for both categorical inputs for the highest rewards. 86

5.3 Three example cases for selecting a batch of 4 unique points ($b = 4$). $\mathbf{u}_j$ represents a unique categorical point. If all the categorical points in the batch are different, we select 1 continuous location $\mathbf{x}^{(i)}$ conditioned on each $\mathbf{u}_j$ (Case A). If 2 out of a batch of 4 categorical points are equal to $\mathbf{u_1}$, we sequentially select 2 continuous locations $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ conditioned on $\mathbf{u_1}$ using KB and collect only 1 continuous location for the other two categorical data $\mathbf{u_2}$ and $\mathbf{u_3}$ (Case B). 89

5.4 Performance of CoCaBOs against existing methods on various tasks in the sequential setting ($b = 1$). 94

5.5 Performance of CoCaBOs against existing methods on synthetic and real-world tasks in the batch setting ($b = 4$). 96

5.6 Illustration of CASMOPOLITAN in mixed space. Note that in **Steps 1** & **3** we show the *GP posterior* on $\mathcal{X}$ conditioned on the incumbent $\mathbf{h}^*_\mathbf{T}$, and in **Step 2** we show the *acquisition function* on $\mathcal{X}$ conditioned on $\mathbf{h}$ at various optimisation steps. Suppose we optimise over a 5-dimensional mixed problem with 3 categorical dimensions with $\{3, 2, 2\}$ choices for each dimension respectively, and 2 additional continuous dimensions. Initially (**Step 1**), the best location so far $\mathbf{z}^*_t = \arg\max_\mathbf{z}\{y_j\}_{z=1}^t = [\mathbf{h}^*_\mathbf{t}, \mathbf{x}^*_\mathbf{t}]$ (with the continuous TR and $\mathbf{x}^*_T$ in red box and cross). In optimisation of acquisition function (**Step 2**), we interleave the local search on $\mathcal{H}$ described in Section 5.6.3 with gradient-based optimisation on $\mathcal{X}$ until convergence. In **Steps 3a**/**3b**, we adjust both the continuous and categorical TRs correspondingly and restart if/when either shrinks below its minimum length. 97

5.7 Results on various categorical optimisation problems. Lines and shaded area denote mean ± 1 standard error. 101

5.8 Results on various mixed optimisation problems. Lines and shaded area denote mean ± 1 standard error. 103

5.9 Ablation studies of our method on categorical and mixed optimisation problems. 104

6.1 Performance of BayesOpt attack with GP surrogate in different reduced dimension d^r. Each color represents a particular d^r. The **left** subplot shows the attack success rates(ASR) for attacking all the other 9 classes on each of the 5 original images. The d^r that leads to the highest ASR varies with the original images. The **middle** subplot shows the number of queries needed to attack the same attack instances (i.e. the same original image and the same attack target) by using different d^r. For the same 4 attack instances, the **right** subplot show the effect of the choice of d^r on the L_2 distances between the resultant adversarial examples and the original image. . 120

6.2 CIFAR10 adversarial examples generated by our BayesOpt attack. True labels and target labels correspond to the rows and columns. . 124

6.3 Query efficiency of BayesOpt Attacks. The plots show the attack success rate (ASR) of different methods up to certain query counts. The best BayesOpt attacks (i.e. GP-BO-auto-d^r (red) and ADDGP-BO (blue)) can achieve an ASR of 98% on MNIST(a) and 87% on CIFAR10 (b) within a budget of 1000 queries. To achieve the same success rates, GenAttack (purple) takes 3500 for MNIST and 5141 for CIFAR10, and AutoZOOM (green) takes 800 for MNIST and 3880 for CIFAR10. For ImageNet (c), the best BayesOpt attack is ADDGP-BO (blue), which achieves an ASR of 60% with 1985 queries. To achieve the same success rates, GenAttack (purple) takes a budget of 4711 queries and AutoZOOM (green) takes 19451 queries. ZOO fails to make any attack on ImageNet within the given budget. . . . 125

6.4 Attack objective value against BayesOpt iterations(query count) for using various BayesOpt methods to attack one CIFAR10 image of class label 9 (denoted by i). Curves of different colours correspond to the 9 different target labels (denoted by t). Convergence to 0 objective value indicates successful attack. ADDGP-BO and GP-BO-auto-d^r enjoy faster convergence and thus higher attack success rates than simple GP-BO. 127

6.5 Sequential edge selection. At each stage the BO agent sequentially proposes candidate graphs with edge edit distance of 1 from the base graph $\mathcal{G}_0'^{(i)}$ (which is the original unperturbed graph $\mathcal{G}$ at initialisation, or a perturbed graph that led to the largest increase in loss from the previous stage otherwise). This procedure repeats until either the attack succeeds (i.e. we find a graph $\mathcal{G}'$ with $y_{\text{attack}}(\mathcal{G}') > 0$) or the maximum number of B queries to the victim model f is exhausted. 132

6.6 Attack success rate (ASR) against the number of queries to the victim models (normalised by the square of the number of nodes of each graph) in various datasets on GCN and GIN models. Note the x-axis is on log-scale. Lines and shades denote mean ±1 sd across 3 random initialisations. It is evident that `grabnel` outperforms other attack methods considerably. `Random` and `Genetic` appear to converge faster in some tasks as they always use exploit full perturbation budget allocated, while `grabnel` only attempts a higher perturbation budget when attack fails in a lower one. 134
6.7 ASR of GRABNEL against number of queries on a GCN model trained on PROTEINS dataset under various attack constraints. . . 136
6.8 Adversarial examples found by the proposed method. Red edges denote deleted edges from the original samples and green edges indicate those added. In Twitter fake news detection task, green nodes/edges denote the injected nodes and their connections to the existing graphs. 137

7.1 Illustration of one WL iteration. Given two architecture cells at initialisation, WL kernel first collects the neighbourhood labels of each node (Step 1) and compress the collected $h = 0$ labels into $h = 1$ features (Step 2). Each node is then relabelled with $h = 1$ features (Step 3) and the two graphs are compared based on the histogram on both $h = 0$ and $h = 1$ features (Step 4). This WL iteration will be repeated until $h = H$. h is both the index the WL iteration and the depth of the subtree features extracted. Substructures at $h = 0$ and $h = 1$ in Arch A are shown in the bottom right of the plot. . . 146
7.2 Motif discovery on NB201-CIFAR10 and DARTS spaces. 150
7.3 Best cells discovered by (left to right) DARTS, ENAS, LaNet, BOGCN and NAS-BOWL (ours) in the DARTS search space. Note the dominance of separable convolutions (especially `3x3`) in operation nodes (i.e. nodes excluding `input`, `output` and `add`) and the presence of highlighted structures encompassing the boxed motif in Figure 7.2 in all cells. 154
7.4 Mean Spearman correlation achieved by various surrogates across 20 trials on different datasets. Error bars denote ±1 standard error. The red dashed lines are there to help with visual comparison between the performance of GPWL and other baselines. GPWL can achieve superior regression performance measured in terms of high rank correlation with at least 3 times fewer training data than GCN on NB201 (CIFAR10, CIFAR100, ImageNet120) datasets and over 20 times fewer training data on datasets with larger search spaces (RWNN-Flower102 and NB101-CIFAR10). 158

7.5 Spearman correlation on train/validation sets and the negative log-marginal likelihood of GP against H (the maximum WL iteration) and the histograms of selected H by GPWL over 20 trials on (a) NB101-CIFAR10 and (b) RWNN-Flower102. 160
7.6 Predicted vs ground-truth validation error of GPWL in various NAS-Bench tasks in log-log scale. Error bar denotes ±1 SD from the GP posterior predictive distribution. 161
7.7 Median test error on NAS-Bench datasets with *deterministic* observations from 20 trials. Shades denote ±1 standard error and black dotted lines are ground-truth optima. Note the seemingly large regret in NB201-ImageNet120 is due to that there are only 5 out of 15.6K architectures with test error in the interval of [52.69 (optimum), 53.25]. 162
7.8 Median test error on NAS-Bench datasets with *noisy* observations from 20 trials. Shades denote ±1 standard error and black dotted lines are ground-truth optima. 163
7.9 Equivalent representations of the best cell identified by NAS-BOWL in the DARTS search space. Our method uses the node-attributed version during search, and this cell is used for both the normal and reduction cells. 165
7.10 Effect of varying P on NAS-BOWL in NB101. 167
7.11 Ablation studies of various design choices for NAS-BOWL 168

8.1 Rank correlation performance of TSE-E computed over E most recent epochs. Different E values are investigated for architectures in NASBench-201 (NB201) on three image datasets and 5000 architectures from NASBench-301 (DARTS) on CIFAR10. In all cases, smaller E consistently achieves better rank correlation performance with $E = 1$ being the best choice. 182
8.2 Rank correlation performance of TSE-EMA with γ on architectures in NASBench-201 (NB201) on three image datasets and 5000 architectures from NASBench-301 (DARTS) on CIFAR10. In all cases, TSE-EMA is very robust to different γ values; the difference among TSE-EMA with different γ is indistinguishable compared to that with TSE-E. 183
8.3 Rank correlation performance of various baselines for (a) to (c) NB201 architectures on three image datasets and for (d) RWNNs on Flowers102. In all cases, our TSE-EMA and TSE-E achieve superior rank correlation with the true test performance in much fewer epochs than other baselines. In (a) and (c), we mark SNIP in a violet dotted line labelled with its rank correlation value as it falls out of the plotted range. 184

8.4 Rank correlation performance of various baselines for ResNet and ResNeXt architectures on CIFAR10. In (b) and (d), we evaluate estimators on the top 1% of the ResNet/ResNeXt architectures and show that our TSE-EMA and TSE-E can remain competitive on ranking among top architectures, which are particularly desirable for NAS. 185

8.5 Rank correlation performance of various baselines for 5000 small 8-cell architectures (a) and 150 large 20-cell architectures (b) to (d) from DARTS search space on CIFAR10. We use NAS-Bench-301 dataset(NAS301) for computing (a) and for large architectures, we test three training hyper-parameter set-ups with different initial learning rates, learning rate schedulers and batch sizes as denoted in the subcaptions. On all four settings, our TSE-E again consistently achieves superior rank correlation in fewer epochs than other baselines. Note all three zero-cost estimators perform poorly (below the plotted range) on DARTS search space across all settings. We denote them in dotted lines with their rank correlation value labelled. 186

8.6 Rank correlation performance up to $T = T_{\text{end}}$. If the users want to apply our estimators for large training budget, they can estimate the effective range of our estimators based on the minimum epoch T_o when overfitting happens among the N_s observed architectures. They can then stop our estimators early at $0.9T_o$(marked by vertical lines) or switch back to validation accuracy beyond that. 187

8.7 Model selection among 69 random graph generator hyper-parameters on RWNN-Flower102 dataset using our TSE-EMA (a) and VAccES (b). We use each hyper-parameter value to generate 8 architectures and evaluate their true test accuracies after complete training. The mean and standard error of the test performance across 8 architectures for each hyper-parameter value are presented as Test Acc (yellow) and treated as ground truth (Right y-axis). We then compute our TSE-EMA estimator for all the architectures by training them for $T < T_{\text{end}} = 250$ epochs. The mean and standard error of TSE-EMA scores for $T = 10, \ldots, 90$ are presented in different colours (Left y-axis of (a)). The rank correlation between the mean Test Acc and that of TSE-EMA for various T is shown in the corresponding legends in (a). The same experiment is conducted by using early-stopped validation accuracy (VAccES) for performance estimation (b). With only 10 epochs of training, our TSE-EMA estimator can already capture the trend of the true test performance of different hyper-parameters relatively well (Rank correlation= 0.851) and can successfully identify 24-th hyperparamter setting as the optimal choice. The prediction of best hyper-parameter by VAccES is less consistent and the rank correlation scores of VAccES at all epochs are lower than those of TSE-EMA. 190

8.8 NAS performance of Regularised Evolution (RE), Bayesian Optimisation (BO) and Random Search (RS) in combined with final validation accuracy Val Acc (T=200), early-stopping validation accuracy Val Acc (T=10) and our estimator TSE-EMA(T=10) on NB201. For each subplot, we experiment on the three image datasets: on CIFAR10 (left), CIFAR100 (middle) and ImageNet120 (right). TSE-EMA leads to the fastest convergence to the top performing architectures in all cases. The black dashed line is to facilitate the comparison of runtime taken to reach a certain test error among different variants. 191

8.9 Test accuracy of the subnetwork recommended by differentiable NAS methods over the search epochs. Original DARTS, DrNAS (in green) use the gradient of validation loss to update the architecture parameters but their variants (DARTS-TSE and DrNAS-TSE) (in red) uses that of our estimator computed over 100 mini-batches. TSE help mitigate the overfitting of DARTS on NB201. 195

8.10 Rank correlation (with final test *accuracy*) performance of TSE (yellow) and TSE-E (blue), and those of validation losses (pink), SoVL (solid) and SoVL-E (dash dot), as well as that of final test loss (black) for architectures in NB201 on three image datasets. Note the correlation performances of final test loss and SoVL-E near the end of training get surprisingly poor for CIFAR10/100. 197
8.11 Training losses, validation losses and validation accuracies of three example architectures on CIFAR100. The average of the training losses, validation losses and validation accuracies over the final 10 epochs is presented in the subcaption of each architecture. 199
8.12 Rank correlation performance of our TSE (yellow), the sum over training accuracy, SoTAcc (blue), the sum over validation losses, SoVL (pink), the sum of validation accuracy, SoVAcc (green) as well as their summing over the recent E epoch counterparts (dash dot) for 5000 random architectures in NASBench-201 on three image datasets. 200
8.13 Rank correlation performance of the sum of training losses over E most recent epochs (TSE-E) on the 20-cell DARTS dataset. Different E values include those < 1 are investigated for 100 random architectures in DARTS search space under three different evaluation set-ups. In all three settings, $E = 1$ again is the best choice. 200

A.1 Log median absolute error in approximating the entropy of a Gaussian mixture. 213
A.2 Log median fractional error in approximating the entropy of a gaussian mixture. 214
A.3 Compare the running times of the approximation methods. 215
A.4 Bayesian Optimisation (BO) on a 1D toy example with acquisition functions computed by different approximation methods. In the top subplot, the red dash line is the unknown objective function, the black crosses are the observed data points, and the blue solid line and shaded area are the posterior mean and variance, respectively, of the GP surrogate that we use to model the latent objective function. The coloured triangles are the next query point recommended by the BO algorithms, which correspond to the maximiser of the acquisition functions in the bottom subplot. In the bottom plot, the red solid line, black dash line, and green dotted line are the acquisition functions computed by Quad, MM, and MEMe using 10 Legendre moments, respectively. 222

A.5 Performance of various versions of FITBO on 2 benchmark test problems: (**a**) Michalewicz-5D function and (**b**) Hartmann-6D function. The immediate regret (IR) on the y-axis is expressed in the logarithm to the base 10. 222
A.6 A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO on ack-10 over *evaluation time, t*. . 224
A.7 A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO on ack-10 over *number of evaluations, N*. 225
A.8 Predictive vs ground-truth validation error of GPWL with additive kernel on NB101-CIFAR10 and RWNN-Flower102 in log-log scale. Error bar denotes ±1 SD from the GP posterior predictive distribution. 231
A.9 Motif discovery on NB101 and CIFAR100 and ImageNet120 tasks of NB201. Note that since NB101 is trained on CIFAR10 only, it is not possible to show the results transferred on another task. All symbols and legends have the same meaning as in Figure 7.2 in Section 7.3.3. 232
A.10 Computed motifs and ground-truth optimal cells for all 3 tasks of NB201. Note that optimal architecture cell for CIFAR10 contains motifs 1 and 3, optimal architecture cell for CIFAR100 contains motif 3 and optimal architecture cell for ImageNet120 contains motif 2. . 233
A.11 Median validation error on NAS-Bench datasets with *deterministic* observations from 20 trials. Shades denote ±1 standard error. . . . 234
A.12 Median validation error on NAS-Bench datasets with *noisy* observations from 20 trials. Shades denote ±1 standard error. 235
A.13 Mean and 5 standard error of training losses and validation losses on all architectures on different NB201 image datasets. (a) shows the training curves and (b) shows the number of architectures whose training losses go below 0.1 as the training proceeds. Many architectures reach very small training loss in the later phase of the training on CIFAR10 and CIFAR100, thus may overfitting on these two datasets. But all the architectures suffer high training losses on ImageNet120, which is a much more challenging classification task, and none of them overfits. 236

List of Tables

3.1 Runtime of evaluating PI, GP-UCB and FITBO-MM at 100 2D inputs using M hyper-parameter samples (Unit: Second). 47
3.2 Runtime of evaluating PI, GP-UCB and FITBO-MM for 100 test inputs of dimension d with M = 400 (Unit: Second). 47

4.1 Test accuracy (%) on CIFAR-10 after training the best model chosen by various asynchronous BO methods for 80 epochs 72

5.1 Continuous and categorical input range of the synthetic test functions 91
5.2 Continuous and categorical input ranges of the real-world problems 92
5.3 Mean and standard error of the predictive log likelihood of the CoCaBO and the One-hot BO surrogates on synthetic test functions. Both models were trained on 250 samples and evaluated on 100 test points. The CoCaBO surrogate can model the function surface better than the One-hot surrogate as the number of categorical variables increases. 93

6.1 Attack results on 50 random MNIST images by using $d^r = 14 \times 14 \times 1$. Q denotes the query count. ASR denotes attack success rate. The standard errors are in parentheses. 121
6.2 Attack results on 50 randomly selected CIFAR10 images with $d^r = 14 \times 14 \times 3$. Q denotes the query count. ASR denotes attack success rate.The standard errors are in parentheses. 122
6.3 Attack results on 50 randomly ImageNet images with random target label under a query budget of 2000. Q denotes the query count. ASR denotes attack success rate. The standard errors are in parentheses. ZOO fails to make any successful attack within this budget. 122
6.4 Summary of attack results on CIFAR10 for different decomposition learning method. $d_r = 14 \times 14 \times 3$ which is further decomposed into 12 subspaces of $d_s = 49$. Q denotes the query count. ASR denotes attack success rate. The standard errors are in parentheses. 127
6.5 Key statistics of the TU datasets used. 133

6.6 Validation accuracy of the victim models on the TU datasets before (clean) and after various attack methods. Results shown in mean ± 1 standard deviation across 3 trials. 133

7.1 Regression performance in terms of Spearman's rank correlation for different graph kernels. WL graph kernel adopted by us consistently outperforms other kernels across search spaces while retaining modest computational costs. 159
7.2 Performances on CIFAR10. GPU days do *not* include the evaluation cost of the final architecture; NAS-BOWL results from 4 random seeds on a single NVIDIA GeForce RTX 2080 Ti. 165

8.1 NAS search spaces used. The true test accuracy of architectures from each search space is obtained after training with SGD on the corresponding image datasets for T_{end} epochs. N_{total} denotes the total possible architectures exist in the search space and $N_{samples}$ denotes the number of architectures we sample/generate for our experiments. 181
8.2 Results of performance estimators in one-shot NAS setting over 3 supernetwork training initialisations. For each supernetwork, we randomly sample 500 random subnetworks for DARTS and 200 for NB201, and compute their TSE, Val Acc, SoVL, Tlmini after inheriting the supernetwork weights and training for B additional minibatches. *Rank correlation* measures the estimators' correlation with the rankings of the true test accuracies of subnetworks when *trained from scratch independently*, and we compute the *average test accuracy of the top 10 architectures* identified by different estimators from all the randomly sampled subnetworks. 192

A.1 Mean fractional error in approximating the entropy of the mixture of M Gaussians using various methods. 220
A.2 Mean runtime of approximating the entropy of the mixture of M Gaussians using various methods. 220
A.3 Mean and s.t.d. of the log(regret) after 50 steps of asynchronous BO. 226
A.4 Mean and s.t.d. of the log(regret) after 75 steps of asynchronous BO. 227
A.5 Mean and s.t.d. of the log(regret) after 100 steps of asynchronous BO. 228
A.6 Summary of attack results on 7 MNIST images. Q denotes the query count. *ASR* denotes attack success rate. The standard errors are in parentheses. 229
A.7 Summary of attack results on 27 CIFAR10 images. Q denotes the query count. *ASR* denotes attack success rate. The standard errors are in parentheses. 230

A.8 Regression performance (i.t.o rank correlation) of additive kernels . 230

B.1 Search spaces for ResNet and ResNeXt. Each network is formed of three stages and for each of the stage i, there are d_i the number of blocks, w_i number of channels per block. For ResNeXt, we also need to decide on the bottleneck width ratio r_i and the number of groups g_i per stage. The total number of possible architectures is $(dw)^3$ and $(dwrg)^3$ for ResNet and ResNeXt. 246

1 Introduction

1.1 Motivation

Machine learning models have been successfully applied in many domains, ranging from robotics (Stone and Veloso, 2000), material science (Butler et al., 2018) to biology (Jumper et al., 2021) and medicine (Rajkomar et al., 2019). However, their performance heavily depends on the appropriate choices of model architectures and hyper-parameters. Designing or selecting a model often requires strong expertise and tuning its hyper-parameters is traditionally done via laborious trial and error. These manual processes often lead to results that are suboptimal or not scientifically reproducible (Hinz et al.). The difficulty of model design and hyper-parameter optimisation is exacerbated by the fact that the complexity of machine learning models grows rapidly over the years and many of them act like black boxes which are hard to analyse and expensive to evaluate (Golovin et al., 2017). Moreover, the heavy reliance on the inputs from human experts also hinders the deployment of such models on many new or niche applications where prior knowledge or experience are highly limited. Therefore, the rising demand for machine learning models, together with the supply shortage and potential limitations of human experts, creates the need for methods that automatically design, configure and/or deploy algorithms to perform well for specific applications. This motivating research direction of

automated machine learning is called AutoML and aims to empower non-experts to apply machine learning tools for their own problems. The scope of AutoML covers the entire end-to-end pipeline from data preparation to model deployment or even result analyses/visualisation (Hutter et al., 2019). In this thesis, we focus on three core processes/tasks in the scope: hyper-parameter optimisation, neural architecture search (i.e. model discovery) and model robustness assessment, and the key technique that we work on to develop efficient solutions is Bayesian optimisation (BO)

BO is a powerful global optimisation technique, which is particularly useful when the objective function does not provide gradients and for which each function evaluation/query is very expensive (Brochu et al., 2010b; Shahriari et al., 2016b). There are two key components in BO. First, we need to define a probabilistic surrogate model to model the unknown objective problem. A popular choice for this is a Gaussian process (GP) due to its expressiveness and well-calibrated uncertainty estimates. Second, based on the probabilistic model, we need to design a utility function, named acquisition function (actually all expected utility), to balance the exploitation and exploration during the search. The next point to evaluate is then acquired/selected by maximising the acquisition function. Essentially, BO transfers the challenging problem of optimising an expensive-to-evaluate black-box objective to a much easier one of optimising a cheap-to-evaluate self-designed/known acquisition function.

The most fundamental and common problem in AutoML is to automatically optimise the hyper-parameters of machine learning models (Hutter et al., 2019). The high computational costs associated with hyper-parameter evaluation and the complex interaction between hyper-parameters and the model performance make it an ideal problem to adopt BO (Hutter et al., 2019; Shahriari et al., 2016b) and since then, BO has demonstrated state-of-the-art performance in a wide range of hyper-parameter tuning tasks (Snoek et al., 2012b; Bergstra et al., 2011b, 2013; Hutter et al., 2011b; Klein et al., 2016a). In comparison to other approaches such as grid search, random search (Bergstra and Bengio, 2012; Li et al., 2017), evolutionary algorithms (Hansen and Ostermeier, 2001; Real et al., 2017a) and

reinforcement learning (Zoph and Le, 2016; Baker et al., 2016), BO can often achieve satisfactory results with many fewer queries of the objective model and thus, is more cost-efficient. Moreover, BO also enjoys the additional merit of enabling the incorporation of expert knowledge and key characteristics of the objective problem to deign a more informative prior model (Hennig et al., 2015) and thus guide the hyper-parameter search. With these advantages, BO has become the backbone technique for hyper-parameter tuning in most AutoML packages/platforms such as Google Vizier (Golovin et al., 2017), Amazon SageMaker (Perrone et al., 2021), Microsoft NNI [1] and Huawei VEGA (Wang et al., 2020) while being gradually/recently applied to other AutoML processes such as neural architecture search (Kandasamy et al., 2018b; Shi et al., 2019; Ma et al., 2019a; White et al., 2021a; Ru et al., 2021).

1.2 Contribution

Despite the initial success of BO methods on hyper-parameter optimisation, existing BO methods still suffer many limitations which prevent their usage on a wider range of AutoML tasks: there remains much room for improvement in the acquisition function design and parallel computing resource usage, and better solutions are needed for practical problems involving search spaces of high-dimensions, mixed input types or graph-like structure. Our research therefore focuses on further advancing the *efficiency* and *performance* of BO methods for hyper-parameter optimisation while extending BO methods to be a highly competitive candidate for other AutoML processes such as neural architecture search and model robustness assessment. At a high level, we hope our research endeavours can contribute to solving a large variety of challenges in the AutoML pipeline by improving and using the principled tool of BO.

Specially, our contributions are as follows:

1. **Accelerating Information-theoretic Acquisition Function:** Acquisition function is a core component of BO. Compared to traditional acquisition

[1] nni.readthedocs.io

functions such as Probability of Improvement (PI) (Dixon and Szegö, 1978; Jones et al., 1998) and Expected Improvement (EI) (Močkus et al., 1978; Jones et al., 1998), the more recently proposed information-theoretic family offers a fresh perspective to efficiently select the query locations that maximise our information gain about the unknown global minimiser (Hennig and Schuler, 2012b; Hernández-Lobato et al., 2014; Wang and Jegelka, 2017a) and thus enjoys higher robustness to query noise (objective evaluation is noisy) and better exploration when applied for AutoML processes. Despite these merits, prior information-theoretic approaches (Hennig and Schuler, 2012b; Hernández-Lobato et al., 2014; Wang and Jegelka, 2017a) require many approximations in implementation, introduce often-costly computational overhead due to the need for sampling the global minimum/minimiser. We develop a fast information-theoretic BO method, FITBO, that reduces the expensive process of sampling the global minimum/minimiser to a much more efficient process of sampling one additional surrogate hyper-parameter, thus overcoming the speed bottleneck of information-theoretic approaches. We demonstrate empirically that FITBO inherits the competitive performance associated with information-theoretic methods, while being as fast to compute as simpler acquisition functions, such as EI and PI. This makes information-theoretic approaches a practical option for potential usage on AutoML.

2. **Asynchronous Batch Selection:** Whilst standard BO [2] may be sufficient for many applications, it is often the case that multiple model configurations can be evaluated concurrently in AutoML tasks given the availability of parallel computing resources. This leads to the development of synchronous batch/parallel BO algorithms (Ginsbourger et al., 2010a; Azimi et al., 2010; Ginsbourger et al., 2011; Azimi et al., 2012; Contal et al., 2013a; Chevalier and Ginsbourger, 2013; González et al., 2016b; Shah and Ghahramani, 2015a; Wu and Frazier, 2016a; Rontsis et al., 2017; Wang et al., 2017; Hernández-Lobato

[2] standard BO or sequential BO means that at each optimisation step, only one query location is proposed and evaluated.

et al., 2017; Kandasamy et al., 2018a), which, at each optimisation iteration, recommend a batch of k configurations to be evaluated simultaneously. In cases where the evaluation runtime of all configurations are roughly equal, this synchronous setting is usually sufficient, but, if the runtimes in a batch vary widely, this will lead to inefficient utilisation of our parallel computing hardware – we need to wait for the worker evaluating the slowest configuration in the batch to finish before starting the next BO iteration, leaving the other workers idle. We address this problem by developing an asynchronous approach, Penalising Locally for Asynchronous BO on k workers (PLAyBOOK), which immediately assigns a new configuration to the free worker when it completes the previous/current evaluation. PLAyBOOK uses penalisation-based strategies to prevent redundant batch selection and harnesses the local variability features of the surrogate to more effectively explore the search space. We demonstrate that our proposed PLAyBOOK can outperform its synchronous variants both in terms of wall-clock time and sample efficiency, particularly for larger batch sizes. This renders PLAyBOOK a competitive batch BO method for performing resource-efficient AutoML.

3. **Handling Mixed Input Types:** Many practical optimisation problems including those in AutoML processes often involve a mixture of *multiple* continuous and *multiple* categorical variables. However, having a mixture of categorical and continuous variables presents significant challenges for conventional BO methods which mainly focus on continuous input spaces. Prior solutions, like converting categorical variables into continuous ones via one-hot encoding (authors, 2016), severely increases the dimension of the search space, while separate modelling of category-specific data (Gopakumar et al., 2018; Nguyen et al., 2019) is query-inefficient. Moreover, these work-arounds are not scalable to practical applications involving multiple categorical variables with multiple possible values. We present a novel optimisation framework, named Continuous and Categorical BO (CoCaBO), which combines the strengths of multi-armed bandits and GP-based BO to

search over both categorical and continuous inputs. We model this mixed-type space using a tailored GP kernel to allow sharing of information across multiple categorical variables; this allows CoCaBO to leverage all available query data efficiently. We further extend CoCaBO to handle high-dimensional problems by modifying the kernel design and incorporating the concept of local trust region (Eriksson et al., 2019). Our methods achieve superior performance over existing approaches on both synthetic and real-world AutoML tasks with multiple categorical and continuous inputs.

4. **BO for Assessing Model Robustness:** The widespread adoption of deep learning models in many important real applications, fuelled by the rise of AutoML, has led to ever-growing costs and risks associated with potential model failure. This motivates the study of adversarial attacks in order to assess the robustness of such models, especially those found via AutoML. In this thesis, we focus on the a highly practical adversarial attacks setting where 1) the attacker has little knowledge about the target model and can only interact with the model by querying it, i.e. black-box setting, and 2) the maximum query number of limited due to the high inherent cost (computational or monetary) of model evaluation. Prior black-box adversarial attacks which rely on substitute model training (Papernot and McDaniel, 2016; Su et al., 2018), gradient estimation (Chen et al., 2017; Ilyas et al., 2018; Tu et al., 2018) or genetic algorithms (Alzantot et al., 2018), often require an excessive number of queries. Although techniques like searching for adversarial perturbations in a low-dimensional latent space (search dimensionality reduction) have been adopted to improve query efficiency (Chen et al., 2017; Tu et al., 2018; Alzantot et al., 2018), learning the effective dimensionality of the latent search space can be challenging by itself, and has not been investigated by the prior works. In view of these limitations, we introduce a new query-efficient black-box attack method, BayesOpt attack, which uses BO with GP surrogate models to find the successful adversarial example for state-of-the-art image classifiers and is capable of dealing with high-dimensional image inputs. We

also proposes the use of Bayesian model selection to learn the optimal degree of search dimensionality reduction, which can be incorporated naturally into our attack procedure, leading to efficient optimisation over both adversarial perturbation and the latent search space dimension. Empirical evidence shows that our BayesOpt attack can achieve comparable success rates with 2-5 times fewer queries compared to previous state-of-the-art black-box attacks, while finding adversarial examples that are closer to the original image (in terms of average L2 distance). Built on the BayesOpt attack, we further develop the first BO-based black-box adversarial attack for graph classification models, and demonstrate its effectiveness and flexibility on a variety of graph-based learning tasks under different attack constraints. We believe that BO-based adversarial attacks can be a competitive alternative for efficient assessment of the machine learning model robustness.

5. **BO for Neural Architecture Search:** Recently, a highly popular research direction under AutoML is Neural Architecture Search (NAS) which aims to automate the design process of good neural network architectures for a given task/dataset. Neural architectures found via different NAS strategies have achieved impressive performance outperforming human experts' design on a variety of tasks (Real et al., 2017a; Zoph and Le, 2016; Cai et al., 2019; Liu et al., 2018b; Real et al., 2019; Pham et al., 2018; Zoph et al., 2018b; Xie et al., 2018). NAS is very similar to hyperparameter optimisation: both can be formulated as a black-box optimisation problem where the objective evaluation is very expensive (Elsken et al., 2018; Kandasamy et al., 2018b; Shi et al., 2019). However, despite the similarity, conventional BO methods cannot be applied directly to NAS because the popular NAS search spaces are non-continuous and graph-like (Zoph et al., 2018b; Ying et al., 2019; Dong and Yang, 2020b); each possible architecture can be represented as an acyclic directed graph whose nodes correspond to operation units/layers and the edges represent connection among these units (Ying et al., 2019). To accommodate such search space, prior works either depend on tailored

encoding schemes (Ying et al., 2019; Kandasamy et al., 2018b; White et al., 2021a) or resort to the use of graph neural networks (Shi et al., 2019; Ma et al., 2019a); the former is not scalable to large architectures and ignore the implicit topological structure of architectures while the latter requires additional hyperparameter tuning and a large amount of observed data which is particularly expensive to obtain in NAS. We propose a neat BO-based strategy for NAS, named NAS-BOWL. NAS-BOWL combines the Weisfeiler-Lehman (WL) graph kernel (Shervashidze et al., 2011) with a GP surrogate to capture the topological structure of architectures without having to explicitly define a GP over high-dimensional vector spaces. Building on our proposed framework, many GP-based BO techniques previously developed for hyper-parameter optimisation (e.g. those deal with the multi-objective or transfer learning settings) can be easily re-deployed to tackle the more challenging NAS problems. More importantly, during the search, our NAS-BOWL also learns human interpretable architecture sub-features and their corresponding effects on the architecture performance; this marks the first step towards interpretable NAS whereby NAS algorithms can not only return the optimal architecture but also feedback interpretable insights and guidances on good architecture design to help human practitioners better understand this topic. We show the superior query-efficiency of our method over existing NAS approaches on a diverse set of NAS search spaces and demonstrate the usefulness of the interpretable sub-features via both qualitative and quantitative analyses.

6. **Efficient Generalisation Performance Estimation:** Reliably estimating the generalisation performance of a proposed configuration/model has always been the major computational bottleneck in many AutoML processes, such NAS (Elsken et al., 2018) and is crucial to their success. Traditional approaches face various limitations: training each model to completion is prohibitively expensive, early stopped validation accuracy may correlate poorly with fully trained performance, and predictor-based estimators require large training sets. In this thesis, we target at the domain of NAS and propose a simple

model-free method, Training Speed Estimation (TSE), which provides a reliable yet computationally cheap estimate of the generalisation performance of architectures based on a direct measure of training speed. Our estimator is theoretically motivated by the connection between generalisation and training speed, and is also inspired by the marginal likelihood under the Bayesian setting. We find that on various NAS search spaces that our TSE estimator consistently outperforms many other alternatives in achieving better correlation with the true test performance rankings. We further show that our estimator can be easily incorporated into both query-based and one-shot NAS methods, including BO, to improve the speed and/or quality of the search. We believe our reliable yet efficient performance estimator can significantly reduce the computation and time required for running AutoML processes, making them more affordable to potential users with limited budgets.

1.3 Thesis Structure

The remainder of this thesis is structured as follows. In Chapter 2, we provide basic backgrounds of GP and BO, as well as give an overview of various research directions in BO ranging from acquisition function design, non-continuous input space to multi-objective problems. In Chapter 3, we present our work on speeding up the information-theoretic acquisition functions, followed by our work on developing asynchronous batch selection for parallel BO methods in Chapter 4 as well as our proposed BO solutions for optimisation problems involve a mixture of *multiple* continuous and *multiple* categorical variables in Chapter 5. In Chapter 6, we showcase our use of BO for performing query-efficient black-box adversarial attacks on image classifiers and graph machine learning models. In Chapter 7, we present our work on interpretable NAS which leverages WL graph kernel in BO to do architecture search on graph space while learning interpretable sub-features together with their impact on architecture performance. In Chapter 8, we discuss a diverse set of performance estimation methods in the example context of NAS and show the competitive performance of our cheap model-free estimator, TSE. Finally,

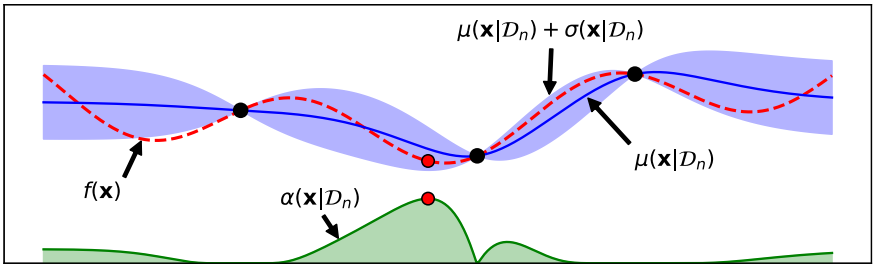
in Chapter 9, we conclude all our works, and suggest interesting and important directions for future exploration.

2

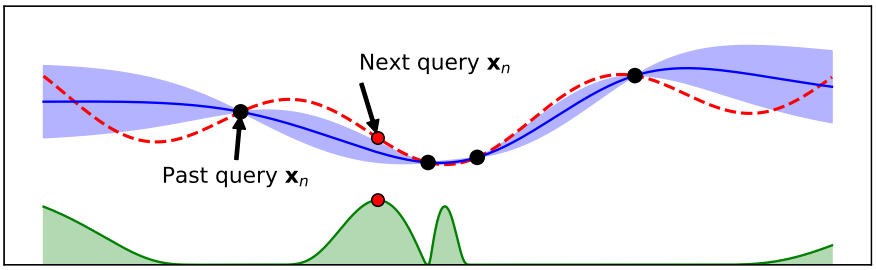
Background on Bayesian Optimisation

This chapter provides the background materials and an overview of the recent advancements in Bayesian optimisation (BO) which is the main theme of this thesis. Specifically, in Section 2.1, we provide the basics of Gaussian process (GP) as it is the most popular type of surrogate model in the BO literature and is also the one adopted in all our BO works. We then review different types of acquisition functions for BO in Section 2.2, followed by an overview over various extensions such as batch BO, high-dimensional BO and multi-objective BO in Section 2.3. BO is a power tool for finding the global optimum of a black-box objective function. It is particularly useful when the unknown objective function is expense to evaluate and only noisy observations of the function value at queried points can be obtained (Brochu et al., 2010b).

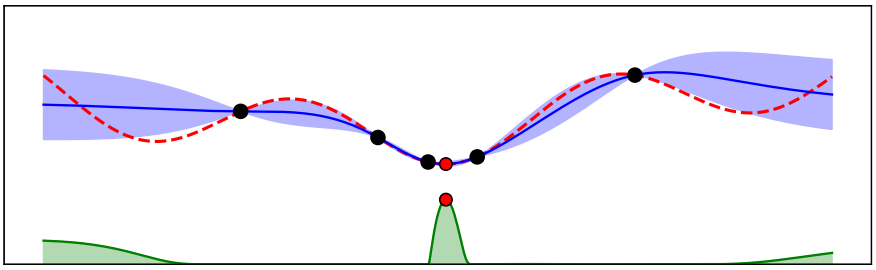
Assume we want to minimise an objective function with BO. We first define a probabilistic surrogate model to represent our prior belief over the unknown objective function. We then update this probabilistic/prior model by incorporating query observations to obtain a Bayesian posterior which reflects our updated beliefs about the objective function based on data observed so far. Based on the posterior model, we design an acquisition function to evaluate the utility of potential test locations by trading off exploitation (where the posterior mean of the model is low) and exploration (where the posterior variance of the model is high). The



(a) t=3



(b) t=4



(c) t=5

Figure 2.1: Demonstration of BO on a 1D objective function. The red dashed line represents true objective function.The solid black line and blue shade represents the mean and ± standard deviation of the prediction by the surrogate model (GP in this case) that approximates the objective function. The green line denotes the acquisition function whose value is high at locations where the surrogate model gives a low predictive mean (exploitation) and/or high predictive variance (exploration). This figure is inspired by a similar figure in (Brochu et al., 2010b).

next query location is then determined by maximising the acquisition function (Shahriari et al., 2016b). The procedures of BO is summarised in Algorithm 1 (Brochu et al., 2010b). And Figure 2.1 demonstrates three iterations of Bayesian optimisation on a 1D objective function. In essence, BO has two key components:

1) the probabilistic surrogate that models the unknown objective function and 2) the acquisition function that evaluates the utility of possible query points, for which we will go through in more details in the next two sections.

Algorithm 1 General algorithm of Bayesian optimisation

Input: Initial observation data $\mathcal{D}_t$, A black-box function f, Total number of BO steps T
Output: The best recommendation about the global optimiser $\hat{\mathbf{x}}_T$
for $t = 1, \ldots, T$ **do**
  Select the next $\mathbf{x}_{t+1}$ by maximising acquisition function $\alpha(\mathbf{x}|\mathcal{D}_t)$
  Evaluate the objective function at $\mathbf{x}_{t+1}$ to obtain y_{t+1}
  $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \cup (\mathbf{x}_{t+1}, y_{t+1})$
  Update the surrogate model
end for

2.1 Gaussian Process

Many regression models have been used as the surrogate models in BO research, such as a random forest (Hutter et al., 2011b), a deep ensemble (White et al., 2021a) as well as a Bayesian neural network (Springenberg et al., 2016a) and its various approximations (Snoek et al., 2015; Hernández-Lobato et al., 2017). However, GPs remain as the most common choice due to some favourable properties. First and foremost, GPs enjoy the desirable identities of a Gaussian distribution i.e. its marginals and conditionals remain Gaussian and thus can be analytically calculated (Rasmussen and Williams, 2006b). This leads to analytically defined posterior distribution if we assume Gaussian likelihoods as well as tractable acquisition functions. Second, GPs provide better calibrated uncertainty estimates than the other alternatives because in contrast to the analytic uncertainty (posterior variance) of GPs, other surrogate choices often need to estimate uncertainty measures from limited number of empirical samples (sample variance). Given the quality of uncertainty estimate is crucial for the trade-off between exploration and exploitation in BO, GPs tend to deliver better empirical performance compared to other alternatives (Golovin et al., 2017; Snoek et al., 2012a) especially when the number of queries are small. Third, GPs are highly expressive and data efficient in modelling

the objective functions. This is particular suitable for BO applications where the objective is assumed to be very costly to evaluate and thus the surrogate model can only train on a relatively small number of query/observation data to perform regression. Note that the common small-data regime of BO also nicely circumvents the scalability issue of GPs [1], except in the transfer-learining or multi-task scenarios (Wistuba et al., 2016, 2018).

The GP is defined as a collection of random variables, any finite number of which have a multivariate Gaussian distribution (Rasmussen and Williams, 2006b). It is a non-parametric model which can be viewed as a distribution over functions or over infinite number of variables. A GP is fully specified by a mean function $\mu(\cdot)$ and a covariance function (also known as the kernel function) $k(\cdot, \cdot)$ (Rasmussen and Williams, 2006b):

$$f(\mathbf{x}) \sim \mathcal{GP}\left(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{2.1}$$

At an arbitrary input $\mathbf{x} = \hat{\mathbf{x}}$, a GP returns the mean and variance of a normal distribution over the possible values of $f(\hat{\mathbf{x}}) \sim \mathcal{N}\left(\mu(\hat{\mathbf{x}}), k(\hat{\mathbf{x}}, \hat{\mathbf{x}}')\right)$. The mean function of the prior distribution provides an offset and specifies our inference far from observations. The kernel function affects the smoothness properties of the GP model (Brochu et al., 2010b) and thus determines the structure of the surrogate we can model. For simplicity, the common practice is to set the mean function to zero i.e. $\mu(\mathbf{x}) = 0, \forall \mathbf{x} \in \mathcal{X}$ (Rasmussen and Williams, 2006b) as we can simply reformulate the regression objective from f to $f - \mu(\mathbf{x})$. Thus, the GP prior is often solely determined by the covariance function.

After querying the objective function at n input locations, we obtain observation data $\mathcal{D}_f = \{(\mathbf{x}_i, f_i)\}_{i=1}^n$. As we impose a GP prior on the objective function, the function values $\mathbf{f}_n$ are assumed to be drawn from the multivariate normal distribution $p(\mathbf{f}_n) = \mathcal{N}\Big(\mathbf{f}_n; \mathbf{0}, K(\mathbf{X}_n, \mathbf{X}_n)\Big)$, where each element of the covariance matrix K is defined as $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

[1] The computation cost of GPs is $\mathcal{O}(n^3)$ where n is the number of observation data.

The joint distribution of the observed function values $\mathbf{f}_n$ and the function value at a test location $f = f(\mathbf{x})$ is also Gaussian (Rasmussen and Williams, 2006b) :

$$\begin{bmatrix} \mathbf{f}_n \\ f \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} K(\mathbf{X}_n, \mathbf{X}_n) & K(\mathbf{X}_n, \mathbf{x}) \\ K(\mathbf{x}, \mathbf{X}_n) & K(\mathbf{x}, \mathbf{x}) \end{bmatrix} \right). \tag{2.2}$$

By manipulating the joint distribution based on the analytic property of Gaussian conditional distributions, we obtain the predictive posterior distribution at the test point $\mathbf{x}$:

$$p(f|\mathbf{x}, \mathbf{X}_n, \mathbf{f}_n) = \mathcal{N}\Big(f; \mu_f(\cdot), K_f(\cdot, \cdot)\Big) \tag{2.3}$$

where

$$\mu_f(\mathbf{x}) = K(\mathbf{x}, \mathbf{X}_n) K(\mathbf{X}_n, \mathbf{X}_n)^{-1} \mathbf{f}_n \tag{2.4}$$

$$K_f(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{X}_n) K(\mathbf{X}_n, \mathbf{X}_n)^{-1} K(\mathbf{X}_n, \mathbf{x}). \tag{2.5}$$

However, in a real world situation, we do not have access to the true function values but only noisy observation of the function $y_i = y(\mathbf{x}_i) = f(\mathbf{x}_i) + \epsilon$ where ϵ is assumed to be an independently and identically distributed Gaussian noise with variance σ^2_{noise} $\Big($i.e. the likelihood function is also Gaussian $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}; \mathbf{f}, \sigma^2_{noise}\mathbf{I}) = \prod_i^n \mathcal{N}(y_i; f_i, \sigma^2_{noise})\Big)$ (Rasmussen and Williams, 2006b). In such case of noisy observation data $\mathcal{D}_y = \{(\mathbf{x_i}, y_i)\}_{i=1}^n = \{\mathbf{X}_n, \mathbf{y}_n\}$, the predictive distribution of the noisy observation y at the test point $\mathbf{x}$ has the form

$$p(y|\mathbf{x}, \mathbf{X}_n, \mathbf{f}_n) = \mathcal{N}\Big(y; \mu_y(\cdot), K_y(\cdot, \cdot)\Big) \tag{2.6}$$

where

$$\mu_y(\mathbf{x}) = K(\mathbf{x}, \mathbf{X}_n)\Big[K(\mathbf{X}_n, \mathbf{X}_n) + \sigma_n^2 \mathbf{I}\Big]^{-1} \mathbf{y}_n \tag{2.7}$$

$$K_y(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{X}_n)\Big[K(\mathbf{X}_n, \mathbf{X}_n) + \sigma_n^2 \mathbf{I}\Big]^{-1} K(\mathbf{X}_n, \mathbf{x}). \tag{2.8}$$

2.1.1 Kernel Functions

The choice of covariance or kernel function is essential for a GP because it affects the smoothness property of the functions we can model (Shahriari et al., 2016b). Take the example of the squared exponential kernel function with automatic relevance determination (SE-ARD), which is a popular kernel choice used to model smooth functions. The analytical form of the SE-ARD kernel function is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \exp\left[-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \Lambda_l^{-2}(\mathbf{x}_i - \mathbf{x}_j)\right] \tag{2.9}$$

where σ is the output scale and Λ_l is a diagonal matrix of d characteristic length scales. The automatic relevance determination allows us to learn the characteristic length scales in each input dimension, thus measuring the influence of each dimension on the function value. Characteristic length scales measure how far we need to move between two points in input space for their function values to be uncorrelated (Rasmussen and Williams, 2006b). As shown in Figure 2.2, if the characteristic length scale of a certain input dimension is very large, the function value will become almost independent of that dimension . On the other hand, a small characteristic length scale leads to more fluctuating function values.

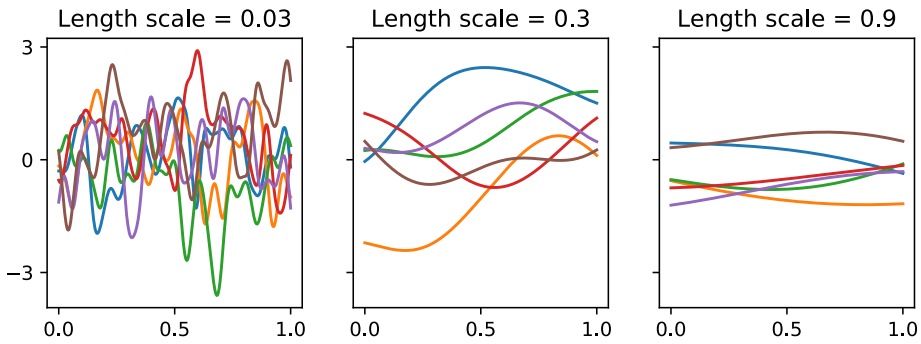


Figure 2.2: Random samples generated by a GP prior with the SE-ARD kernel function and different characteristic length scales. The smaller length scale leads to more wiggly function.

A kernel function is valid if its covariance/Gram matrix K, whose element is $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, is positive semidefinite and a valid kernel function can be expressed as an inner product in the feature space $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ (Bishop, 2006).

It can be easily proven that a combination of (e.g. multiplying and/or adding) simple valid kernel functions (Rasmussen and Williams, 2006b) is also a valid kernel. Thus, we can leverage this to efficiently construct new kernel functions as we will show in Chapter 5.

2.1.2 GP Hyper-parameter Treatment

Hyper-parameters $\boldsymbol{\theta}$ of a GP often refer to the kernel parameters, such as output scale and characteristic length scales, as well as the observation noise variance. As illustrated above, different hyper-parameter values will heavily affect the inference performance of a GP model (Rasmussen and Williams, 2006b). There are mainly two approaches to tune these hyper-parameters. First, we could find one set of optimal hyper-parameter values via maximum likelihood estimation (MLE) or maximum a posterior estimation (MAP). Second, we could marginalise over all possible sets of hyper-parameters to perform inference.

Maximum Likelihood Estimation

The maximum likelihood estimation (MLE) approach, as the name suggested, is to learn an optimal hyper-parameter values $\boldsymbol{\theta}$ by maximising the log marginal likelihood function $\log p(\mathcal{D}_y|\boldsymbol{\theta})$. The log marginal likelihood function for a zero-mean GP model has the analytic form

$$\log p(\mathcal{D}_y|\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}_n^T[K_{\mathbf{y}}^{\boldsymbol{\theta}}(\mathbf{X}_n, \mathbf{X}'_n)]^{-1}\mathbf{y}_n - \frac{1}{2}\log|K_{\mathbf{y}}^{\boldsymbol{\theta}}(\mathbf{X}_n, \mathbf{X}'_n)| - \frac{n}{2}\log 2\pi \quad (2.10)$$

where $K_{\mathbf{y}}^{\boldsymbol{\theta}}(\mathbf{X}_n, \mathbf{X}'_n) = K^{\boldsymbol{\beta}}(\mathbf{X}_n, \mathbf{X}'_n) + \sigma^2_{\text{noise}}\mathbf{I}$ is the covariance matrix for the noisy observation data $\mathcal{D}_y = \{\mathbf{X}_n, \mathbf{y}_n\}$. $K_{\mathbf{y}}^{\boldsymbol{\theta}}(\mathbf{X}_n, \mathbf{X}'_n)$ depends on the hyper-parameters $\boldsymbol{\theta} = [\boldsymbol{\beta}, \sigma_{\text{noise}}]$ which include both the kernel hyper-parameters $\boldsymbol{\beta}$ as well as the Gaussian noise standard deviation σ_{noise}.

With reference to Equation 2.10, the log marginal likelihood comprises three terms, the first term measures how well the GP model fits the data; the second term penalises excessive model complexity and the last term is a normalising constant (Rasmussen and Williams, 2006b). As the model complexity increases (i.e. the

second term decreases), we obtain a more flexible model which supports a larger class of functions/hypotheses and thus leads to better fitting of the data (i.e. the first term increases). Therefore, the maximum likelihood estimation inherently induces the trade-off between model fit and model complexity, leading to the finding of the simplest model that can well fit the data.

However, the log marginal likelihood function is generally a non-convex function with multiple local optima (Bishop, 2006). Thus, maximum likelihood estimation may lead to hyper-parameters corresponding to a bad local optimum, causing the problem of over-fitting. This is more likely to occur when the observation data set is small, which is the case for BO, and the local optima of marginal likelihood are quite close (Rasmussen and Williams, 2006b).

If we have some prior knowledge about hyper-parameters, we can combine the prior distribution over $\boldsymbol{\theta}$ with the marginal likelihood to implement the maximum a posteriori estimation (MAP). By Bayes rule, the posterior over the hyper-parameters is

$$p(\boldsymbol{\theta}|\mathcal{D}_n) = \frac{p(\mathcal{D}_n|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\mathcal{D}_n|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}} \propto p(\mathcal{D}_n|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{2.11}$$

$$\Rightarrow \log p(\boldsymbol{\theta}|\mathcal{D}_n) \propto \log p(\mathcal{D}_n|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \tag{2.12}$$

Thus, maximising the log posterior over hyper-parameters is equivalent to maximising the sum of the log marginal likelihood and the log prior. The maximum a posteriori estimation can be viewed as the maximum likelihood estimation with regulation, thus alleviating the problem of overfitting. In real BO practice, we usually perform multi-start optimisation [2] for MLE or MAP to further alleviate the chance of being trapped into a poor local optima.

Marginalisation

Both maximum likelihood estimation and maximum a posteriori estimation are not desirable as they give point estimates and ignore our uncertainty about the hyper-parameters. In a fully Bayesian treatment of the hyper-parameters in GP inference,

[2] We conduct the local optimisation around multiple starting locations and pick the best of all the local optima found as the final choice.

we should consider all possible hyper-parameter values by marginalising the predictive posterior distribution with respect to the hyper-parameter posterior $p(\boldsymbol{\theta}|\mathcal{D}_n)$:

$$p(y|\mathbf{x}, \mathcal{D}_n) = \int p(y|\mathbf{x}, \mathcal{D}_n, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}_n) d\boldsymbol{\theta}. \tag{2.13}$$

Since complete marginalisation over hyper-parameters is analytically intractable, the integral must be approximated using the Monte Carlo method (Snoek et al., 2012b) by drawing M hyper-parameters samples $\{\boldsymbol{\theta}^{(j)}\}_{j=1}^{M}$ from $p(\boldsymbol{\theta}|\mathcal{D}_n)$, leading to the approximation:

$$p(y|\mathbf{x}, \mathcal{D}_n) \approx \sum_{j=1}^{M} p(y|\mathbf{x}, \mathcal{D}_n, \boldsymbol{\theta}^{(j)}) \tag{2.14}$$

2.2 Acquisition Functions

Acquisition functions are carefully designed to balance exploration and exploitation during the search. They would recommend the next query location where the function value predicted by the surrogate model is low (exploitation in the case of minimising the function) and/or the uncertainty about the prediction is large (exploration) (Brochu et al., 2010b). As compared to the objective function, the acquisition functions is often assumed to be much cheaper to evaluate or approximate and thus can be globally optimised (Shahriari et al., 2016b). In this section, we would introduce different categories of acquisition functions: 1) improvement-based acquisition functions, 2) optimistic acquisition functions and 3) the information-based acquisition functions. For consistent presentation in this Section, we assume that we want to maximise the objective function $\mathbf{x} = \arg\max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$.

2.2.1 Improvement-based Acquisition Functions

Probability of Improvement

An simple form of acquisition function developed early in 1964 is Probability of Improvement (PI) (Kushner, 1964). PI method measures the probability that a location $\mathbf{x}$ leads to an improvement upon the incumbent best value observed f^+. Here we aim to maximise the objective function so improvement means higher than

the maximum value observed so far $f^+ = \max_{i\in\{1,\dots,n\}} f(\mathbf{x}_i)$. This probability is computed in the form of a cumulative distribution function (Brochu et al., 2010b):

$$\alpha_{\mathrm{PI}}(\mathbf{x}|\mathcal{D}_n) = P(f(\mathbf{x}) \geq f^+) = \Phi\left(\frac{\mu(\mathbf{x} - f^+)}{\sigma(\mathbf{x})}\right) \tag{2.15}$$

where Φ is the standard normal cumulative distribution and $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ are the mean and standard deviation of the predictive posterior distribution at a test location $\mathbf{x}$. The next sample is then taken at the location with the maximum $\alpha_{\mathrm{PI}}(\mathbf{x}|\mathcal{D}_n)$.

PI acquisition function tends to be highly exploitation-oriented. To encourage more exploration, Kushner (Kushner, 1964) proposes to add a positive trade-off parameter ζ so that PI method would recommend the next point by maximising the probability of improving over $f^+ + \zeta$. However, the value of ζ is often set arbitrarily. If ζ is too small, the search will be highly local whereas if ζ is too high, the search will be global but slow to converge (Jones, 2001).

Expected Improvement

Expected Improvement (EI) acquisition function improves over PI method by also considering the magnitude of improvement (Dixon and Szegö, 1978) (Jones et al., 1998). In EI acquisition function, the utility is represented by an improvement function $I(\mathbf{x})$ which is defined as:

$$I(\mathbf{x}) = \max\{0, f(\mathbf{x}) - f^+\}. \tag{2.16}$$

$I(\mathbf{x})$ is positive only when the prediction at the test location leads to an improvement over the best value known thus far f^+. Otherwise, $I(\mathbf{x})$ is zero. The new query point is then recommended by maximising the expected improvement

$$\mathbf{x} = \arg\max_{\mathbf{x}} \mathbb{E}[I(\mathbf{x})|\mathcal{D}_n]. \tag{2.17}$$

The expectation can be computed analytically for a GP surrogate (Dixon and Szegö, 1978):

$$\begin{aligned}\alpha_{\mathrm{EI}}(\mathbf{x}|\mathcal{D}_n) &= \mathbb{E}[I(\mathbf{x})|\mathcal{D}_n] \\ &= \int_0^\infty I(\mathbf{x})\mathcal{N}\Big(I(\mathbf{x}); \mu(\mathbf{x}) - f^+, \sigma(\mathbf{x})^2\Big)\mathrm{d}I(\mathbf{x}) \\ &= \begin{cases} \Big(\mu(\mathbf{x}) - f^+\Big)\Phi\left(\dfrac{\mu(\mathbf{x}) - f^+}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\dfrac{\mu(\mathbf{x}) - f^+}{\sigma(\mathbf{x})}\right) & \text{if} \quad \sigma(\mathbf{x}) > 0 \\ 0 & \text{if} \quad \sigma(\mathbf{x}) = 0 \end{cases}\end{aligned}$$

where Φ and ϕ denotes the cumulative distribution function and the probability density function of the standard normal distribution respectively, and $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ are the mean and standard deviation of the predictive posterior distribution at a test location $\mathbf{x}$.

The two terms in EI acquisition function balances the trade-off between exploitation and exploration respectively. Similar to the PI case, Lizotte (Lizotte, 2008) proposes to incorporate a parameter ζ into EI acquisition function to encourage exploration. Based on experiments in (Lizotte, 2008), $\zeta = 0.01\sigma(\mathbf{x}$ works well in almost all cases.

Knowledge Gradient

Knowledge Gradient (KG) is initially proposed in (Frazier et al., 2009). Rather than only considering the function values at the previously queried/observed locations like PI and EI, KG focuses on the posterior (mean) over the entire search space and tend to do better on problems with noisy observations and derivative observations (Frazier et al., 2009; Frazier, 2018). The expression of KG is as follows:

$$\alpha_{\mathrm{KG}}(\mathbf{x}|\mathcal{D}_n) = \mathbb{E}\left[\mu^*_{n+1} - \mu^*_n | \mathbf{x}_{n+1} = \mathbf{x}\right]. \tag{2.18}$$

$\mu^*_n = \max_{\mathbf{x}\in\mathcal{X}} \mu(\mathbf{x}|\mathcal{D}_n)$ denotes the maximum of the current posterior mean $\mu_n(\mathbf{x})$ which is computed conditioned on the observation data so far $\mathcal{D}_n$. $\mu^*_{n+1} = \max_{\mathbf{x}\in\mathcal{X}} \mu(\mathbf{x}|\mathcal{D}_{n+1})$ is the maximum of the mean of the updated posterior $\mu_{n+1}(\mathbf{x})$ if we query at a new location $\mathbf{x}_{n+1}$ (i.e. conditioned on $\mathcal{D}_{n+1} = \mathcal{D}_n \cup (\mathbf{x}_{n+1}, f_{n+1})$.

Since the true function value f_{n+1} at $\mathbf{x}_{n+1}$ is unknown, Equation (2.19) computes the expected value with respect to the current posterior distribution.

Therefore, the next query location $\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}} \alpha_{\text{KG}}(\mathbf{x}|\mathcal{D}_n)$ will lead to the largest expected increase in the maximum of the posterior mean but the function value at $\mathbf{x}_{n+1}$ may not be better than the best value observed previously.

The computation of the expectation in Equation (2.19) is intractable, (Frazier et al., 2009) proposes to approximate $\alpha_{\text{KG}}(\mathbf{x}|\mathcal{D}_n)$ by sampling function values from the posterior distribution, $f_{n+1}^{(j)} \sim \mathcal{N}\left(\mu_n(\mathbf{x}_{n+1}), \sigma_n^2(\mathbf{x}_{n+1})\right)$:

$$\alpha_{\text{KG}}(\mathbf{x}|\mathcal{D}_n) \approx \frac{1}{M} \sum_{j=1}^{M} \left(\hat{\mu^*}_{n+1}^{(j)} - \mu_n^* \right) \tag{2.19}$$

where $\hat{\mu^*}_{n+1}^{(j)} = \max_{\mathbf{x} \in \mathcal{X}} \mu(\mathbf{x}|\mathcal{D}_n \cup (\mathbf{x}_{n+1}, f_{n+1}^{(j)}))$. A more computationally efficient way to compute and optimise KG is proposed in (Wu and Frazier, 2016a) and is based on stochastic gradient ascent.

2.2.2 Optimistic Acquisition Functions

Upper Confidence Bound

The concept of using confidence bounds for BO is first explored by Cox and John (Cox and John, 1992) in their Sequential Design for Optimisation(SDO) algorithm. SDO recommends the next point for evaluation by maximising the following acquisition function:

$$\alpha_{\text{UCB}}(\mathbf{x}|\mathcal{D}_n) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}). \tag{2.20}$$

which is a weighted sum of posterior mean $m(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$. The trade-off between exploitation and exploration is controlled by the user-defined parameter κ.

Srinivas et al. (Srinivas et al., 2009) uses a GP as the surrogate model and theoretically derive a schedule for the value of κ, which can lead to minimisation of the cumulative regret:

$$R_n = \sum_{i=1}^{n} f(\mathbf{x}^*) - f(\mathbf{x}_i) \tag{2.21}$$

where $\mathbf{x}^*$ is the true global maximiser and $\mathbf{x}_i$ is the query point selected via the acquisition function at i-th BO iteration. With this guideline, they define the GP Upper Confidence Bound (GP-UCB) whose acquisition function is a modified version of Equation 2.20:

$$\alpha_{\text{GP-UCB}}(\mathbf{x}|\mathcal{D}_n) = \mu(\mathbf{x}) + \sqrt{\beta_n}\sigma(\mathbf{x}) \tag{2.22}$$

where $\beta_n = 2c\log(n^{d/2+2}\pi^2/3\delta)$ for $c > 0$. d is the input dimension and n is the number of queries taken. Notably, the approach introduced in Srinivas et al. (2009) to derive the cumulative regret bounds for GP-UCB inspires and underlies most theoretical analyses in the BO literature.

2.2.3 Information-based Acquisition Functions

Entropy Search

Entropy Search (ES) (Hennig and Schuler, 2012b) introduces a new perspective to efficiently select the query points in BO. It guides our evaluations to locations where we can maximise the reduction in the uncertainty about the unknown optimum rather than to locations where we expect to obtain lower function values (Hennig and Schuler, 2012b). As a result, the acquisition function for ES has the form (Hennig and Schuler, 2012b):

$$\alpha_{\text{ES}}(\mathbf{x}|\mathcal{D}_n) = H[p(\mathbf{x}^*|\mathcal{D}_n)] - \mathbb{E}_{p(y|\mathcal{D}_n,\mathbf{x})}\Big[H\Big[p\Big(\mathbf{x}^*|\mathcal{D}_n \cup (\mathbf{x},y)\Big)\Big]\Big] \tag{2.23}$$

where $H[p(\mathbf{x})] = -\int p(\mathbf{x})\log p(\mathbf{x})\mathrm{d}\mathbf{x}$ and the expectation is taken with respect to the posterior distribution $p(y|\mathcal{D}_n, \mathbf{x})$.

The first differential entropy term in the above function embodies our current uncertainty about the unknown maximiser $\mathbf{x}_*$. The second term measures our expected uncertainty about $\mathbf{x}_*$ after querying an arbitrary point $(\mathbf{x}, y)$. Thus, the acquisition function (Equation 2.23), which is the mutual information between $\mathbf{x}_*$ and the next query point, represents the information gain about $\mathbf{x}_*$ and ES method would select the point that maximise this information gain.

However, an exact evaluation of the ES acquisition function (Equation 2.23) is only feasible after many approximations because the entropy terms cannot be

computed analytically and the optimisation of Equation 2.23 involves calculating $p\Big(\mathbf{x}^*|\mathcal{D}_n \cup (\mathbf{x}, y)\Big)$ for many different values of $\mathbf{x}$ and y (Hernández-Lobato et al., 2014). The approximation proposed by (Hennig and Schuler, 2012b) is based on discretisation of the continuous search space, which results in a high computational costs of $O(M^4)$ where M is the number of discrete representer points.

Predictive Entropy Search

In view of the difficulties of implementing ES method, Lobato et al.(Hernández-Lobato et al., 2014) propose a modified alternative, named Predictive Entropy Search (PES). PES method utilises the symmetric property of mutual information and rewrites Equation 2.23 as (Hernández-Lobato et al., 2014) :

$$\alpha_{\text{PES}}(\mathbf{x}|\mathcal{D}_n) = H[p(y|\mathcal{D}_n, \mathbf{x})] - \mathbb{E}_{p(\mathbf{x}^*|\mathcal{D}_n)}[H[p(y|\mathcal{D}_n, \mathbf{x}, \mathbf{x}^*)]] \tag{2.24}$$

where $p(y|\mathcal{D}_n, \mathbf{x}, \mathbf{x}^*)$ is the posterior distribution for y conditioned on the observation data $\mathcal{D}_n$, the test point $\mathbf{x}$ and the global maximiser $\mathbf{x}^*$. Different from ES acquisition function (Equation 2.23) which depends on the entropies of distributions on maximiser $\mathbf{x}^*$, the PES acquisition function (Equation 2.24) uses the entropies of predictive posterior distributions over output y which can be computed analytically or be approximated more easily (Hernández-Lobato et al., 2014).

PES has been shown to achieve better optimisation performance than ES while requiring lower computational cost to be implemented. However, the implementation of PES involves two separate sampling processes:

1. sampling hyper-parameters for marginalisation because PES treats hyper-parameters in a fully Bayesian way,

2. sampling the global maximiser $\mathbf{x}^*$ for approximating the second entropy term in the acquisition function.

The second sampling process not only contributes significantly to the computational burden of these information-based acquisition functions but also requires the construction of a good approximation for the objective function based on Bochner's

theorem (Hernández-Lobato et al., 2014), which limits the kernel choices to the stationary ones (Bochner, 2016). Moreover, the approximation for the intractable distribution $p(y|\mathcal{D}_n, \mathbf{x}, \mathbf{x}^*)$ is also very tedious and requires the first and second partial derivatives of the kernel function, further limiting our kernel choices to the ones whose derivatives are easy to compute or approximate. As a result, PES is still very slow to evaluate in comparison with traditional methods like PI, EI and GP-UCB. In addition, the fact that PES depends heavily on the samples of $\mathbf{x}^*$ and deals with the input space makes it less efficient for higher dimensional problems (Hoffman and Ghahramani, 2015; Wang and Jegelka, 2017a).

Output-space Predictive Entropy Search and Max-value Entropy Search

Output-space Predictive Entropy Search (OPES) (Hoffman and Ghahramani, 2015) and Max-value Entropy Search (MES) (Wang and Jegelka, 2017a) improve on PES by focusing on the information content in output space instead of input space. Therefore, they aim to select the next query point by minimising the entropy of the global maximum value $f(\mathbf{x}^*)$ instead of the global maximiser $f(\mathbf{x}^*)$. For simplicity, we denote $f(\mathbf{x}^*)$ as f^* and the acquisition functions of OPES and MES have the same form:

$$\alpha_{\text{MES}}(\mathbf{x}|\mathcal{D}_n) = H[p(y|\mathcal{D}_n, \mathbf{x})] - \mathbb{E}_{p(f^*|\mathcal{D}_n)}\Big[H\Big[p(y|\mathcal{D}_n, \mathbf{x}, f^*)\Big]\Big]. \tag{2.25}$$

Since OPES and MES deal with output space which is always 1-dimensional, they are more efficient than PES, especially in higher dimensional problems.

2.3 Extension Settings

2.3.1 Batch BO

The conventional BO algorithms (Algorithem 1) evaluate at only one query location $\mathbf{x}_{t+1}$ at each optimisation iteration and we call them the sequential BO methods. In many applications, these BO methods have been extended to recommend a batch of query points at each iteration so that we can take the advantage of parallel processing

resources (e.g. multiple GPUs in a server) to evaluate multiple query locations simultaneously. Algorithm 2 shows the general procedure of a batch BO method.

Algorithm 2 General algorithm of batch BO

Input: Initial observation data $\mathcal{D}_t$, A black-box function f, Total number of BO steps T
Output: The best recommendation about the global optimiser $\hat{\mathbf{x}}_T$
for $t = 1, \ldots, T$ **do**
  Select the next $\mathcal{B}_{t+1}$ by maximising acquisition function $\alpha(\mathcal{B}|\mathcal{D}_t)$
  Evaluate the objective function at $\mathcal{B}_{t+1} = \{\mathbf{x}_{t+1,j}\}_{j=1}^{b}$ to obtain $\{y_{t+1,j}\}_{j=1}^{b}$
  $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \cup \{(\mathbf{x}_{t+1,j}, y_{t+1,j})\}_{j=1}^{b}$
  Update the surrogate model
end for

Most of the current batch selection techniques can be categorised into two classes:

1. *Non-greedy* methods by which all the points in the batch are selected simultaneously,

2. *Greedy* methods by which the points in the batch are selected sequentially (one by one).

Some examples of batch BO algorithms that use non-greedy batch selection are q-EI method (Chevalier and Ginsbourger, 2013), which obtains the batch by maximising the approximated multi-points EI acquisition function, and Parallel Predictive Entropy Search (PPES) (Shah and Ghahramani, 2015b), which selects the batch that leads to the maximum information gain about the unknown global minimiser. In addition, Rontsis et al (Rontsis et al., 2017) derive a computationally tractable lower bound on multi-points EI acquisition function and solve it efficiently using distributionally ambiguous optimisation techniques to obtain the batch points. Wang et al. (Wang et al., 2017) propose a batch BO targeted at high-dimensional problems and use the determinantal point process (DPP) to select a diverse batch. Thompson sampling is another simple yet effective way to obtain batch points (Hernández-Lobato et al., 2017; Kandasamy et al., 2018a) and is more scalable to large batch sizes than other batch selection methods. For example, Hernández-Lobato et al. (2017) has applied Thompson sampling to collect up to 500 query points

in each single batch when searching for chemical molecules with high-throughput virtual screening. Different from the previous methods which all use a single type of acquisition function, a more recent technique proposed in Lyu et al. (2018) deploys an ensemble of multiple types of acquisition functions (PI, EI and UCB) and performs a multi-objective optimisation to define the Pareto front of the ensemble. The batch points are then selected from the Pareto front by random sampling.

As for the greedy batch selection, many batch methods (Ginsbourger et al., 2010b; Contal et al., 2013b; Desautels et al., 2014) exploit a nice feature of GPs: the GP posterior variance only depends on the function inputs and thus can be updated without evaluating at these inputs. Specifically, the two heuristics proposed in Ginsbourger et al. (2010b), the Kriging Believer (KB) and the Constant Liar (CL), greedily approximate the multi-points EI method. For both heuristics, after choosing a query location by optimising the EI acquisition function, the observation data is augmented with the chosen location and an estimator of the function value at that location (KB uses the posterior mean at that location while CL uses a constant value specified by the user) and the GP posterior is then updated with the augmented data. The optimisation-augmentation process is repeated untill all the b points in the batch are selected. The GP-batch UCB (BUCB) method developed in Desautels et al. (2014) works in a similar manner and selects the batch of points by iteratively updating the posterior model and optimising the UCB score computed via the out-dated posterior mean but up-to-date posterior variance. Another UCB-based method, GP-UCB and Pure Exploratio (GP-UCB-PE) (Contal et al., 2013b), decides the first query point in the batch via the UCB criteria and chooses the remaining batch points from the relevant subregions [3] in the search space to maximise the information gain about the unknown objective function f. GP-UCB-PE also enjoys theoretical guarantee on the regret bounds (Contal et al., 2013b). Apart from exploiting the nice property of GP posterior variance, González et al. (2016a) propose a local penalisation (LP) function $\phi(\mathbf{x}; \mathbf{x}_j)$

[3] In cases of maximising the objective function, the relevant subregion is defined to be regions where the upper confidence bound on the function values is higher than the maximum of the lower confidence bound, thus promising higher probability of containing the global optimum $\mathbf{x}^*$.

to efficiently reduce the acquisition function value in the neighbourhood around the selected batch point $\mathbf{x}_j$. The batch points are then recommended via an iterative maximisation-penalisation loop around the acquisition function. The penalisation function is designed such that it penalises a smaller region if the existing point in the batch $\mathbf{x}_j$ is close to the global optimiser $\mathbf{x}^*$ and/or if the variation in the objective function value is large (i.e. a large Lipschitz constant). However, since the objective function and its optimum are unknown, both $\mathbf{x}^*$ and the Lipschitz constant need to be approximated based on the surrogate model. This procedure of penalising the acquisition function essentially simulates the effect of updating GP posterior with $\mathbf{x}_j$ on the acquisition function but avoids the need and thus the computational costs of performing the update. Thus, the LP method is computationally more scalable to large batch size. Also the LP method, though being empirically tested on GP-UCB and EI only, can be applied to any type of acquisition functions.

Synchronous vs Asynchronous Settings

Most of the batch BO approaches focus on the synchronous setting: we evaluate all b points in the batch $\mathcal{B}_t = \{\mathbf{x}_j\}_{j=1}^{b}$ simultaneously on b parallel workers and wait for *all the workers to complete their evaluations* before updating the surrogate model and the acquisition function, as well as sending a new batch of points $\mathcal{B}_{t+1}$ to the workers. In applications where the evaluation times of different input points are roughly equal, this synchronous setting is usually sufficient. However, in AutoML tasks, the evaluation times in a batch often vary largely. Take the example of neural architecture search. A larger neural architecture with more parameters and/or flops will take significantly longer to train and validate compared to a small neural architecture with fewer parameters and/or flops. Under the synchronous setting, all the GPU workers need to wait for the evaluation of the largest neural network to finish before receiving the next round of evaluation jobs. Thus, workers that complete their jobs ahead of others will be left idle at each BO iteration. This leads to inefficient utilisation of parallel computing resources and motivates the adoption of asynchronous setting whereby a new evaluation job is assigned to a

worker as soon as it completes their current evaluation. Compared to the rich synchronous batch BO literature, the field of asynchronous BO (Ginsbourger et al., 2011; Wang et al., 2016a; Kandasamy et al., 2018a; Alvi et al., 2019; Li et al., 2020a) receives much less attention and most approaches are direct extensions of their synchronous counterparts. Although asynchronous batch BO has a clear runtime advantage over synchronous methods, it is often believed to lose out in terms of query efficiency (Kandasamy et al., 2018a); given the same number of function evaluations, synchronous methods will outperform the asynchronous method. However, we will provide strong empirical evidence showing the opposite in Chapter 4, where we will discuss synchronous and asynchronous batch BO in more detail and propose a novel asynchronous batch BO which can be potentially useful for AutoML tasks.

2.3.2 High-dimensional BO

Scaling BO to tackling high-dimensional problems is a very important yet difficult task (Wang et al.; Kandasamy et al., 2015). There are two main challenges in using BO for high-dimensional problems. The first is the curse of dimensionality in modelling the objective function. When the unknown function is high-dimensional, estimating it with non-parametric regression becomes very difficult because it is impossible to densely fill the input space with finite number of sample points, even if the sample size is very large (Györfi et al., 2006). The second challenge is the computational difficulty in optimising the acquisition function. For commonly used global optimisation methods such as grid search, multi-start methods and DIRECT algorithm, the computation cost needed for optimising the acquisition function to within a desired accuracy grows exponentially with dimension (Kandasamy et al., 2015). Thus, the default assumption that the acquisition function can be optimised accurately at negligible cost no longer holds in high dimensions.

An effective way to extend the scalability of BO to high dimensions is to adopt an additive structure (Duvenaud et al., 2011; Kandasamy et al., 2015; Rolland et al., 2018; Wang et al., 2017; Gardner et al., 2017; Wang et al., 2018; Mutny and Krause, 2018). Namely, we make the key assumption that the objective function

can be decomposed into a sum of low-dimensional composite functions defined over overlapping or disjoint subspaces $f(\mathbf{x}) = \sum_{j=1}^{M} f^{(j)}(\mathbf{x}^{(A_j)})$ where $\mathbf{x} = \cup_j^M \mathbf{x}^{(A_j)}$ (Duvenaud et al., 2011). If we impose a GP prior for each $f^{(j)}$, the prior for the overall objective f is an additive GP. We then construct and optimise an acquisition function in each low-dimensional subspace. The optimum input at all the subspaces $\{\mathbf{x}_*^{(A_j)}\}_{j=1}^{M}$ are then combined to give the next query point (Kandasamy et al., 2015). As a result, the two challenges mentioned above are resolved. However, the exact decomposition is often unknown and accurately inferring the decomposition (e.g. via sophisticated sampling (Gardner et al., 2017; Wang et al., 2018)) can be expensive.

Another group of high-dimensional BO methods (Binois et al., 2015; Wang et al., 2016b; Binois et al., 2020) assume the objective function is mainly influenced by a small subset of *effective dimensions* and aims to learn such low-dimensional latent space (Wang et al., 2016b; Nayebi et al., 2019; Letham et al., 2020). The surrogate modelling and acquisition function optimisation are conducted on the low-dimensional space, thus circumventing the difficulties in the original high-dimensional space. The low-dimensional points found by BO are then transformed back to the original high dimension via different projections such as random projection in REMBO (Wang et al., 2016b) or hash functions in HesBO (Nayebi et al., 2019).

Other alternatives include BOCK (Oh et al., 2018) which introduces a cylindrical kernel to mitigate excessive exploration near the boundary region in high dimensional space, LineBO (Kirschner et al., 2019) which restrict the search at each iteration to a one-dimensional subspace to achieve efficient optimisation of the acquisition function, and TuRBO (Eriksson et al., 2019), which constructs local trust regions centred around the best previously observed locations and resizes these trust region to focus the search on promising local zones.

In Chapter 6, we adopt both the assumptions of the additive structure as well as the low-dimensional effective embedding to efficiently handle the high dimensionality of the image input space for performing adversarial attacks. In Chapter 5, we make use of the local trust region idea to achieve efficient optimisation of high-dimensional problem involving both categorical and continuous inputs.

2.3.3 BO on Categorical or Mixed Inputs

The basic approach to deal with categorical variable is to transform them to continuous variables via one-hot encoding (Rasmussen and Williams, 2006a; authors, 2016; Snoek et al., 2012a). While simple in implementation, one-hot encoding suffers several obvious drawbacks: first, it is not scalable to problems with large number of categorical variables and/or categorical variables with large number of choices. For d_h categorical variables, each facing n_i choices, the one-hot-transformed continuous input has a high dimension of $\sum_{i=1}^{d_h} n_i$ in total, further exacerbating the curse of dimensionality. Second, a categorical space differ fundamentally with the continuous counterparts in terms of differentiability and continuity; the function value over categorical input space are only defined in finite locations. This lead to difficulties in using gradient-based methods to optimise the acquisition function of the transformed problems. Therefore, approaches that directly handle the categorical inputs without the need for one-hot encoding is more desirable.

BOCS (Baptista and Poloczek, 2018) is a BO method for the binary categorical inputs: it uses a sparse monomial representation up to the second order and adopts Bayesian linear regression as the surrogate. Inevitably, its expressiveness is constrained by the quadratic model, while scaling it beyond the second order and/or to high dimensionality is usually intractable due to the exponentially-increasing number of parameters that need be learnt explicitly. Also BOCS is specific to binary variables only. COMBO (Oh et al., 2019) is another method that uses a GP surrogate (which is capable of learning interactions of an arbitrary order) with a tailored kernel to deal with categorical inputs directly. It represent all possible joint assignments of the variables (i.e. all categorical combinations) as a combinatorial graph and uses a diffusion graph kernel to model the interactions as well as guide the search. However, COMBO suffers from poor scalability; to avoid overfitting, COMBO approximately marginalises the GP hyper-parameters via Monte Carlo sampling, and it needs to pre-compute the combinatorial graph beforehand. To overcome the challenge of optimising the acquisition function over categorical input space, several recent works proposes the use of the genetic algorithm to evolve

to good categorical structures (Kandasamy et al., 2018b; White et al., 2021a), a learned search control knowledge to uncover promising discrete candidates (Deshwal et al., 2020) or training a policy to generate good candidates and updating the policy in an online fashion (Swersky et al., 2020).

Besides categorical inputs, many practical problems also involve multiple continuous inputs at the same time. For example, when tuning the hyper-parameters for a neural network, we will face categorical hyper-parameters such as the activation type and the optimiser type, as well as continuous variables such as the learning rate and the dropout value. However, BO in mixed categorical-continuous search spaces is rather under-explored. Earlier attempts such as SMAC with random forest surrogates (Hutter et al., 2011a) as well as TPE with non-parametric densities over inputs (Bergstra et al., 2011b) are compatible with such mixed input types. A recent approach, MVRSM, proposed in Bliek et al. (2020) uses a linear combination of rectified linear units (ReLU) to trade off modelling expressiveness for query efficiency in dealing with high-dimensional mixed-variable problems. In terms of GP-based BO, Gopakumar et al. (2018) and Nguyen et al. (2020) consider the simplified setting of a single categorical input with multiple continuous inputs. The continuous inputs are constrained to be *specific* to each categorical choice: namely, they divide the observed data into smaller subsets, one for each categorical and fit a separate independent GP surrogate for each category. In Chapter 5, we propose new BO solutions which fit a shared GP surrogate across categorical inputs to enhance data efficiency, and thus can generalise to the more practical setting involving a mixture of multiple categorical variables and multiple continuous variables.

3

Fast Information-theoretic Bayesian Optimisation

The content of this chapter is based on the following paper:

> Binxin Ru, Mark McLeod, Diego Granziol, and Michael A. Osborne. Fast information-theoretic Bayesian optimisation. In *International Conference on Machine Learning (ICML 2018)*, 2018.

where our contributions were listed in the acknowledgements section of this thesis.

As discussed in Section 2.2, acquisition function controls the trade-off between exploration and exploitation in Bayesian optimisation (BO) and thus is crucial to its search efficiency. BO techniques with information-theoretic acquisition functions have demonstrated state-of-the-art performance in tackling the noisy global optimisation problems, which are commonly seen in AutoML tasks due to the stochasticity in performance evaluation. However, current information-theoretic approaches require many approximations in implementation, introduce often-prohibitive computational overhead and limit the kernel choices available to model the objective. We develop a fast information-theoretic BO method, FITBO, that avoids the need for sampling the global minimiser, thus significantly reducing computational overhead. Moreover, in comparison with existing approaches, our

method faces fewer constraints on kernel choice and enjoys the merits of dealing with the output space. We demonstrate empirically that FITBO inherits the performance associated with information-theoretic BO, while being even faster than simpler BO approaches, such as Expected Improvement.

3.1 Introduction

A core step in BO is to define an acquisition function which uses the available observations effectively to recommend the next query location (Shahriari et al., 2016b). There are many types of acquisition functions such as Probability of Improvement (PI) (Kushner, 1964), Expected Improvement (EI) (Močkus et al., 1978; Jones et al., 1998) and Gaussian Process Upper Confidence Bound (GP-UCB) (Srinivas et al., 2009). The most recent type is based on information theory and offers a new perspective to efficiently select the sequence of sampling locations based on entropy of the distribution over the unknown minimiser $\mathbf{x}_*$ (Shahriari et al., 2016b). The information-theoretic approaches guide our evaluations to locations where we can maximise our learning about the unknown minimum rather than to locations where we expect to obtain lower function values (Hennig and Schuler, 2012b). Such methods have demonstrated impressive empirical performance and tend to outperform traditional methods in tasks with highly multimodal and noisy objective functions, which are particularly common in AutoML.

One popular information-based acquisition function is Predictive Entropy Search (PES) (Villemonteix et al., 2009; Hennig and Schuler, 2012b; Hernández-Lobato et al., 2014) . However, it is very slow to evaluate in comparison with traditional methods like EI, PI and GP-UCB and faces serious constraints in its application. For example, the implementation of PES requires the first and second partial derivatives as well as the spectral density of the Gaussian process kernel function (Hernández-Lobato et al., 2014; Requeima, 2016). This limits our kernel choices. Moreover, PES deals with the input space, thus less efficient in higher dimensional problems (Wang and Jegelka, 2017a). The more recent methods such as Output-space Predictive Entropy Search (OPES) (Hoffman and Ghahramani, 2015) and

Max-value Entropy Search (MES) (Wang and Jegelka, 2017a) improve on PES by focusing on the information content in output space instead of input space. However, current entropy search methods, whether dealing with the minimiser or the minimum value, all involve two separate sampling processes : 1) sampling hyper-parameters for marginalisation and 2) sampling the global minimum/minimiser for entropy computation. The second sampling process not only contributes significantly to the computational burden of these information-based acquisition functions but also requires the construction of a good approximation for the objective function based on Bochner's theorem (Hernández-Lobato et al., 2014), which limits the kernel choices to the stationary ones (Bochner, 1959).

In view of the limitations of the existing methods, we propose a fast information-theoretic BO technique (FITBO). Inspired by the Bayesian integration work in Gunter et al. (2014), the creative contribution of our technique is to approximate any black-box function in a parabolic form: $f(\mathbf{x}) = \eta + 1/2 g(\mathbf{x})^2$. The global minimum is explicitly represented by a hyper-parameter η, which can be sampled together with other hyper-parameters. As a result, our approach has the following three major advantages:

1. Our approach reduces the expensive process of sampling the global minimum/minimiser to the much more efficient process of sampling one additional hyper-parameter, thus overcoming the speed bottleneck of information-theoretic approaches.

2. Our approach faces fewer constraints on the choice of appropriate kernel functions for the Gaussian process prior.

3. Similar to MES (Wang and Jegelka, 2017a), our approach works on information in the output space and thus is more efficient in high dimensional problems.

3.2 Fast Information-theoretic Bayesian Optimisation

Information-theoretic techniques aim to reduce the uncertainty about the unknown global minimiser $\mathbf{x}_*$ by selecting a query point that leads to the largest reduction in entropy of the distribution $p(\mathbf{x}_*|\mathcal{D}_n)$ (Hennig and Schuler, 2012b). The acquisition function for such techniques has the form (Hennig and Schuler, 2012b; Hernández-Lobato et al., 2014):

$$\alpha_{\text{ES}}(\mathbf{x}|\mathcal{D}_n) = H[p(\mathbf{x}^*|\mathcal{D}_n)] - \mathbb{E}_{p(y|\mathcal{D}_n,\mathbf{x})}\Big[H\Big[p\Big(\mathbf{x}^*|\mathcal{D}_n \cup (\mathbf{x},y)\Big)\Big]\Big]. \tag{3.1}$$

PES makes use of the symmetry of mutual information and arrives at the following equivalent acquisition function:

$$\alpha_{\text{PES}}(\mathbf{x}|\mathcal{D}_n) = H[p(y|\mathcal{D}_n,\mathbf{x})] - \mathbb{E}_{p(\mathbf{x}^*|\mathcal{D}_n)}\Big[H\Big[p(y|\mathcal{D}_n,\mathbf{x},\mathbf{x}^*)\Big]\Big], \tag{3.2}$$

where $p(y|\mathcal{D}_n,\mathbf{x},\mathbf{x}^*)$ is the predictive posterior distribution for y conditioned on the observed data $\mathcal{D}_n$, the test location $\mathbf{x}$ and the global minimiser $\mathbf{x}^*$ of the objective function. For more detailed description on information-theoretic acquisition functions, please refer to Section 2.2.3 in Chapter 2.3.1.

FITBO harnesses the same information-theoretic thinking but measures the entropy about the latent global minimum $f^* = f(\mathbf{x}^*)$ instead of that of the global minimiser $\mathbf{x}^*$. Thus, the acquisition function of FITBO method is the mutual information between the function minimum f^* and the next query point (Wang and Jegelka, 2017a). In other words, FITBO aims to select the next query point which minimises the entropy of the global minimum:

$$\alpha_{\text{FITBO}}(\mathbf{x}|\mathcal{D}_n) = H[p(y|\mathcal{D}_n,\mathbf{x})] - \mathbb{E}_{p(f^*|\mathcal{D}_n)}\Big[H\Big[p(y|\mathcal{D}_n,\mathbf{x},f^*)\Big]\Big]. \tag{3.3}$$

This idea of changing entropy computation from the input space to the output space is also shared by Hoffman and Ghahramani (2015) and Wang and Jegelka (2017a). Hence, the acquisition function of the FITBO method is very similar to those of OPES (Hoffman and Ghahramani, 2015) and MES (Wang and Jegelka, 2017a).

However, our novel contribution is to express the unknown objective function in a parabolic form $f(\mathbf{x}) = \eta + {}^1\!/\!_2 g(\mathbf{x})^2$, thus representing the global minimum $\mathbf{x}^*$ by a hyper-parameter η and circumventing the laborious process of sampling the global minimum. FITBO acquisition function can then be reformulated as:

$$\begin{aligned}\alpha_{\text{FITBO}}(\mathbf{x}|\mathcal{D}_n) &= H[p(y|\mathcal{D}_n, \mathbf{x})] - \mathbb{E}_{p(\eta|\mathcal{D}_n)}\Big[H\Big[p(y|\mathcal{D}_n, \mathbf{x}, \eta)\Big]\Big] \\ &= H\Big[\int p(y|\mathcal{D}_n, \mathbf{x}, \eta) p(\eta|\mathcal{D}_n) \mathrm{d}\eta\Big] - \int p(\eta|\mathcal{D}_n) H\Big[p(y|\mathcal{D}_n, \mathbf{x}, \eta)\Big] \mathrm{d}\eta.\end{aligned}$$

The intractable integral terms can be approximated by drawing M samples of η from the posterior distribution $p(\eta|\mathcal{D}_n)$ and using a Monte Carlo method (Hernández-Lobato et al., 2014). The predictive posterior distribution $p(y|\mathcal{D}_n, \mathbf{x}, \eta)$ can be turned into a neat Gaussian form by applying a local linearisation technique on our parabolic transformation as described in Section 3.2.1. Thus, the first term in the above FITBO acquisition function is an entropy of a Gaussian mixture, which is intractable and demands approximation as described in Section 3.2.3. The second term is the expected entropy of a one-dimensional Gaussian distribution and can be computed analytically because the entropy of a Gaussian has the closed form: $H[p(y|\mathcal{D}_n, \mathbf{x}, \eta)] = 0.5 \log\Big[2\pi e\Big(\sigma_f(\mathbf{x}|\mathcal{D}_n, \eta)^2 + \sigma^2_{\text{noise}}\Big)\Big]$ where $\sigma_f(\mathbf{x}|\mathcal{D}_n, \eta)^2 = K_f(\mathbf{x}, \mathbf{x}')$ and σ^2_{noise} is the variance of observation noise.

3.2.1 Parabolic Transformation and Predictive Posterior Distribution

Gunter et al. (2014) use a square-root transformation on the integrand in their warped sequential active Bayesian integration method to ensure non-negativity. Inspired by this work, we creatively express any unknown objective function $f(\mathbf{x})$ in the parabolic form:

$$f(\mathbf{x}) = \eta + {}^1\!/\!_2 g(\mathbf{x})^2, \tag{3.4}$$

where η is the global minimum of the objective function. Given the noise-free observation data $\mathcal{D}_f = \{(\mathbf{x_i}, f_i)\}_{i=1}^n = \{\mathbf{X}_n, \mathbf{f}_n\}$, the observation data on g is $D_g = \{(\mathbf{x_i}, g_i)|i = 1, \ldots n\} = \{\mathbf{X}_n, \mathbf{g}_n\}$ where $g_i = \sqrt{2(f_i - \eta)}$.

We impose a zero-mean Gaussian process prior on $g(\mathbf{x})$, $g \sim \mathcal{GP}\Big(0, k(\mathbf{x}, \mathbf{x}')\Big)$, so that the posterior distribution for g conditioned on the observation data D_g and the test point $\mathbf{x}$ also follows a Gaussian process:

$$p(g|D_g, \mathbf{x}, \eta) = \mathcal{GP}\Big(g; \mu_g(\cdot), K_g(\cdot, \cdot)\Big) \tag{3.5}$$

where

$$\mu_g(\mathbf{x}) = K(\mathbf{x}, \mathbf{X}_n)K(\mathbf{X}_n, \mathbf{X}_n)^{-1}\mathbf{g}_n,$$

$$K_g(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{X}_n)K(\mathbf{X}_n, \mathbf{X}_n)^{-1}K(\mathbf{X}_n, \mathbf{x}').$$

The parabolic transformation causes the distribution for any f to become a non-central χ^2 process, making the analysis intractable. In order to tackle this problem and obtain a posterior distribution $p(f|D_f, \mathbf{x}, \eta)$ that is also Gaussian, we employ a linearisation technique (Gunter et al., 2014).

We perform a local linearisation of the parabolic transformation $h(g) = \eta + 1/2 g^2$ around g_0 and obtain $f \approx h(g_0) + h'(g_0)(g - g_0)$ where the gradient $h'(g) = g$. By setting g_0 to the mode of the posterior distribution $p(g|D_g, \mathbf{x}, \eta)$ (i.e. $g_0 = \mu_g$), we obtain an expression for f which is linear in g:

$$\begin{aligned} f(\mathbf{x}) &\approx [\eta + 1/2 \mu_g(\mathbf{x})^2] + \mu_g(\mathbf{x})[g(\mathbf{x}) - \mu_g(\mathbf{x})] \\ &= \eta - 1/2 \mu_g(\mathbf{x})^2 + \mu_g(\mathbf{x})g(\mathbf{x}). \end{aligned} \tag{3.6}$$

Since the affine transformation of a Gaussian process remaiGns Gaussian, the predictive posterior distribution for f now has a closed form:

$$p(f|D_f, \mathbf{x}, \eta) = \mathcal{GP}\Big(f; \mu_f(\cdot), K_f(\cdot, \cdot)\Big) \tag{3.7}$$

where

$$\mu_f(\mathbf{x}) = \eta + 1/2 \mu_g(\mathbf{x})^2$$

$$K_f(\mathbf{x}, \mathbf{x}') = \mu_g(\mathbf{x})K_g(\mathbf{x}, \mathbf{x}')\mu_g(\mathbf{x}').$$

However, in real world situations, we do not have access to the true function values but only noisy observations of the function, $y(\mathbf{x}) = f(\mathbf{x}) + \epsilon$, where ϵ is assumed to be an independently and identically distributed Gaussian noise with

variance σ^2_{noise} (Rasmussen and Williams, 2006b). Given noisy observation data $\mathcal{D}_n = \{(\mathbf{x_i}, y_i)\}_{i=1}^n = \{\mathbf{X}_n, \mathbf{y}_n\}$, the predictive posterior distribution (3.7) becomes:

$$p(y|\mathcal{D}_n, \mathbf{x}, \eta) = \mathcal{GP}\Big(y; \mu_f(\cdot), K_f(\cdot,\cdot) + \sigma^2_{\text{noise}}\delta(\cdot,\cdot)\Big). \tag{3.8}$$

3.2.2 Hyper-parameter Treatment

Hyper-parameters are the free parameters, such as output scale and characteristic length scales in the kernel function for the Gaussian processes as well as noise variance. We use $\boldsymbol{\theta}$ to represent a vector of hyper-parameters that includes all the kernel parameters and the noise variance. Recall that we introduce a new hyper-parameter η in our model to represent the global minimum. To ensure that η is not greater than the minimum observation y_{min}, we assume that $\log(y_{min} - \eta)$ follows a broad normal distribution. Thus the prior for η has the form:

$$p(\eta) = \frac{1}{(y_{min} - \eta)}\mathcal{N}\Big(\log(y_{min} - \eta); \mu, \sigma^2\Big). \tag{3.9}$$

The most popular approach to hyper-parameter treatment is to learn hyper-parameter values via maximum likelihood estimation (MLE) or maximum a posterior estimation (MAP). However, both MLE and MAP are not desirable as they give point estimates and ignore our uncertainty about the hyper-parameters (Hernández-Lobato et al., 2014). In a fully Bayesian treatment of the hyper-parameters, we should consider all possible hyper-parameter values. This can be done by marginalising the terms in the acquisition function with respect to the posterior $p(\boldsymbol{\psi}|\mathcal{D}_n)$ where $\boldsymbol{\psi} = \{\boldsymbol{\theta}, \eta\}$:

$$\alpha_{\text{FITBO}}(\mathbf{x}|\mathcal{D}_n) = H\Big[\int p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\psi})p(\boldsymbol{\psi}|\mathcal{D}_n)\mathrm{d}\boldsymbol{\psi}\Big] - \int p(\boldsymbol{\psi}|\mathcal{D}_n)H\Big[p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\psi})\Big]\mathrm{d}\boldsymbol{\psi}.$$

Since complete marginalisation over hyper-parameters is analytically intractable, the integral can be approximated using the Monte Carlo method (Hoffman and Ghahramani, 2015; Snoek et al., 2012b), leading to the final expression:

$$\begin{aligned}\alpha_{\text{FITBO}}(\mathbf{x}|\mathcal{D}_n) = H\Big[\frac{1}{M}\sum_j^M p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)})\Big] \\ - \frac{1}{2M}\sum_j^M \log\Big[2\pi e\Big(\sigma_f(\mathbf{x}|D, \boldsymbol{\theta}^{(j)}, \eta^{(j)})^2 + \sigma^2_{\text{noise}}\Big)\Big].\end{aligned} \tag{3.10}$$

3.2.3 Approximation for the Gaussian Mixture Entropy

The entropy of a Gaussian mixture is intractable and can be estimated via the following three methods:

1. **Taylor expansion** Huber et al. (2008) propose a novel method for approximating the entropy of a Gaussian mixture model by using a Taylor-series expansion of the logarithm of the Gaussian mixture. Let $q(y) = \sum_j^M w_j \mathcal{N}(y; \mu_j, \sigma_j^2)$. The Gaussians in the mixture are univariate in our case because the function value at a test location $\mathbf{x}$ is 1-D. The entropy of this mixture is:

$$H\left[q(y)\right] = -\int q(y) \log h(y) \mathrm{d}y \tag{3.11}$$

where $h(y) = q(y)$ but we uses different notations to differentiate the Gaussian mixture that's argument of the logarithm from that in front of the logarithm.

By expanding the logarithm term around the mean of each Gaussian term μ_j in $h(y)$, the resultant R-th order Taylor series is

$$\log h(y) = \sum_{k=0}^{R} \frac{(y-\mu_j)^k}{k!} \frac{\mathrm{d}^k\Big(\log h(y)\Big)}{\mathrm{d}y^k}\bigg|_{y=\mu_j}. \tag{3.12}$$

We then substitute Equation (3.12) into Equation (3.11) and obtain

$$\begin{aligned} H\left[q(y)\right] &= -\int q(y) \log h(y) dy \\ &= -\int \sum_{j}^{M} w_j \mathcal{N}(y; \mu_j, \sigma_j^2) \log h(y) \mathrm{d}y \\ &= -\sum_{j}^{M} w_j \sum_{k=0}^{R} \frac{1}{k!} \frac{\mathrm{d}^k\Big(\log h(y)\Big)}{\mathrm{d}y^k}\bigg|_{y=\mu_j} \int \mathcal{N}(y; \mu_j, \sigma_j^2)(y-\mu_j)^k \mathrm{d}y \end{aligned}$$

where $\int \mathcal{N}(y; \mu_j, \sigma_j^2)(y-\mu_j)^k \mathrm{d}y$ is the k-th central moment of a Gaussian distribution and thus has a closed form (Requeima, 2016). The k-th derivative of the logarithm of Gaussian mixture $h(y)$ can also be computed analytically because the derivatives of a Gaussian distribution always exist and Kronecker algebra can be used to achieve a compact representation (Huber et al., 2008).

The entropy approximation by Taylor expansion faces the trade-off between the accuracy and computational burden as we can obtain more accurate

approximations by including higher order Taylor-series terms at the expense of computational speed (Huber et al., 2008). Experiments with this approximation approach are carried out with a second-order Taylor-series expansion whose explicit form is provided by the Appendix in Huber et al. (2008):

$$\begin{aligned} H\left[\frac{1}{N}\sum_{j=1}^{M} p(y|D_n, \mathbf{x}, \eta^{(j)})\right] &\approx H_0[y] + H_2[y] \\ &= -\sum_{j=1}^{M} w_j \log h(\mu_j) - \sum_{j=1}^{M} \frac{w_j \sigma_j^2}{2} F(\mu_j) \end{aligned}$$

where

$$\begin{aligned} F(y) =& h(y)^{-1} \sum_{i=1}^{N} w_i \sigma_i^{-2} [h(y)^{-1}(y-\mu_i)h'(y) \\ &+ \sigma_i^{-2}(y-\mu_i)^2 - 1]\mathcal{N}(y; \mu_i, \sigma_i^2). \end{aligned}$$

2. **Numerical integration** As mentioned before, one advantage of FITBO method is that it allows us to transform the entropy calculation from the multi-dimensional input space to the one-dimensional output space. This, thus, permits the use of numerical integration techniques to effectively compute the entropy of a Gaussian mixture. Experiments with numerical integration are performed with the `quad` function in MATLAB which utilises the adaptive Simpson quadrature.

3. **Monte Carlo integration** Let $p(y|I^{(j)})$ denotes $p(y|D_n, \mathbf{x}, \eta^{(j)})$. The first term in our FITBO acquisition function can be reformulated in the following way:

$$\begin{aligned} H\left[\sum_{j}^{M} w_j p(y|I^{(j)})\right] &= -\int \left(\sum_{j}^{M} w_j p(y|I^{(j)})\right) \log \left(\sum_{j}^{M} w_j p(y|I^{(j)})\right) \mathrm{d}y \\ &= -\sum_{j}^{M} w_j \int p(y|I^{(j)}) \log \left(\sum_{j}^{M} w_j p(y|I^{(j)})\right) \mathrm{d}y \end{aligned}$$

where $w_j = \frac{1}{N}$ in our case. By drawing N samples of y from $p(y|I^{(j)})$ and using Monte Carlo integration, the entropy of a Gaussian mixture can be

approximated as

$$H\left[\sum_{j}^{M} w_j p(y|I^{(j)})\right] \approx -\sum_{j}^{M} w_j \left[\frac{1}{N}\sum_{i}^{N} \log\left(\sum_{j}^{M} w_j p(y^{(i)}|I^{(j)})\right)\right] \tag{3.13}$$

The accuracy of the simple Monte Carlo approximation can be enhanced by increasing the sample size M. But larger number of samples will increase the computational burden. Thus, we also face a trade-off between the approximation precision and computational speed.

Of these three, our experimentation in Appendix A.1 revealed that numerical integration (in particular, an adaptive Simpson's method) was clearly the most performant for our application. Note that our Gaussian mixture is univariate. In another work (Granziol et al., 2019) for which I am the joint first author, we also propose the use of a maximum entropy algorithm to efficiently approximate the Gaussian mixture and the results are shown in Appendix A.2.

A faster alternative is to approximate the first entropy term by matching the first two moments of a Gaussian mixture. The mean and variance of a univariate Gaussian mixture model $p(z) = \sum_j^M \frac{1}{M}\mathcal{N}(z|\mu_j, K_j)$ have the analytical form:

$$\mathbb{E}[z] = \sum_{j}^{M} \frac{1}{M}\mu_j \tag{3.14}$$

$$\sigma(z)^2 = \sum_{j}^{M} \frac{1}{M}(K_j + \mu_j^2) - \mathbb{E}[z]^2. \tag{3.15}$$

By fitting a Gaussian to the Gaussian mixture, we can obtain a closed-form upper bound for the first entropy term: $H[p(z)] \approx 0.5 \log\left[2\pi e\left(\sigma(z)^2 + \sigma^2_{\text{noise}}\right)\right]$, thus further enhancing the computational speed of FITBO approach.

However, the moment-matching approach results in a looser approximation than numerical integration shown in Section 3.3.1 and we will compare both approaches in our experiments in Section 3.3.

3.2.4 The Algorithm

The procedures of computing the acquisition function of FITBO are summarised by Algorithm 3. Figure 3.1 illustrates the sampling behaviour of FITBO method for a simple 1D BO problem. The optimisation process is started with 3 initial observation data. As more samples are taken, the mean of the posterior distribution for the objective function gradually resembles the objective function and the distribution of η converges to the global minimum.

Algorithm 3 FITBO acquisition function

Input: a test input $\mathbf{x}$; noisy observation data $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
Sample hyper-parameters and η from $p(\boldsymbol{\psi}|\mathcal{D}_n)$: $\boldsymbol{\Psi} = \{(\boldsymbol{\theta}^{(j)}, \eta^{(j)})\}_{j=1}^M$
for j $= 1, \ldots, M$ **do**
  Use $f(\mathbf{x}) = \eta + 1/2 g(\mathbf{x})^2$ to approximate $p(f|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)}) = \mathcal{GP}\Big(\mu_f(\cdot), K_f(\cdot,\cdot)\Big)$
  Compute $p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)})$
  Compute $H[p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)})]$
end for
Estimate the entropy of the Gaussian mixture : $\mathrm{E}_1(\mathbf{x}|\mathcal{D}_n) = H\Big[\frac{1}{M}\sum_j^M p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)})\Big]$
Compute the entropy expectation: $\mathrm{E}_2(\mathbf{x}|\mathcal{D}_n) = \frac{1}{M}\sum_j^M H[p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)})] = \frac{1}{2M}\sum_j^M \log\Big[2\pi e\Big(\sigma_f(\mathbf{x}|\mathcal{D}_n, \boldsymbol{\theta}^{(j)}, \eta^{(j)})^2 + \sigma^2_{\text{noise}}\Big)\Big]$
return $\alpha_n(\mathbf{x}|\mathcal{D}_n) = \mathrm{E}_1(\mathbf{x}|\mathcal{D}_n) - \mathrm{E}_2(\mathbf{x}|\mathcal{D}_n)$

3.3 Experiments

We conduct a series of experiments to test the empirical performance of FITBO and compare it with other popular acquisition functions. In this section, FITBO denotes the version using numerical integration to estimate the entropy of the Gaussian mixture while FITBO-MM denotes the version using moment matching. In all experiments, we adopt a zero-mean Gaussian process prior with the squared exponential kernel function and use the elliptical slice sampler (Murray et al., 2010) for sampling hyper-parameters $\boldsymbol{\theta}$ and η. For the implementation of EI, PI, GP-UCB, MES and PES, we use the open source Matlab code by Wang and Jegelka (2017a) and Hernández-Lobato et al. (2014). Our Matlab code for

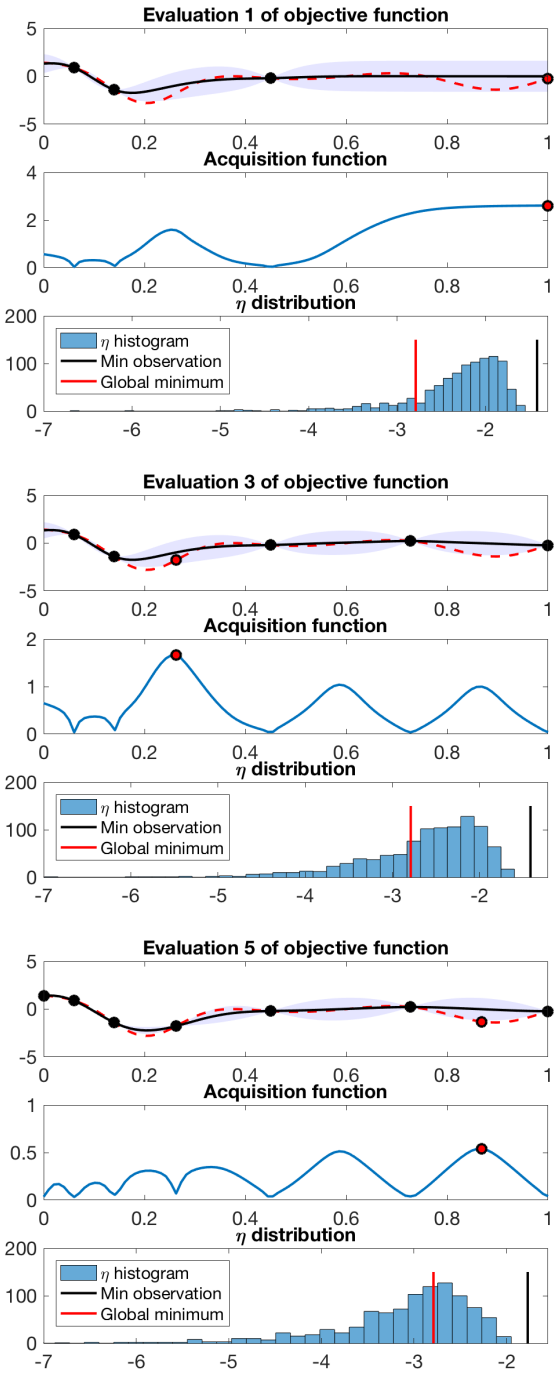


Figure 3.1: BO for a 1D objective function using FITBO method at the 1st, 3rd, 5th evaluations. In each subfigure, the top plot shows the objective function (red dotted line), the posterior mean (black solid line) and the 95% confidence interval (blue shaded area) estimated by the Gaussian process model as well as the observation points (black dot) and the next query point (red dot). The middle plot shows the acquisition function. The bottom plot is the histogram of η samples as well as its relation to the minimum observation (black vertical line) and the true global minimum (red vertical line).

FITBO will be available at https://github.com/rubinxin/FITBO. We use the type of MES method that samples the global minimum $f(\mathbf{x}^*)$ from an approximated posterior function $\tilde{f}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\tilde{\mathbf{a}}$ where $\boldsymbol{\phi}(\mathbf{x})$ is an m-dimensional feature vector and $\tilde{\mathbf{a}}$ is a Gaussian weight vector (Wang and Jegelka, 2017a). This is also the minimiser sampling strategy adopted by PES (Hernández-Lobato et al., 2014). The computational complexity of sampling $\tilde{\mathbf{a}}$ from its posterior distribution $p(\tilde{\mathbf{a}}|\mathcal{D}_n)$ is $\mathcal{O}(n^2m)$ when $n < m$ (Hernández-Lobato et al., 2014). Minimising $\tilde{f}(\mathbf{x})$ to within ζ accuracy using any grid search or branch and bound optimiser requires $\mathcal{O}(\zeta^{-d})$ calls to $\tilde{f}(\mathbf{x})$ for d-dimensional input data (Kandasamy et al., 2015). For both PES and MES, we apply their fastest versions which draw only 1 minimum or minimiser sample to estimate the acquisition function.

3.3.1 The Acquisition Function Obtained by Numerical Integration and Moment-matching

We first compare the resultant acquisition functions obtained by using (1) numerical integration (FITBO) and (2) moment-matching (FITBO-MM) to approximate Gaussian mixture entropy. Figure 3.2 shows the acquisition function of FITBO-MM in comparison with those of FITBO which use different tolerance levels $\{1e-3, 1e-4, 1e-6\}$ for numerical integration. We assume the acquisition function obtained using numerical integration with a tolerance level of $1e-6$ (pink line in Figure 3.2) to be a fair representation of the true value. It is evident that the moment-matching method leads to a looser upper bound than numerical integration but the resultant acquisition function manages to capture the true function shape quite well and recommend a query point that is very close to the best numerical approximation.

3.3.2 Runtime Tests

The first set of experiments measure and compare the runtime of evaluating the acquisition functions $\alpha_n(\mathbf{x}|\mathcal{D}_n)$ for methods including GP-UCB, PI, EI, PES, MES, FITBO and FITBO-MM. All the timing tests were performed exclusively on a 2.3 GHz Intel Core i5. The runtime measured excludes the time taken for sampling

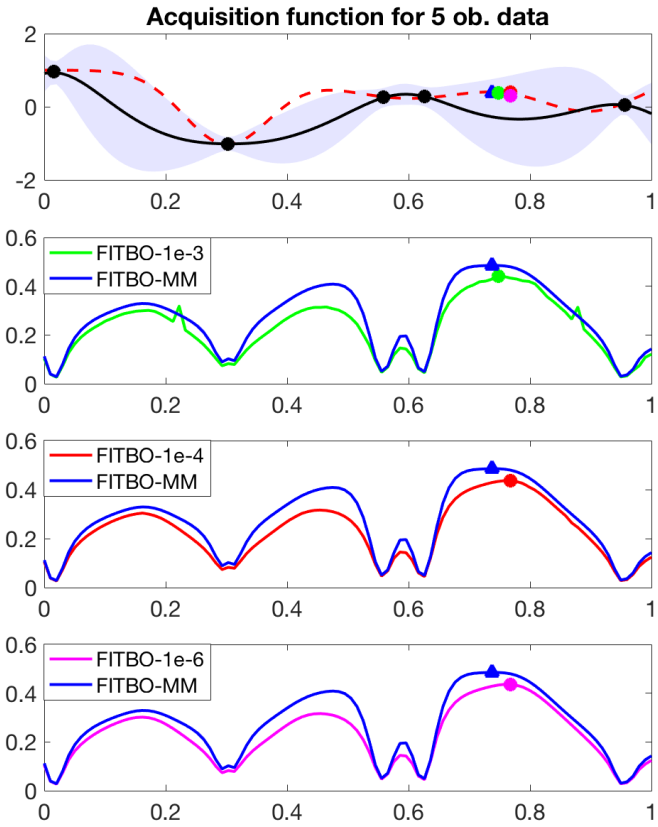


Figure 3.2: Acquisition functions obtained using numerical integration and moment-matching approximation methods. The top plot shows the objective function (red dotted line), the posterior mean (black solid line) and the 95% confidence interval (blue shaded area) estimated by the Gaussian process model as well as the observation points (black dot). The following three plots show the acquisition function value of FITBO-MM and those of FITBO with different tolerance level (1e-3, 1e-4, 1e-6) used for numerical integration.The next query points are recommended by maximising respective acquisition functions: FITBO-1e-3 (green dot), FITBO-1e-4 (red dot), FITBO-1e-6 (pink dot) and FITBO-MM (blue triangle). The acquisition functions are computed using 600 hyperparameter samples.

hyper-parameters as well as optimising the acquisition functions. The methodology of the tests can be summarised as follows:

1. Generate 10 initial observation data from a d-dimensional test function and sample a set of M hyper-parameters $\boldsymbol{\Psi} = \{\boldsymbol{\theta}^{(j)}, \eta^{(j)} | j = 1, \ldots, M\}$ from the log posterior distribution $\log \tilde{p}(\boldsymbol{\psi}|\mathcal{D}_n)$ using the elliptical slice sampler.

2. Use this set of hyper-parameters to evaluate all acquisition functions at 100 test points.

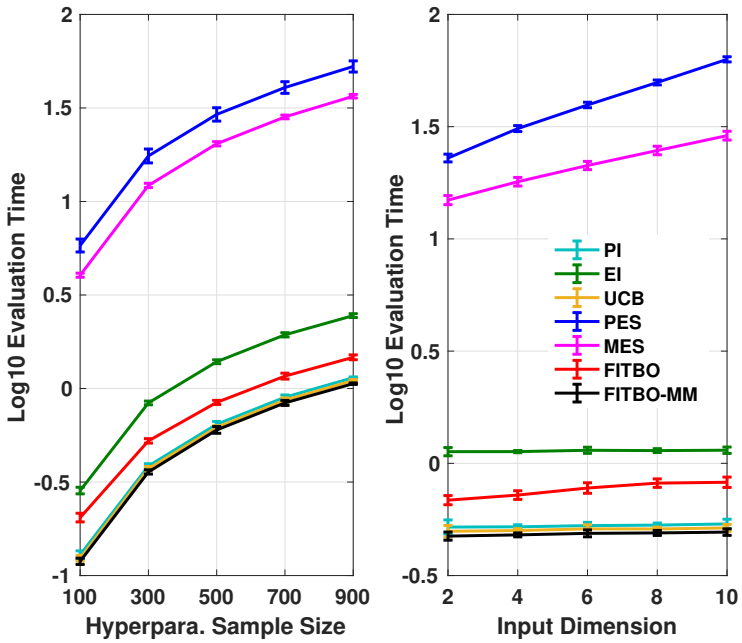


Figure 3.3: The runtime of evaluating 7 different acquisition functions (PI, EI, GP-UCB, PES, MES, FITBO and FITBO-MM) at 100 test inputs. The left plot shows the runtime of evaluating the acquisition functions using M hyper-parameter samples for 2D input data and M tested are 100, 300, 500, 700, 900. The right plot shows the runtime of evaluating the acquisition functions using 400 hyper-parameter samples for input data of dimension d where d are 2, 4, 6, 8, 10. The y-axes are the evaluation runtime expressed in the logarithm to the base 10.

Table 3.1: Runtime of evaluating PI, GP-UCB and FITBO-MM at 100 2D inputs using M hyper-parameter samples (Unit: Second).

M	PI	GP-UCB	FITBO-MM
100	0.1293	0.1238	**0.1193**
	(± 0.006)	(± 0.005)	(± 0.005)
300	0.3856	0.3731	**0.3582**
	(± 0.011)	(± 0.010)	(± 0.009)
500	0.6442	0.6205	**0.6011**
	(± 0.025)	(± 0.012)	(± 0.027)
700	0.8990	0.8670	**0.8382**
	(± 0.026)	(± 0.026)	(± 0.028)
900	1.1426	1.1025	**1.0618**
	(± 0.011)	(± 0.014)	(± 0.010)

Table 3.2: Runtime of evaluating PI, GP-UCB and FITBO-MM for 100 test inputs of dimension d with M = 400 (Unit: Second).

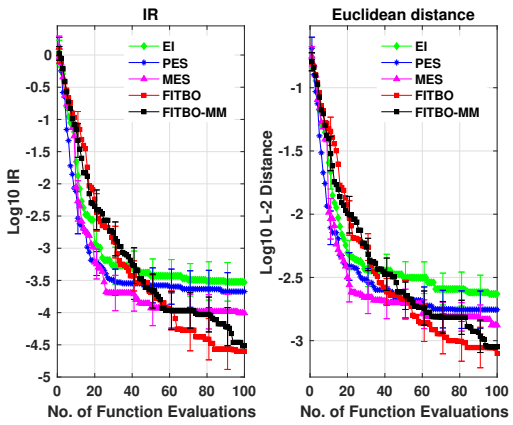
d	PI	GP-UCB	FITBO-MM
2	0.5217	0.4991	**0.4745**
	(± 0.047)	(± 0.034)	(± 0.021)
4	0.5215	0.5020	**0.4800**
	(± 0.011)	(± 0.010)	(± 0.010)
6	0.5281	0.5112	**0.4879**
	(± 0.019)	(± 0.023)	(± 0.019)
8	0.5307	0.5102	**0.4899**
	(±0.011)	(± 0.010)	(± 0.013)
10	0.5378	0.5159	**0.4942**
	(± 0.029)	(± 0.019)	(± 0.017)

3. Repeat the procedures 1 and 2 for 100 different initialisations and compute the mean and standard deviation of the runtime taken for evaluating various acquisition functions.

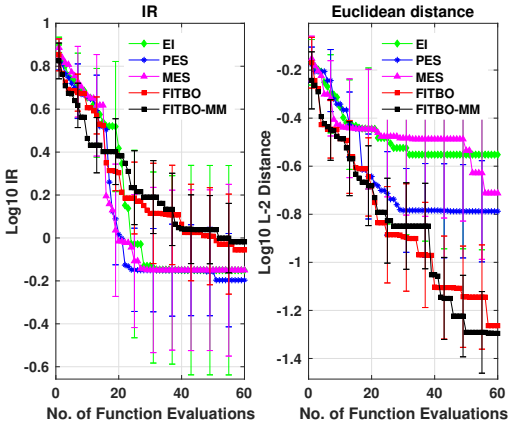
We did not include the time for sampling η alone into the runtime of evaluating FITBO and FITBO-MM because η is sampled jointly with other hyper-parameters and does not add to the overall sampling burden significantly. In fact, we have tested that sampling η by the elliptical slice sampler adds 0.09 seconds on average when drawing 2 000 samples and 0.93 seconds when drawing 20 000 samples. Note further that we will limit all methods to a fixed number of hyper-parameter samples in both runtime tests and performance experiments: this will impart a slight performance penalty to our method, which must sample from a hyper-parameter space of one additional dimension.

The above tests are repeated for different hyper-parameter sample sizes $M = 100, 300, 500, 700, 900$ and input data of different dimensions $d = 2, 4, 6, 8, 10$. The results are presented graphically in Figure 3.3 with the evaluation runtime being expressed in the logarithm to the base 10 and the exact numerical results for methods that are very close in runtime are presented in Tables 3.1 and 3.2.

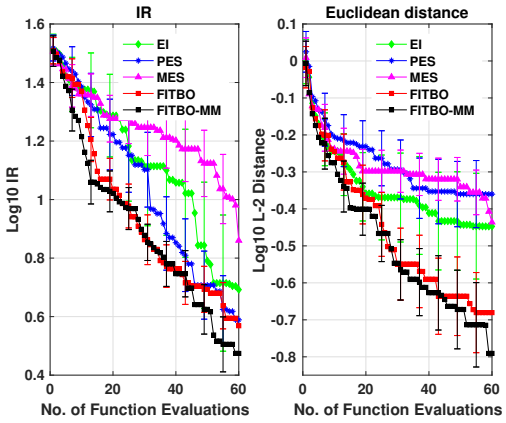
Figure 3.3 shows that FITBO is significantly faster to evaluate than PES and MES for various hyper-parameter sample sizes used and for problems of different input dimensions. Moreover, FITBO even gains a clear speed advantage over EI. The moment matching technique manages to further enhance the speed of FITBO, making FITBO-MM comparable with, if not slightly faster than, simple algorithms like PI and GP-UCB. In addition, we notice that the runtime of evaluating FITBO-MM, EI, PI and GP-UCB tend to remain constant regardless of the input dimensions while the runtime for PES and MES tends to increase with input dimensions at a rate of 10^d. Thus, our approach is more efficient and applicable in dealing with high-dimensional problems.



(a) Branin-2D



(b) Eggholder-2D



(c) Hartmann-6D

Figure 3.4: Optimisation performance of EI, PES, MES, FITBO and FITBO-MM for three benchmark test functions.

3.3.3 Tests with Benchmark Functions

We perform optimisation tasks on three challenging benchmark functions: Branin (defined in $[0,1]^2$), Eggholder (defined in $[0,1]^2$) and Hartmann (defined in $[0,1]^6$). In all tests, we set the observation noise to $\sigma^2_{\text{noise}} = 10^{-3}$ and resample all the hyper-parameters after each function evaluation. In evaluating the optimisation performance of various BO methods, we use the two common metrics adopted by Hennig and Schuler (2012b). The first metric is Immediate regret (IR) which is defined as:

$$IR = |f(\mathbf{x}^*) - f(\hat{\mathbf{x}}_n)| \tag{3.16}$$

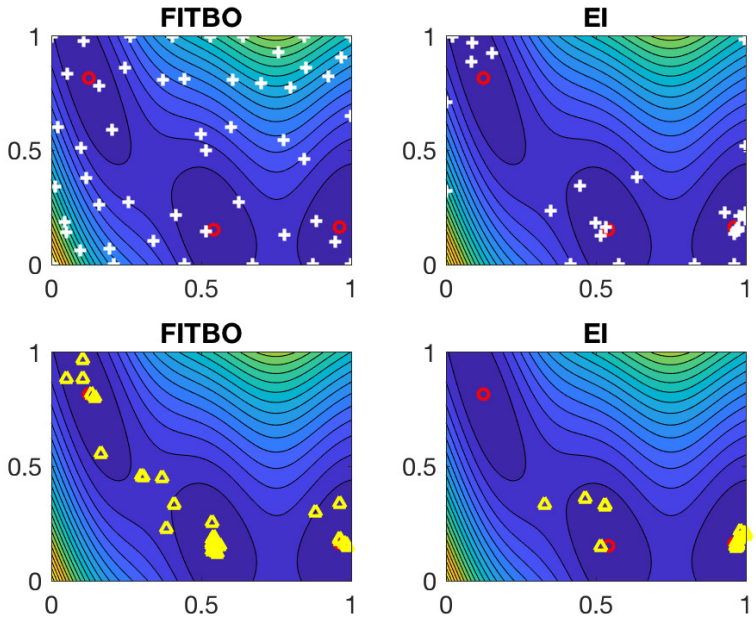


Figure 3.5: Evaluations taken by FITBO and EI in the Branin-2D problem. The white crosses in the top plots indicate the first 50 query points recommended by the two algorithms. The yellow triangles in the bottom plots indicate the guesses of the global minimiser recommended by the algorithms (i.e. $\hat{\mathbf{x}}_n$) after each evaluation. FITBO, which is more explorative in taking evaluations, successfully identifys all three global minimisers (red circle) but EI misses out one of the global minimisers.

where $\mathbf{x}^*$ is the location of true global minimum and $\hat{\mathbf{x}}_n$ is the best guess recommended by a Bayesian optimiser after n iterations, which corresponds to the minimiser of the posterior mean. The second metric is the Euclidean distance of an optimiser's recommendation $\hat{\mathbf{x}}_n$ from the true global minimiser $\mathbf{x}^*$, which is defined as:

$$\|L\|_2 = \|\mathbf{x}^* - \hat{\mathbf{x}}_n\|. \tag{3.17}$$

We compute the median IR and the median $\|L\|_2$ over 40 random initialisations. At each initialisation, all BO algorithms start from 3 random observation data for Branin-2D and Eggholder-2D problems and from 9 random observation data for Hartmann-6D problem. The results are presented in Figure 3.4. The plots on the left show the median IR achieved by each approach as more evaluation steps are taken. The plots on the right show the median $\|L\|_2$ between each optimiser's recommended global minimiser and the true global minimiser. The error bars indicate one standard deviation.

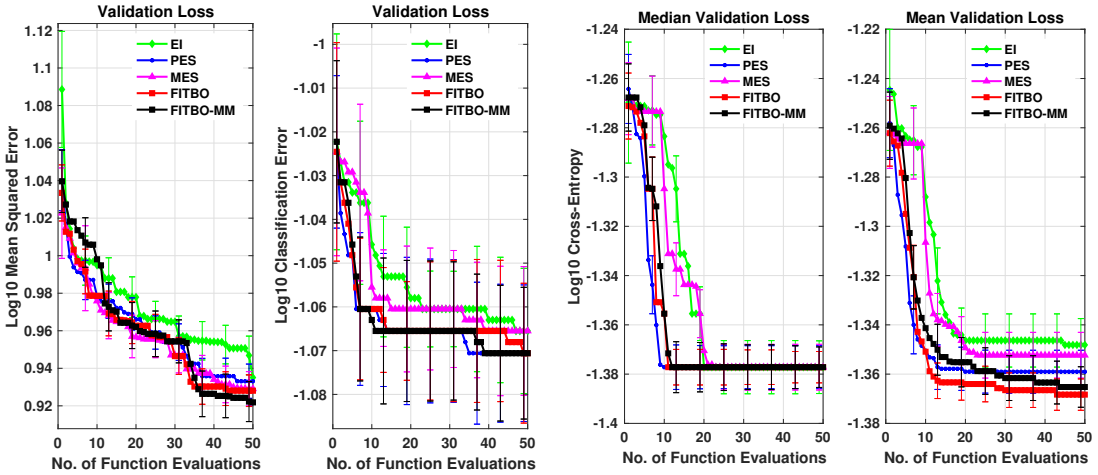
In the case of Branin-2D, FITBO and FITBO-MM lose out to other methods initially but surpass other methods after 50 evaluations. One interesting point we would like to illustrate through the Branin problem is the fundamentally different mechanisms behind information-based approaches like FITBO and improvement-based approaches like EI. As shown in Figure 3.5, FITBO is much more explorative compared to EI in taking new evaluations because FITBO selects the query points that maximise the information gain about the minimiser instead of those that lead to an improvement over the best function value observed. FITBO successfully finds all three global minimisers but EI quickly concentrates its searches into regions of low function values, missing out one of the global minimisers.

In the case of Eggholder-2D which is more complicated and multimodal, FITBO and FITBO-MM perform not as well as other methods in finding lower function values but outperform all competitors in locating the global minimiser by a large margin. One reason is that the function value near the global minimiser of Eggholder-2D rises sharply. Thus, although FITBO and FITBO-MM are able to better identify the location of the true global minimum, they return higher function values than other methods that are trapped in locations of good local minima.

As for a higher dimensional problem, Hartmann-6D, FITBO and FITBO-MM outperform all other methods in finding both the lower function value and the location of the global minimum. In all three tasks, FITBO-MM, despite using a looser upper bound of the Gaussian mixture entropy, still manages to demonstrate similar, sometimes better, results compared with FITBO. This shows that the performance of our information-theoretic approach is robust to slightly worse approximation of the Gaussian mixture entropy.

3.3.4 Test with Real-world Problems

Finally, we experiment with a series of real-world optimisation problems. The first problem (Boston) returns the L2 validation loss of a 1-hidden layer neural network (Wang and Jegelka, 2017a) on the Boston housing dataset (Bache and Lichman, 2013). The dataset is randomly partitioned into train/validation/test sets and the



(a) Boston **(b)** MNIST-SVM **(c)** Cancer4D

Figure 3.6: Performance on tuning hyper-parameters for (a) training a neural network on the Boston housing dataset, (b) training an SVM on the MNIST dataset and (c) training a classification neural network on the breast cancer dataset.

neural network is trained with Levenberg-Marquardt optimisation. The 2 variables tuned with BO are the number of neurons and the damping factor λ.

The second problem (MNIST-SVM) outputs the classification error of a support vector machine (SVM) classifier on the validation set of the MNIST dataset (LeCun et al., 1998). The SVM classifier adopts a radial basis kernel and the 2 variables to optimise are the kernel scale parameter and the box constraint.

The third problem (Cancer) returns the cross-entropy loss of a 1-hidden layer neural network (Wang and Jegelka, 2017a) on the validation set of the breast cancer dataset (Bache and Lichman, 2013). This neural network is trained with the scaled conjugate gradient method and we use BO methods to tune the number of neurons, the damping factor λ, the $\lambda-$increase factor and the $\lambda-$decrease factor.

We initialise all BO algorithms with 3 random observation data and set the observation noise to $\sigma^2_{\text{noise}} = 10^{-3}$. All experiments are repeated 40 times. In each case, the ground truth is unknown but our aim is to minimise the validation loss. Thus, the corresponding loss functions are used to compare the performance of various BO algorithms. Figure 3.6a and 3.6b shows the median of the best validation losses achieved by all BO algorithms after n iterations for the Boston and MNIST-SVM problems. Our FITBO and FITBO-MM perform competitively well

compared to their information-theoretic counterparts and all information-theoretic methods outperform EI in these real-world applications.

As for the Cancer problem (Figure 3.6c), FITBO and FITBO-MM converge to the stable median value of the validation loss at a much faster speed than MES and EI and are almost on par with PES. By examining the mean validation loss shown in the right plot of Figure 3.6c, it is evident that both FITBO and FITBO-MM demonstrate better performance than all other methods on average with FITBO gaining a slight advantage over FITBO-MM. Moreover, the comparable performance of FITBO and FITBO-MM in all three real-world tasks re-affirmed the robustness of our approach to entropy approximation as our moment matching technique, while improving the speed of the algorithm, does not really compromise the performance.

3.4 Conclusion

We have proposed a novel information-theoretic acquisition function for BO, FITBO. With the creative use of the parabolic transformation and the hyper-parameter η, FITBO enjoys the merits of less sampling effort, more flexible kernel choices and much simpler implementation in comparison with other information-based methods like PES and MES. As a result, its computational speed outperforms current information-based methods by a large margin and even exceeds EI to be on par with PI and GP-UCB. While requiring much lower runtime, it still manages to achieve satisfactory optimisation performance which is as good as or even better than PES and MES in a variety of tasks. Therefore, FITBO approach offers a very efficient and competitive alternative to existing BO approaches for expensive optimisation tasks in AutoML and beyond.

Future directions One direction to investigate is the prior distribution for η. Here we simply adopts a log-normal distribution for $y_{min} - \eta$ to ensure $\eta \leq y_{min}$. However, this might lead to unnecessary exploration after we have already sampled the global minimum. Thus, the effect of using other prior distributions, such as Gumbel distribution, on FITBO performance is worth studying to find a better

alternative prior. In addition, we linearise the parabolic transformation in FITBO to make the global minimum explicit in a tractable way. However, the trade-off between the tractability and the potential drawbacks caused by the linearisation has not been analysed in this work. Such analysis would be helpful, especially if we want to deploy the linearisation technique in other future applications.

4

Asynchronous Bayesian Optimisation with Improved Local Penalisation

The content of this chapter is based on the following paper:

> Ahsan Alvi*, Binxin Ru*[1], Jan-Peter Calliess, Stephen Roberts, and Michael A. Osborne. Asynchronous Batch Bayesian Optimisation with Improved Local Penalisation. In *International Conference on Machine Learning (ICML 2019)*, 2019.

where our contributions were listed in the acknowledgements section of this thesis.

AutoML processes, such as hyper-parameter tuning, can be significantly sped up with parallel computing resources. Batch Bayesian optimisation (BO) has been successfully developed to leverage such parallel computing and evaluate multiple configurations simultaneously at each search iteration. However, in most AutoML tasks, the evaluation runtime can vary significantly across different configurations. The synchronous batch setting can thus be wasteful of resources: workers that complete jobs ahead of others are left idle. We address this problem by developing an approach, *Penalising Locally for Asynchronous Bayesian Optimisation on k workers* (PLAyBOOK), for asynchronous parallel BO. We demonstrate empirically the

[1]* denotes equal contribution.

efficacy of PLAyBOOK and its variants on synthetic tasks and a real-world problem. We also undertake empirical comparisons between synchronous and asynchronous BO, and show that asynchronous BO often outperforms its synchronous counterpart in both wall-clock time and number of function evaluations.

4.1 Introduction

Whilst standard BO may be sufficient for many applications, it is often the case that multiple experiments can be run at the same time in parallel. For example, in the case of drug discovery, many different compounds can be tested in parallel via high throughput screening equipment (Hernández-Lobato et al., 2017), and when optimising machine learning algorithms, we can train different model configurations concurrently on multiple workers (Chen et al., 2018b; Kandasamy et al., 2018a). This observation lead to the development of parallel ("batch") BO algorithms, which, at each optimisation step, recommend a batch of k configurations to be evaluated.

In cases, such as high throughput screening, where the runtimes of tasks are roughly equal, this is usually sufficient, but, if the runtimes in a batch vary, which is often the case in AutoML processes, this will lead to inefficient utilisation of our hardware. For example, consider the optimisation of the number of units in the layers of a neural network. Training and evaluating a large network (greater number of units per layer) will take significantly longer than a small network (fewer units per layer), so for an iteration of (synchronous) batch BO to complete, we need to wait for the slowest configuration in the batch to finish, leaving the other workers idle. In order to improve the utilisation of parallel computing resources, we can run function evaluations *asynchronously*: as soon as c workers ($c < k$) complete their jobs, we choose new tasks for them.

Although asynchronous batch BO has a clear advantage over synchronous batch BO in terms of wall-clock time (Kandasamy et al., 2018a), it may lose out in terms of sample efficiency, as an asynchronous method takes decisions with less data than its synchronous counterpart at each stage of the optimisation. We investigate this empirically in this work.

Our contributions can be summarised as follows.

- We develop a new approach to asynchronous parallel BO, *Penalising Locally for Asynchronous Bayesian Optimisation on k workers* (PLAyBOOK), which uses penalisation-based strategies to prevent redundant batch selection. We show that our approach compares favourably against existing asynchronous methods.
- We propose a new penalisation function, which prevents redundant samples from being chosen. We also propose designing the penalisers using local (instead of global) variability features of the surrogate to more effectively explore the search space.
- We demonstrate empirically that asynchronous methods perform at least as well as their synchronous variants. We also show that PLAyBOOK outperforms its synchronous variants *both* in terms of wall-clock time and sample efficiency, particularly for larger batch sizes. This renders PLAyBOOK a competitive parallel BO method.

4.2 Related Work

Many different synchronous batch BO methods have been proposed over the past years. Approaches include using hallucinated observations (Ginsbourger et al., 2010b; Desautels et al., 2014), maximising the information gained about the objective function or the global minimiser (Contal et al., 2013b; Shah and Ghahramani, 2015b), and sampling-based simulation (Azimi et al., 2010; Kandasamy et al., 2018a; Hernández-Lobato et al., 2017). A recent synchronous batch BO method that demonstrated promising empirical results is Local Penalisation (LP) (González et al., 2016a). After adding a configuration $\mathbf{x}_j$ to the batch, LP penalises the value of the acquisition function in the neighbourhood of $\mathbf{x}_j$, encouraging diversity in the batch selection.

Asynchronous BO has received surprisingly little attention compared to synchronous BO to date. Ginsbourger et al. (2011) proposed a sampling-based approach

that approximately marginalises out the unknown function values at busy locations by taking samples from the posterior at those locations. Due to its reliance on sampling, it suffers from poor scaling, both in batch size and BO steps.

Wang et al. (2016a) developed an efficient global optimiser (MOE) which estimates the gradient of q-EI, a batch BO method proposed by Ginsbourger et al. (2008), and uses it in a stochastic gradient ascent algorithm to solve the prohibitively-expensive maximisation of the q-EI acquisition function, which selects all points in the batch simultaneously.

A more recent method utilizes Thompson Sampling (Kandasamy et al., 2018a) (TS) to select new batch points. This has the benefit of attractive scaling, since the method minimises samples from the surrogate model's posterior. In the case of a Gaussian process (GP) model, a batch point is placed at the minimum location of a draw from a multivariate Gaussian distribution. The disadvantage of TS is that it relies on the uncertainty in the surrogate model to ensure that the batch points are well-distributed in the search space.

4.3 Preliminaries

To perform Bayesian optimisation to find the global minimum of an expensive objective function f, we must first decide on a surrogate model for f. As discussed in Chapter 2 , a GP is a popular choice, due to its potent function approximation properties and ability to quantify uncertainty. A more detailed introduction to GPs can be found in Section 2.1 and just to quickly recap that the posterior distribution of the GP at a test input $\mathbf{x}$ is also Gaussian:

$$p(f \mid \mathcal{D}_t, \mathbf{x},) = \mathcal{N}\Big(f; \mu(\mathbf{x}), \sigma^2(\mathbf{x})\Big), \tag{4.1}$$

with mean and variance

$$\mu(\mathbf{x}) = K(\mathbf{x}, \mathbf{X}_t)K(\mathbf{X}_t, \mathbf{X}_t)^{-1}\mathbf{y}_t, \tag{4.2}$$

$$\sigma^2(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{X}_t)K(\mathbf{X}_t, \mathbf{X}_t)^{-1}K(\mathbf{X}_t, \mathbf{x}'), \tag{4.3}$$

where $\mathcal{D}_t = \{(\mathbf{x}_i, y_i)\}_{i=1}^t = \{\mathbf{X}_t, \mathbf{y}_t\}$ are the observation data. The hyper-parameters of the model have been dropped in these equations for clarity.

The second choice we make is that of the acquisition function $\alpha : \mathbb{R}^d \to \mathbb{R}$. Many different functional forms for $\alpha(x)$ have been proposed to date (Kushner, 1964; Jones et al., 1998; Srinivas et al., 2009; Hennig and Schuler, 2012b; Hernández-Lobato et al., 2014; Ru et al., 2018), each with their relative merits and disadvantages as discussed in Section 2.2. Although our method is applicable to most acquisition functions, we use the popular GP-UCB in our experiments (Srinivas et al., 2009) which is described in detail in Section 2.2.

4.4 Asynchronous vs Synchronous BO

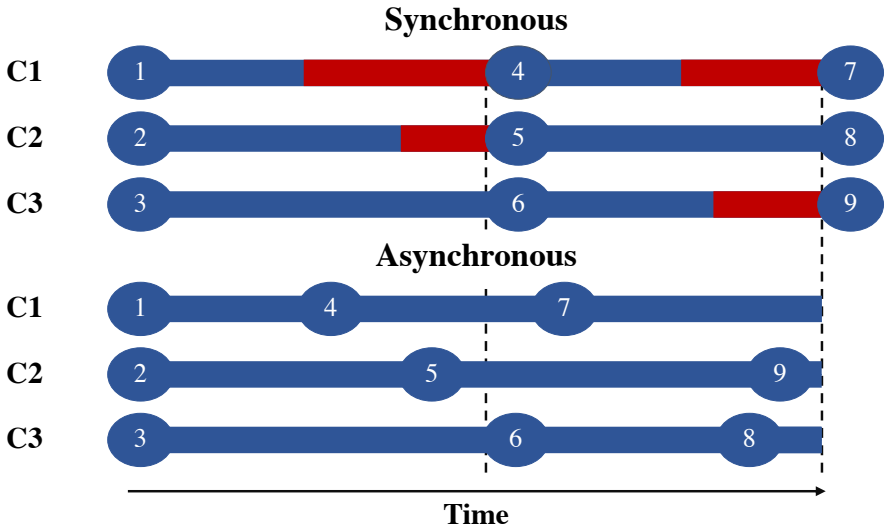


Figure 4.1: Illustration showing the difference between synchronous and asynchronous batch BO in the case of $k = 3$ parallel workers. The blue bar indicates the processing time taken for a worker to evaluate its assigned task and the red bar indicates the waiting time for a worker between completing its previous task and beginning a new task. It is clear that asynchronous batch BO, which makes better use of the computing resources, can complete a greater number of evaluations than its synchronous counterpart within the same duration.

In synchronous BO, the aim is to select a batch of promising locations $\mathcal{B} = \{\mathbf{x}_j\}_{j=1}^k$ that will be evaluated in parallel (Figure 4.1). Solving this task directly is difficult, which is why most batch BO algorithms convert this selection into

a sequential procedure, selecting one point at a time for the batch (i.e. greedy batch methods in Section 2.3.1). At the t-th BO step, the optimal choice of batch point x_j for $j \in \{1, 2, ..., k\}$ should then not only take into account our current knowledge of f, but also marginalise over possible function values at the locations $\{\mathbf{x}_i\}_{i=1}^{j-1}$ that we have chosen so far for the batch:

$$\mathbf{x}_j = \arg\max_{\mathbf{x}\in\mathcal{X}} \int \alpha(\mathbf{x} \mid \mathcal{D}_t, \mathcal{D}_{j-1}) \prod_{i=1}^{j-1} p(y_i \mid \mathbf{x}_i, \mathcal{D}_t, \mathcal{D}_{i-1}) \mathrm{d}y_i, \tag{4.4}$$

where $\mathcal{D}_t$ are the observations we have gathered so far and $\mathcal{D}_{j-1} = \{\mathbf{x}_i, y_i\}_{i=1}^{j-1}$ and $\mathcal{D}_0 = \varnothing$ (González et al., 2016a).

In asynchronous BO, the key motivation is to maximise the utilisation of our k parallel workers. After a desired number of workers $c < k$ complete their tasks, we assign new tasks to them without waiting for the remaining $b = (k - c)$ busy workers to complete their tasks. Now the general design for selecting the next query point marginalises over the likely function values both at locations under evaluation by busy workers, as well as the already-selected points in the batch:

$$\begin{aligned}\mathbf{x}_j = \arg\max_{\mathbf{x}\in\mathcal{X}} \int \alpha(\mathbf{x} \mid \mathcal{D}_{t\_}, \mathcal{D}_b, \mathcal{D}_{j-1}\}) \\ \prod_{i=1}^{j-1} p(y_i \mid \mathbf{x}_i, \mathcal{D}_{t\_}, \mathcal{D}_b, \mathcal{D}_{i-1}) \prod_{l=1}^{b} p(y_l \mid \mathbf{x}_l, \mathcal{D}_{t\_}) \mathrm{d}y_i,\end{aligned} \tag{4.5}$$

where $\mathcal{D}_b = \{\mathbf{x}_i, y_i\}_{i=1}^{b}$ represents the locations and function values of the busy locations. $\mathcal{D}_{t\_}$ are the observations available at the point of constructing the asynchronous batch. In general, $\mathcal{D}_{t\_}$ contains fewer observations than the $\mathcal{D}_t$ that would be used to select the equivalent batch of evaluations in the synchronous setting. Figure 4.1 shows the case of $c = 1$ and thus $b = k - 1 = 2$.

In a given period of time, asynchronous batch BO is able to process a greater number of evaluations than the synchronous approach: asynchronous BO offers clear gains in resource utilisation. However, Kandasamy et al. (2018a) claim that the asynchronous setting does not lead to better performance when measured by the number of evaluations. The authors point out that a new evaluation in a sequentially-selected synchronous batch will be selected with at most $k - 1$ evaluations "missing"

(that is, with knowledge of their locations $\mathbf{x}$ but absent the knowledge of their values y), corresponding to the previously-selected points in the current batch (i.e. $j-1 \leq k-1$ in Equation (4.4)). Evaluations in the asynchronous case are always chosen with $k-1$ "missing" evaluations. Take the example of task points $\{7, 8, 9\}$ in Figure 4.1. In the synchronous setting, task 7 is selected informatively by conditioning on past data. task 8 is selected with one missing evaluation at 7 and task 9 is chosen with two missing evaluations at tasks 8 and 7. But in the asynchronous case, each of the three tasks is chosen with two missing evaluations.

However, to our knowledge, there exists little empirical investigation of the performance difference between synchronous and asynchronous batch methods. We conducted this comparison on a large set of benchmark test functions and found that asynchronous batch BO can be as good as synchronous batch BO for different batch selection methods. Additionally, for the penalisation-based methods we propose, asynchronous operation often outperforms the synchronous setting, particularly as the batch size increases. We will discuss this interesting empirical observation in Section 4.6.2.

4.5 Penalisation-based Asynchronous BO

We now present our core algorithmic contributions. As discussed in Section 4.2, the existing asynchronous BO methods suffer drawbacks such as the prohibitively high cost of repeatedly updating GP surrogates when selecting batch points (Ginsbourger et al., 2011) or the risk of redundant sampling at or near a busy location in the batch (Kandasamy et al., 2018a). In view of these limitations, we propose a penalisation-based asynchronous method which encourages sampling diversity among the points in the batch as well as eliminating the risk of repeated samples in the same batch. Our proposed method remains computationally efficient, and thus scales well to large batch sizes.

Inspired by the Local Penalisation approach (LP) in synchronous BO (González et al., 2016a), we approximate Equation (4.5) for the case of $c = 1$ as:

$$\mathbf{x}_j = \arg\max_{\mathbf{x}\in\mathcal{X}} \left\{ \alpha(\mathbf{x} \mid \mathcal{D}_{t\_}) \prod_{i=1}^{k-1} \phi(\mathbf{x} \mid \mathbf{x}_i, \mathcal{D}_{t\_}) \right\}, \tag{4.6}$$

where $\phi(\mathbf{x} \mid \mathbf{x}_i, \mathcal{D}_{t\_})$ is the penaliser function centred at the busy locations $\{\mathbf{x}_i\}_{i=1}^{k-1}$. In the following subsections, we design effective penaliser functions by harnessing the Lipschitz properties of the function and its GP posterior. To simplify notation, we denote $\phi(\mathbf{x} \mid \mathbf{x}_i, \mathcal{D}_{t\_})$ as $\phi(\mathbf{x} \mid \mathbf{x}_i)$ and $\alpha(\mathbf{x} \mid \mathcal{D}_{t\_})$ as $\alpha(\mathbf{x})$ in the remainder of the section.

4.5.1 Hard Local Penaliser

Assume the unknown objective function is Lipschitz continuous with constant L and has a global minimum value $f(\mathbf{x}^*) = M$ and $\mathbf{x}_j$ is a busy task,

$$|f(\mathbf{x}_j) - M| \leq L\|\mathbf{x}_j - \mathbf{x}^*\|. \tag{4.7}$$

This implies $\mathbf{x}^*$ cannot lie within the spherical region centred on $\mathbf{x}_j$ with radius $r_j = \frac{f(\mathbf{x}_j)-M}{L}$:

$$\mathbb{S}(\mathbf{x}_j, r_j) = \mathcal{X} \ \{\mathbf{x} \in \mathcal{X} : \|\mathbf{x} - \mathbf{x}_j\| \leq r_j\}. \tag{4.8}$$

If $\mathbf{x}_j$ is still under evaluation by a worker, there is no need for any further selections inside $\mathbb{S}(\mathbf{x}_j, r_j)$.

Given that $f(\mathbf{x}_j) \sim \mathcal{N}(\mu(\mathbf{x}_j), \sigma^2(\mathbf{x}_j))$ and thus $\mathbb{E}(r_j) = \frac{|\mu(\mathbf{x}_j)-M|}{L}$, applying Hoeffding's inequality for all $\epsilon > 0$ (Jalali et al., 2013) gives

$$P\left(r_j > \mathbb{E}(r_j) + \epsilon\right) \leq \exp\left(-\frac{2\epsilon^2 L^2}{\sigma(\mathbf{x}_j)^2}\right), \tag{4.9}$$

If we substitute $\epsilon = \frac{1.5\sigma(\mathbf{x}_j)}{L}$ into Equation (4.9):

$$\begin{aligned} P\left(r_j > \mathbb{E}(r_j) + \frac{1.5\sigma(\mathbf{x}_j)}{L}\right) &\leq \exp\left(-\frac{2L^2}{\sigma(\mathbf{x}_j)^2}\left(\frac{1.5\sigma(\mathbf{x}_j)}{L}\right)^2\right) \\ &= \exp(-2 \times 1.5^2) \\ &\approx 0.0111 \end{aligned}$$

Thus,

$$P\left(r_j \leq \mathbb{E}(r_j) + \frac{1.5\sigma(\mathbf{x}_j)}{L}\right) \geq 0.99 \tag{4.10}$$

which implies there is a high probability (around 99%) that $r_j \leq \frac{|\mu(\mathbf{x}_j)-M|}{L} + 1.5\frac{\sigma(\mathbf{x}_j)}{L}$.

The penalisation function $\phi(\mathbf{x} \mid \mathbf{x}_j)$ should incorporate this belief to guide the selection of the next asynchronous batch point by reducing the value of the acquisition function at locations $\{\mathbf{x} \in \mathbb{S}(\mathbf{x}_j, r_j)\}$. A valid penaliser should possess the several properties:

- the penalisation region shrinks as the expected function value at $\mathbf{x}_j$ gets close to the global minimum (i.e. small $|\mu(\mathbf{x}_j) - M|$) (González et al., 2016a);
- the penalisation region shrinks as L increases (González et al., 2016a);
- the extent of penalisation on $\alpha(\mathbf{x})$ increases as $\mathbf{x}$ gets closer to $\mathbf{x}_j$ with $\alpha(\mathbf{x}_j) = 0$ if $\alpha(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathcal{X}$.

The Local Penaliser (LP) in González et al. (2016a) fulfils the first two properties but not the final one which we believe is crucial. Thus, directly using it for the asynchronous case makes the algorithm vulnerable to redundant sampling as illustrated in Figure 4.2. In view of this limitation, we propose a simple yet effective Hard Local Penaliser (HLP) which satisfies all three conditions

$$\phi(\mathbf{x} \mid \mathbf{x}_j) = \min\left\{\frac{\|\mathbf{x} - \mathbf{x}_j\|}{\mathbb{E}(r_j) + \gamma\frac{\sigma(\mathbf{x}_j)}{L}}, 1\right\}, \tag{4.11}$$

where γ is a constant.

The above expression can be made differentiable by the approximation:

$$\hat{\phi}(\mathbf{x} \mid \mathbf{x}_j) = \left[\left(\frac{\|\mathbf{x} - \mathbf{x}_j\|}{\mathbb{E}(r_j) + \gamma\frac{\sigma(\mathbf{x}_j)}{L}}\right)^p + 1^p\right]^{1/p}, \tag{4.12}$$

with $\hat{\phi}(\mathbf{x} \mid \mathbf{x}_j) \to \phi(\mathbf{x} \mid \mathbf{x}_j)$ as $p \to -\infty$.

In addition, the global optimum M is unknown in practice and is usually approximated by the best function value observed $\hat{M} = \min\{f(\mathbf{x}_i)\}_i^n$ (González et al., 2016a). This approximation tends to lead to underestimation of $\mu(\mathbf{x}_j) - M$ and

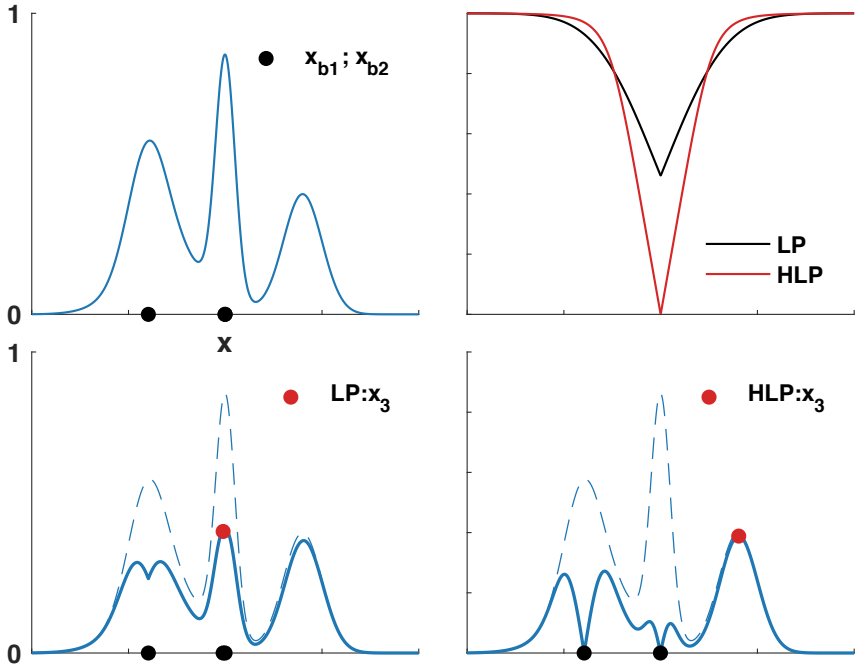
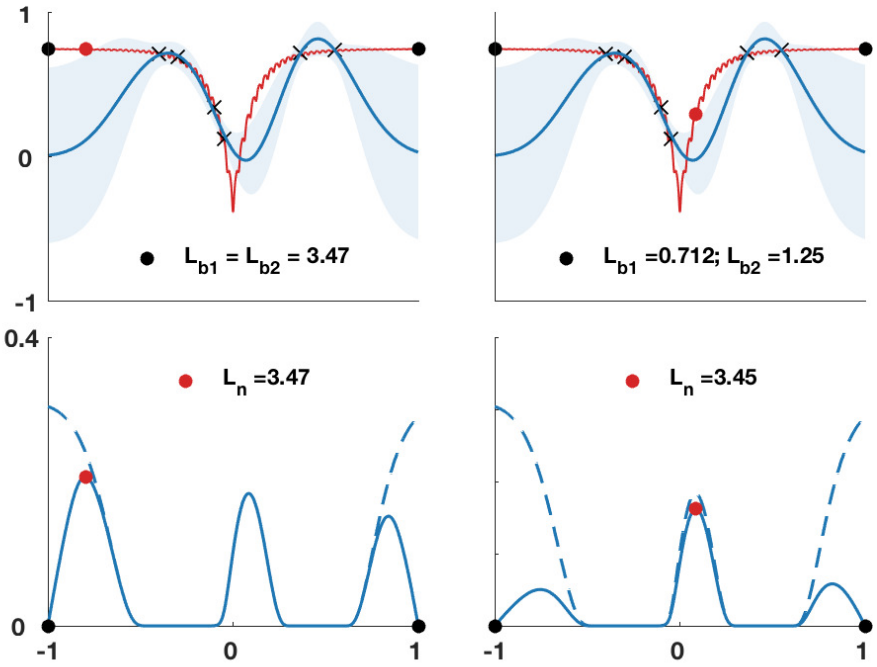


Figure 4.2: Illustration of asynchronous batch selection by naïve LP and HLP. The top left plot shows the acquisition function $\alpha(\mathbf{x})$ and the locations (i.e. $\mathbf{x}_{b1}$ and $\mathbf{x}_{b2}$ denoted in black dots) under evaluation by busy workers. The top right plot shows the shapes of two penalisers at the busy location $\mathbf{x}_{b1}$. Their respective penalisation effects on $\alpha(\mathbf{x})$ at $\mathbf{x}_{b1}$ and $\mathbf{x}_{b2}$ as well as the new batch point $\mathbf{x}_3$ to be assigned to the available worker are shown in the subplots that follow, LP on the left and HLP on the right.

thus $\mathbb{E}(r_j)$, reducing the extent of the penalisation at $\mathbf{x}_j$ and in the region nearby. HLP mitigates this effect by penalising significantly harder than the penaliser proposed by González et al. (2016a), and maximally at $\mathbf{x}_j$ ($\alpha(\mathbf{x}_j) = 0$). Thus, our method is less affected by over-estimation of the global minimum $\hat{M} > M$.

4.5.2 Local Estimated Lipschitz Constants

In BO, the global Lipschitz constant L of the objective function is unknown. Assuming the true objective function f is a draw from its GP surrogate model, we can approximate L with $\hat{L} = \max_{\mathbf{x}\in\mathcal{X}} \|\mu_{\nabla}(\mathbf{x})\|$ where $\mu_{\nabla}(\mathbf{x})$ is the posterior mean of the derivative GP (González et al., 2016a). However, using the estimated global Lipschitz constant $\hat{L}$ to design the shape of the penalisers at all busy locations in the batch may not be optimal. Consider the case where a point in an unexplored region is still under evaluation. If $\hat{L}$ is large, then the penaliser's



(a) Penalisation with global L **(b)** Penalisation with local L

Figure 4.3: Different penalisation effects on $\alpha(x)$ of using a single global Lipschitz constant compared to local Lipschitz constants. The top plots in both (a) and (b) show the true objective function (red line), six observed points (black crosses), the GP posterior mean (black line) and variance (blue shade), the two busy locations (black dots) and the next query point (red dot) selected by using the HLP with global and local Lipschitz constants respectively. The plots in (a) show the penalisation effect on busy locations using the same global Lipschitz constant while those in (b) show the effect of using local Lipschitz constants. It is clear that penalising the busy locations based on local Lipschitz constants allows the algorithm to capture the informative peak at the central region while selection based on the single global Lipschitz constant leads us to revisit the flat region near the boundary due to insufficient penalisation at $\mathbf{x}_1$.

radius will be small and we will end up selecting multiple points in the same unexplored region, which is undesirable.

Therefore, we propose to use a separate Lipschitz constant, which is locally estimated, for each busy location. Here, "locality" is encoded in our choice of kernel and its hyper-parameters, e.g. via the lengthscale parameter in the Matérn class of kernels. The use of local Lipschitz constants will enhance the efficiency of exploration because they allow the penaliser to create larger exclusion zones in areas in which we are very uncertain (the surrogate model is near its prior or has low curvature) and smaller penalisation zones in interesting, high-variability, areas.

We demonstrate the different effects of using approximate global and local Lipschitz constants with a qualitative example. In Figure 4.3a, the estimated global Lipschitz constant is used for penalisation at both busy locations $\mathbf{x}_{b1} = -1$ and $\mathbf{x}_{b2} = 1$ (denoted as black dots). The relatively large value of the global Lipschitz constant ($\hat{L}_{b1} = \hat{L}_{b2} = \hat{L} = 3.47$) due to the high curvature of the surrogate in the central region leads to a small penalisation zone around the two busy locations at the boundary. This causes the algorithm to miss the informative region in the centre and instead revisit the region near $\mathbf{x}_{b1}$ to choose the new point in the asynchronous batch. On the other hand, in Figure 4.3b, the use of a locally estimated Lipschitz constant allows us to penalise a larger zone around points where the surrogate is relatively flat ($\hat{L}_{b1} = 0.712$ for $\mathbf{x}_{b1}$), while still penalising smaller regions where there is higher variability ($\hat{L}_3 = 3.45$ at $\mathbf{x}_3$).

In our experiments we used a Matérn-52 kernel and defined the local region for evaluating the Lipschitz constant for a batch point $\mathbf{x}_j$ to be a hypercube centred on $\mathbf{x}_j$ with the length of the each side equal to the lengthscale corresponding to that input dimension.

In summary, we propose a new class of asynchronous BO methods, Penalisation Locally for Asynchronous Bayesian Optimisation Of K workers (PLAyBOOK), which uses analytic penaliser functions to prevent redundant sampling at or near the busy locations in the batch and encourage desirably explorative searching behaviour. We differentiate between PLAyBOOK-L, which uses a naïve Local penaliser, PLAyBOOK-H, that uses the HLP penaliser, as well as their variations with locally estimated Lipschitz constants, PLAyBOOK-LL and PLAyBOOK-HL.

4.6 Experiments

We begin our empirical investigations by performing a head-to-head comparison of synchronous and asynchronous BO methods, to test the intuitions described in Section 4.4. We specifically look at optimisation performance for asynchronous and synchronous variants of the parallel BO methods measured over time and

number of evaluations, and we show empirically that asynchronous is preferable over synchronous BO on both counts.

We then experiment with our proposed asynchronous methods (PLAyBOOK-L, PLAyBOOK-H, PLAyBOOK-LL and PLAyBOOK-HL) on a number of benchmark test functions as well as a real-world expensive optimisation task. Our methods are compared against the state-of-the-art asynchronous BO methods, Thompson sampling (TS) (Kandasamy et al., 2018a), as well as the Kriging Believer heuristic method (KB) (Ginsbourger et al., 2010b) applied asynchronously.

For all the benchmark functions, we measure the log of the simple regret R, which is the difference between the true minimum value $f(\mathbf{x}^*)$ and the best value found by the BO method:

$$\log(R) = \log \left| f(\mathbf{x}^*) - \min_{i=1,\ldots,n} f(\mathbf{x}_i) \right|. \tag{4.13}$$

4.6.1 Implementation Details

To ensure a fair comparison, we implemented all methods in Python using the same packages[2].

In all experiments, we used a zero-mean Gaussian process surrogate model with a Matérn-52 kernel with ARD. We optimised the kernel and likelihood hyper-parameters by maximising the log marginal likelihood. For the benchmark test functions, we fixed the noise variance to $\sigma^2_{\text{noise}} = 10^{-6}$ and started with $3 * d$ random initial observations. Each experiment was repeated with 30 different random initialisations and the input domains for all experiments were scaled to $[-1, 1]^d$.

All methods except TS used UCB as the acquisition function $\alpha(\mathbf{x})$. For our PLAyBOOK-H and PLAyBOOK-HL, we choose $\gamma = 1$ and $p = -5$ in the HLP (Equation (4.12)). For TS, we use 10,000 sample points for each batch point selection. For the other methods, we evaluate $\alpha(\mathbf{x})$ at 3,000 random locations and then choose the best one after locally optimising the best 5 samples for a small number of local optimisation steps.

[2] Implementation will be made available via a GitHub repo.

We evaluate the performance of the different batch BO strategies using popular global optimisation test functions[3]. We show results for the Eggholder function defined on $\mathbb{R}^2$ (egg-2D), the Ackley function defined on $\mathbb{R}^5$ and the Michalewicz function defined on $\mathbb{R}^{10}$ (mic-10D).

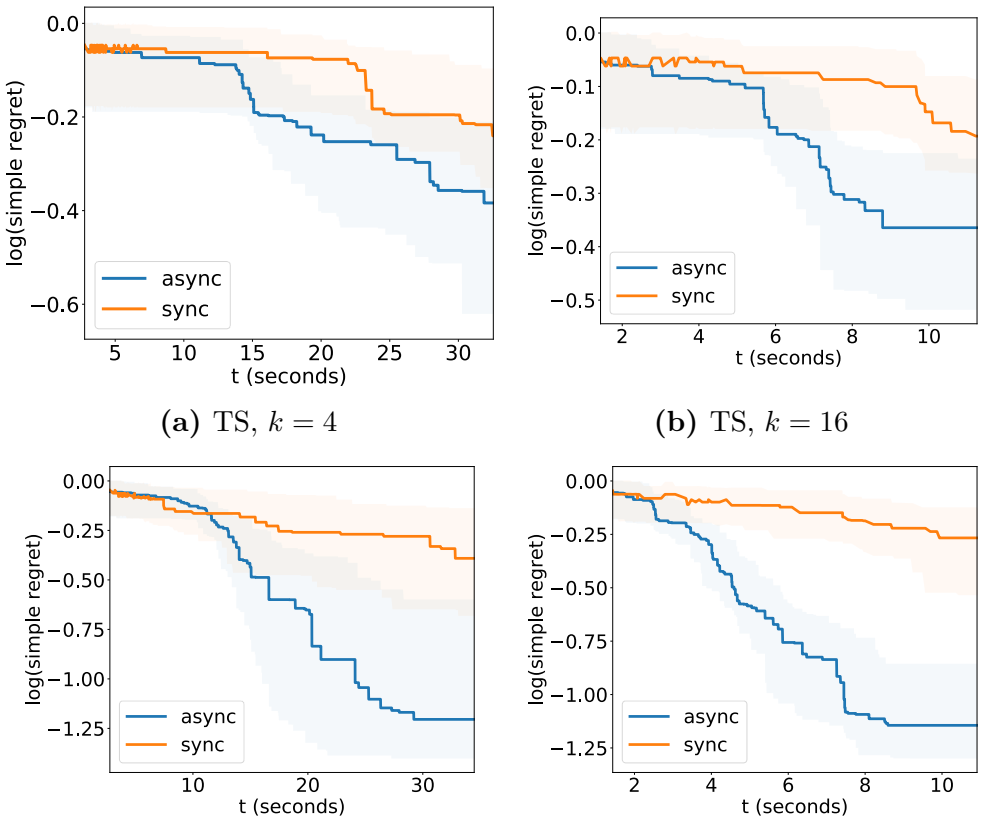
4.6.2 Synchronous vs Asynchronous BO

In this section we address the question of choosing between asynchronous and synchronous BO. In order to investigate their relative merits, we compared asynchronous and synchronous BO methods' performance as a function of wall-clock time and number of evaluations.

Evaluation Time

In order to facilitate this comparison, we needed to inject a measure of runtime for different tasks, as the test functions can be evaluated instantaneously. We followed the procedure proposed in Kandasamy et al. (2018a) to sample an evaluation time for each task so as to simulate the asynchronous setting. We chose to use a half-normal distribution with scale parameter $\sigma = \sqrt{\pi/2}$, which gives us a distribution of runtime values with mean at 1.

Results on the ack-5D task are shown in Figure 4.4. We know that asynchronous BO has the advantage over synchronous BO in terms of utilisation of resources, as shown qualitatively in Figure 4.1, simply due to the fact that any available worker is not required to wait for all evaluations in the batch to finish before moving on. Therefore, given the same time budget, a greater number of evaluations can be taken in the asynchronous setting than in the synchronous setting, which, as confirmed by our experiments, translates to faster optimisation of f in terms of the total (wall-clock) time spent evaluating tasks.



(a) TS, $k = 4$ **(b)** TS, $k = 16$

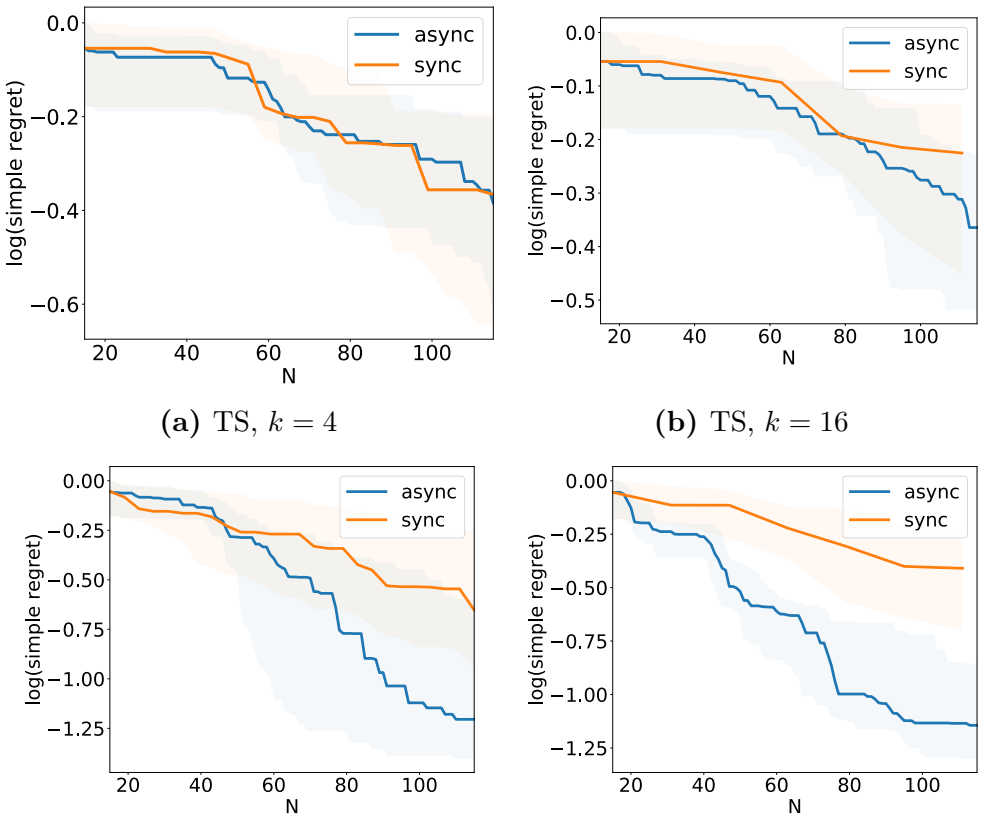
(c) PLAyBOOK-HL, $k = 4$ **(d)** PLAyBOOK-HL, $k = 16$

Figure 4.4: A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO method over *evaluation time*, t. The median (solid line) and quartiles (shaded region) of the regret for optimising ack-5D for 30 random initialisations are shown. As expected, asynchronous methods clearly outperform their synchronous counterparts in terms of evaluation time.

Number of Evaluations

A more interesting question to answer is whether asynchronous BO methods are really less data efficient than synchronous BO methods as discussed in Section 4.4. Figure 4.5 shows a subset of the experiments we conducted. More results can be found in Appendix A.3. An unexpected yet interesting behaviour we note is that as k increases, the PLAyBOOK methods tend to clearly outperform their respective synchronous counterparts even in terms of sample efficiency. This observation runs counter to the guidance provided in Kandasamy et al. (2018a) and such behaviour

[3]Details for these and other challenging global optimisation test functions can be found at https://www.sfu.ca/~ssurjano/optimization.html



(a) TS, $k = 4$ **(b)** TS, $k = 16$

(c) PLAyBOOK-HL, $k = 4$ **(d)** PLAyBOOK-HL, $k = 16$

Figure 4.5: A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO method over *number of evaluations, N*. The median (solid line) and quartiles (shaded region) of the regret for optimising ack-5D for 30 random initialisations are shown. Surprisingly, asynchronous methods also outperform their synchronous counterparts in terms of sample efficiency.

is less evident for the other two batch methods, TS and KB.

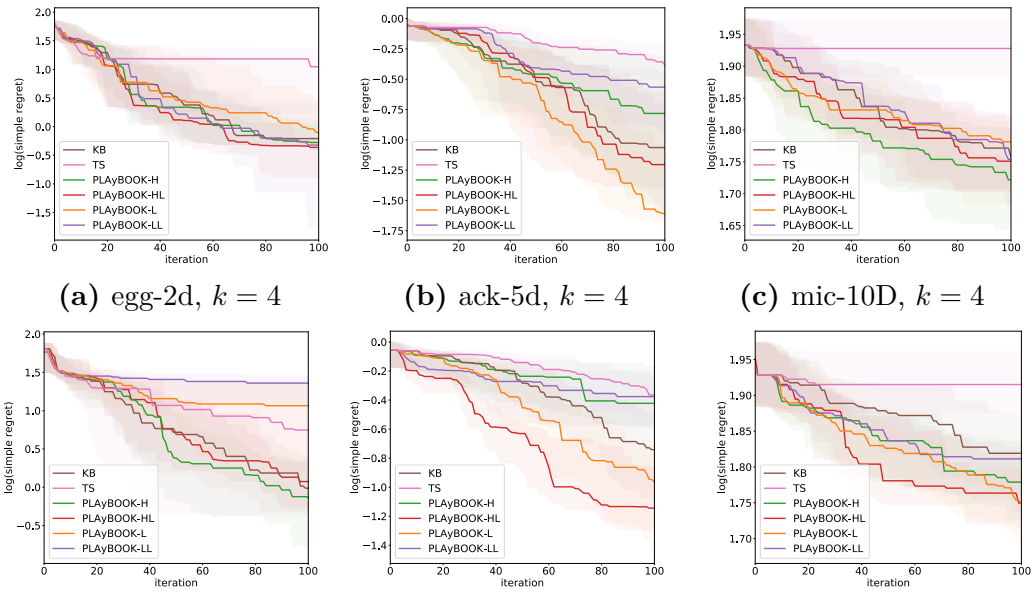
We think this observation may be explained by the difference in nature between the PLAyBOOK and TS/KB: in the case of TS we rely on stochasticity in sampling, and in KB we are re-computing the posterior variance and $\alpha(\mathbf{x})$ each time a batch point is selected. The penalisation-based methods, on the other hand, simply down-weight the acquisition function, and in the synchronous case these penalisers coincide with the high-value regions of the acquisition function. This means that unless the acquisition function has a large number of spaced-out peaks, we will quickly be left without high-utility locations to choose new batch points from.

This seems to be the reason for the superior performance of asynchronous

PLAyBOOK methods over their synchronous variants because they benefit from the fact that the busy locations being penalised do not necessarily coincide with the peaks in $\alpha(\mathbf{x})$, as the surrogate used to compute $\alpha(\mathbf{x})$ is more informed than the one used to decide the locations of the busy locations previously. This means that points with high utilities are more likely to to be preserved.

Taking into account the fact that the asynchronous PLAyBOOK methods tend to perform at least equally well, if not significantly better than their synchronous variants on both time and efficiency, and that the asynchronous PLAyBOOK methods gain more advantage over synchronous ones as the batch size increases, we believe that this points to the fact that penalisation-based methods are inherently better suited as asynchronous methods. Hence, for users that are running parallel BO and have selected LP, we recommend they consider running PLAyBOOK instead due to its attractive benefits.

4.6.3 Asynchronous Parallel BO



(a) egg-2d, $k = 4$ **(b)** ack-5d, $k = 4$ **(c)** mic-10D, $k = 4$

(d) egg-2d, $k = 16$ **(e)** ack-5d, $k = 16$ **(f)** mic-10D, $k = 16$

Figure 4.6: The median (solid line) and quartiles (shaded region) of the regret for different asynchronous BO methods on the global optimisation test functions for 30 random intialisations is shown. We can see that our proposed PLAyBOOK methods perform competitively, especially when we start choosing larger batch sizes.

Now that we have strengthened the appeal of asynchronous BO, we turn to evaluating PLAyBOOK against existing asynchronous BO methods.

Synthetic Experiments

We ran PLAyBOOK and competing asynchronous BO methods on the global optimisation test functions described in Section 4.6.1. The results are shown in Figure 4.6, and more results on different optimisation problems are provided in Appendix A.3. On the global optimisation test functions we noted that in most cases PLAyBOOK outperforms the alternative asynchronous methods TS and KB. The TS algorithm performed poorly on this test, which we believe is caused by the fact that TS relies heavily on the surrogate's uncertainty to explore new regions. PLAyBOOK methods show strong performance, achieving better optimisation performance than both TS and KB baselines.

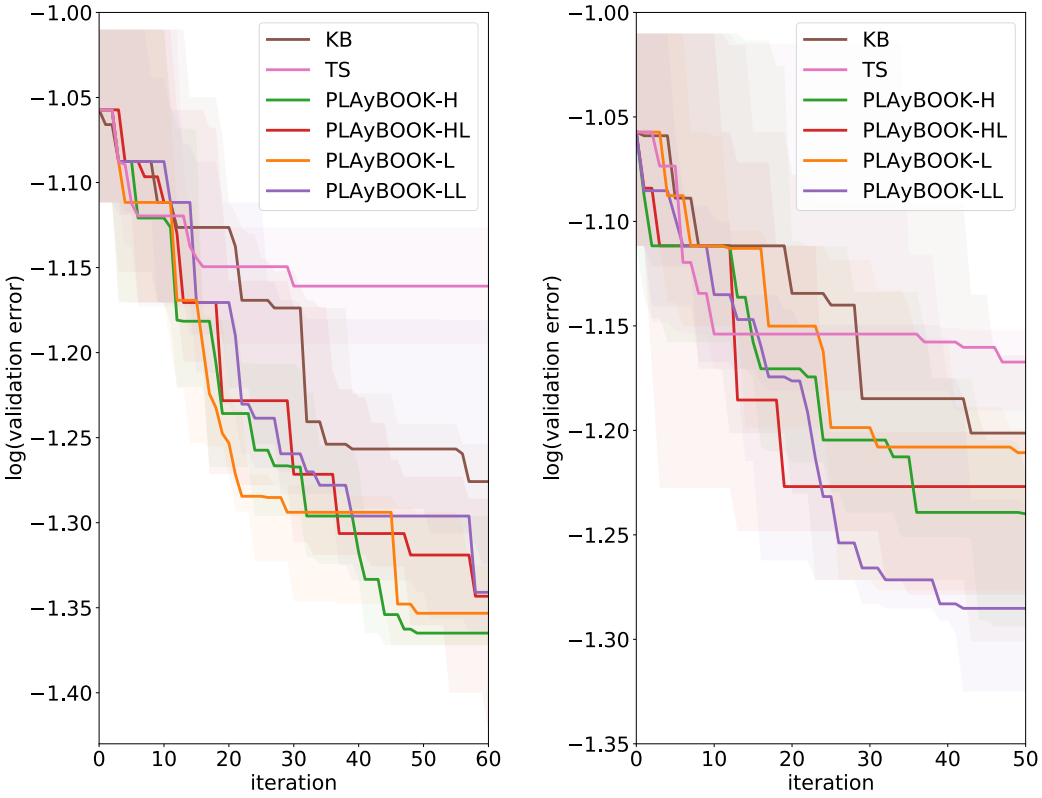
Real-world Optimisation

Table 4.1: Test accuracy (%) on CIFAR-10 after training the best model chosen by various asynchronous BO methods for 80 epochs

k	TS	KB	PLAyBOOK			
			L	H	LL	HL
2	81.0	83.9	84.7	**85.2**	84.1	84.9
4	81.2	82.8	82.5	83.8	**84.2**	83.0

We further experimented on a real-world application of tuning the hyper-parameters of a 6-layer Convolutional Neural Network (CNN)[4] for an image classification task on CIFAR10 dataset (Krizhevsky et al., 2009). The 9 hyper-parameters that we optimise with BO are the learning rate and momentum for the stochastic gradient descent training algorithm, the batch size used and the number of filters in each of the six convolutional layers. We trained the CNN on half of the

[4]Follow the implementation in https://blog.plon.io/tutorials/cifar-10-classification-using-keras-tutorial/



(a) $k = 2$ **(b)** $k = 4$

Figure 4.7: Asynchronous optimisation of 9 hyper-parameters of a 6-layer CNN for image classification on the CIFAR10 dataset. The network is trained on half of the training set and evaluated on the second half. The objective being minimised is the classification accuracy on the validation set. PLAyBOOK outperforms both KB and TS in this expensive optimisation task.

training set for 20 epochs and each function evaluation returns the validation error of the model. We tested the use of $k = 2$ and $k = 4$ parallel workers to run this real-world experiment. The results are shown in Figs. 4.7a and 4.7b.

We can see that for both $k = 2$ and $k = 4$ parallel settings, all PLAyBOOK methods outperform the other asynchronous methods, TS and KB. In the case of $k = 2$ (2 parallel processors), only one busy location is penalised in each batch so there is little gain from using a locally estimated Lipschitz constant. However, as the batch size increases to $k = 4$, we see that methods using estimated Lipschitz constants (PLAyBOOK-LL and PLAyBOOK-HL) show faster decrease in validation error than PLAyBOOK-L and PLAyBOOK-H with PLAyBOOK-LL demonstrating the best performance.

We then took the final configurations recommended by each asynchronous BO method in the $k = 2$ and $k = 4$ settings and retrained the CNN model on the full training set of 50K images for 80 epochs. The accuracy on the test set of 10K images achieved with the best model chosen by each BO method is shown in Table 4.1. In both settings, our PLAyBOOK methods achieve superior performance over TS with PLAyBOOK-H providing the best test accuracy when $k = 2$ and PLAyBOOK-LL doing the best when $k = 4$.

4.7 Conclusions

In this work, we argue for the use of asynchronous (over synchronous) Bayesian optimisation (BO), and provide extensive empirical evidence. Additionally, we developed a new asynchronous BO approach, PLAyBOOK, which penalises the acquisition function values based on the surrogate information about the objective problem at locations that are still under evaluation. PLAyBOOK achieves highly competitive performance on various synthetic functions and a real-world optimisation task. More importantly, PLAyBOOK is more efficient than synchronous BO in both wall-clock time and the number of queries. This makes PLAyBOOK an ideal asynchronous BO option to make efficient use of parallel computing for AutoML processes.

Future directions In Section 4.6.2, we provide an intuitive explanation for the surprising result that asynchronous PLAyBOOK outperforms its synchronous counterparts even when measured against number of evaluations. A theoretical analyses on this phenomenon would be an interesting extension, which can further encourage the use of asynchronous parallel BO when combined with penalisation-based batch selection. Moreover, the estimation quality of the global optimum and the Lipschitz constant is crucial to the performance of PLAyBOOK. Better estimation methods are worth exploring. For example, PLAyBOOK could be combined with the information-theoretic acquisition functions discussed in Chapter 3 to reuse the global optimum samples learnt by such acquisition functions to

design the penaliser. Recently, Cardelli et al. (2019) derives robustness measure for GP against input perturbations, which could be adapted to efficiently derive bounds on the Lipschitz constant.

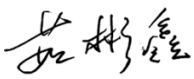
Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Asynchronous Batch Bayesian Optimisation with Improved Local Penalisation
Publication Status	☒Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Ahsan Alvi*, Binxin Ru*, Jan-Peter Calliess, Stephen Roberts, and Michael A. Osborne. Asynchronous Batch Bayesian Optimisation with Improved Local Penalisation. In *International Conference on Machine Learning (ICML 2019),* 2019.

Student Confirmation

Student Name:	Binxin Ru		
Contribution to the Paper	Alvi and I contributed equally to the research idea discussions, experiments, code reviews and result analyses as well as the write-up of the paper. Specifically, I led the design of the local penalization function, created all the illustrative plots in the paper and performed the real-world experiment on tuning CNN hyper-parameters. Alvi led the asynchronous design in the methods, build the initial code repository and performed all other experiments.		
Signature		Date	01/09/2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof. Michael A. Osborne		
Supervisor comments To the best of my knowledge, all above seems fair and correct.		
Signature 	Date	17 September 2021

This completed form should be included in the thesis, at the end of the relevant chapter.

5

Bayesian Optimisation over Multiple Continuous and Categorical Inputs

The content of this chapter is based on the following paper:

> Binxin Ru*, Ahsan Alvi*, Vu Nguyen, Michael A. Osborne, and Stephen Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning (ICML 2020)*, 2020.
>
> Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A. Osborne. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. In *International Conference on Machine Learning (ICML 2021)*, 2021.

where our contributions were listed in the acknowledgements section of this thesis.

The advancements in acquisition functions and parallel batch designs have significantly improved the search efficiency of Bayesian Optimisation (BO), making it the default choice for black-box problems involving continuous inputs such as tuning the learning rate and the regularisation terms for training a deep neural network. However, many important black-box problems often comprise both continuous and categorical inputs and efficient optimisation over such mixed input space still poses

significant challenges. Current approaches, like one-hot encoding, severely increase the dimension of the search space, while separate modelling of category-specific data is sample-inefficient. Both frameworks are not scalable to practical applications involving *multiple* categorical variables, each with *multiple* possible category choices. We propose a new approach, Continuous and Categorical Bayesian Optimisation (CoCaBO), which combines the strengths of multi-armed bandits (MABs) and BO to decide values for both categorical and continuous inputs. We model this mixed-type space using a Gaussian Process (GP) kernel, designed to allow sharing of information across *multiple* categorical variables; this allows CoCaBO to leverage all available data efficiently. We also build a batch version of our method and propose an efficient batch selection procedure that dynamically balances exploration and exploitation whilst encouraging batch diversity. In addition, we further extend CoCaBO to high-dimensional problems involving such mixed input types by leveraging the concept of local trust regions. We demonstrate empirically that our methods outperform existing approaches on both synthetic and real-world optimisation tasks with multiple continuous and categorical inputs.

5.1 Introduction

Existing work has shown BO to be remarkably successful at optimising functions with continuous input spaces (Snoek et al., 2012a; Hennig and Schuler, 2012a; Hernández-Lobato et al., 2015; Ru et al., 2018; Shahriari et al., 2016a; Frazier, 2018; Alvi et al., 2019). However, optimisation problems in many situations, including but not limited to those in AutoML processes, involve a mixture of continuous and categorical variables. For example, with a deep neural network, we may want to adjust the learning rate and the weight decay (continuous), as well as the activation function type in each layer (categorical). Similarly, in a gradient boosting ensemble of decision trees, we may wish to adjust the learning rate and the maximum depth of the trees (continuous), as well as the boosting algorithm and loss function (categorical).

Having a mixture of categorical and continuous variables presents challenges. If some inputs are categorical variables, then the common assumption that the

BO acquisition function is differentiable over the input space, which allows the acquisition function to be efficiently optimised, is no longer valid. Recent research has dealt with categorical variables in different ways. The simplest approach for BO with GP (Rasmussen and Williams, 2006a) surrogates is to use a one-hot encoding on the categorical variables, so that they can be treated as continuous variables, and perform BO on the transformed space (authors, 2016). Alternatively, the mixed-type inputs can be handled using a hierarchical structure, such as using random forests (Hutter et al., 2011a; Bergstra et al., 2011a) or MABs (Gopakumar et al., 2018). These approaches come with their own challenges, which we will discuss below (see Section 5.3). In particular, the existing approaches are not well designed for *multiple* categorical variables with *multiple* possible values. Additionally, no GP-based BO methods have explicitly considered the batch setting for continuous-categorical inputs, to the best of our knowledge.

In this chapter, we present a new Bayesian optimisation approach for optimising a black-box function with multiple continuous and categorical inputs, termed Continuous and Categorical Bayesian Optimisation (CoCaBO). Our main contributions are as follows:

- We propose a method which combines the strengths of MABs and BO to optimise black-box functions with *multiple* categorical and continuous inputs. (Section 5.4).

- We present a GP kernel to capture complex interactions between the continuous and categorical inputs (Section 5.4.2). Our kernel allows sharing of information across different categories without resorting to one-hot transformations.

- We introduce a novel batch selection method that extends CoCaBO to the parallel setting, and dynamically balances exploration and exploitation and encourages batch diversity (Section 5.4.3).

- We demonstrate the effectiveness of our CoCaBO method on a variety of synthetic and real-world optimisation tasks with *multiple* categorical and continuous inputs (Section 5.5).

- We make an extension work which builds on the CoCaBO kernel design and the local trust region idea (Eriksson et al., 2019) to effectively tackle high-dimensional problems with such mixed input types (Section 5.6).

5.2 Preliminaries

Problem Definition

In this chapter, we consider the problem of optimising a black-box function $f(\mathbf{z})$ where the input $\mathbf{z}$ consists of both continuous and categorical inputs, $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$, where $\mathbf{h} = [h_1, \ldots, h_{d_h}]$ are the categorical variables, with each variable $h_j \in \{1, 2, \ldots, N_j\}$ taking one of N_j different values, and $\mathbf{x}$ is a point in a d-dimensional hypercube $\mathcal{X}$. Formally, we aim to find the best configuration to maximise the black-box function

$$\mathbf{z}^* = [\mathbf{h}^*, \mathbf{x}^*] = \arg\max_{\mathbf{z}} f(\mathbf{z}) \tag{5.1}$$

by making a series of evaluations $\mathbf{z}_1$,..., $\mathbf{z}_T$. Later we extend our method to allow parallel evaluation of multiple points, by selecting a batch $\{\mathbf{z}_t^{(i)}\}_{i=1}^{b}$ at each optimisation step t. For an overview on BO and GP, please refer to Chapter 2.

Exponential-weight Algorithm for Exploration and Exploitation (EXP3)

The specific MABs method adopted in our CoCaBO approach is EXP3 (Auer et al., 2002b). EXP3 deals with the non-stochastic, adversarial multi-arm bandit problem. In such general problem setting, we assume that the rewards g_t are chosen by an adversary at each iteration. Thus, EXP3 can handle any form of non-stationarity in the reward distribution (Allesiardo et al., 2017).

Algorithm 4 EXP3 Algorithm

Input: A reward function g, the trade-off parameter $\gamma \in [0, 1]$, the set of categorical choices $\{1, 2, \ldots, N\}$, maximum number of iterations T
Initialise the weights $w_0(j) = 1$ for $j = 1, 2, \ldots, N$
for $t = 1, \ldots, T$ **do**
  Set $p_t(j) = (1 - \gamma)\frac{w_t(j)}{\sum_{j=1}^{N} w_t(j)} + \frac{\gamma}{N}$ for each j
  Sample the next category j' according to the probability distribution of $p_t(j)$
  Obtain the reward $g_t(j')$ and normalise it $\hat{g}_t(j') = \frac{g_t(j')}{p_t(j')}$
  Update the weight of the selected category $w_{t+1}(j') = w_t(j') \exp(\gamma \hat{g}_t(j')/N)$
  Keep all other weights $w_{t+1}(j) = w_t(j)$ for $j \neq j'$
end for

For a categorical variable h with N categories and bounded reward ($g_t \leq 1$), the regret bound for EXP3 algorithm is

$$\mathcal{R}_T^{\text{EXP3}} = \max_{h^* \in \{1,\ldots,N\}} \sum_t^T g_t(h^*) - \mathbb{E}\left[\sum_t^T g_t(h)\right] \leq 2.63\sqrt{TN\log(N)}, \tag{5.2}$$

where h^* is the best single action over all rounds and $g_t(\cdot)$ is the reward function. Please refer to (Auer et al., 2002b) for detailed derivation.

The algorithm of EXP3 is summarised in Algorithm 4. Essentially, EXP3 assigns a dynamic weight to each category which is normalised to give the probability of taking this category in the next iteration. These weights are updated with the reward obtained and a trade-off parameter $\gamma \in [0, 1]$ is introduced to encourage a uniform sampling of categories.

5.3 Related Work

One-hot Encoding

A common method for dealing with categorical variables is to transform them into a one-hot encoded representation, where a variable with N choices is transformed into a vector of length N with a single non-zero element. This is the approach followed by popular BO packages like Spearmint (Snoek et al., 2012a) and GPyOpt (González et al., 2016b; authors, 2016).

There are two main drawbacks with this approach. First, the commonly-used kernels in the GP surrogate assume that f is continuous and differentiable in the

input space, which is not the case for one-hot encoded variables, as the objective is only defined for a small subspace within this representation.

The second drawback is that one-hot encoding adds many dimensions to the search space, making the continuous optimisation of the acquisition function much harder. Additionally, the distances between one-hot encoded categorical variables tend to be large. As a result, the optimisation landscape is characterised by many flat regions, increasing the difficulty of optimisation (Rana et al., 2017).

Category-specific Continuous Inputs

We consider two recent approaches in handling category-specific continuous inputs, a specific setting of mixed categorical-continuous problems. The first approach is EXP3BO (Gopakumar et al., 2018), which can deal with mixed categorical and continuous input spaces by utilising a MAB. When the categorical variable is selected by the MAB, EXP3BO constructs a GP surrogate specific to the chosen category for modelling the continuous domain, i.e. it shares no information across the different categories. The observed data are divided into smaller subsets, one for each category, and as a result EXP3BO can handle only a small number of categorical choices and requires a large number of samples. Recently, Nguyen et al. (2019) considered extending the category-specific continuous inputs to the batch setting using Thompson sampling (Russo et al., 2018).

Hierarchical Approaches

Random forests (RFs) (Breiman, 2001) can naturally consider continuous and categorical variables, and are used in SMAC (Hutter et al., 2011a) as the underlying surrogate model for f. However, the predictive distribution of the RF, which is used to select the next evaluation, is less reliable, as it relies on randomness introduced by the bootstrap samples and the randomly chosen subset of variables to be tested at each node to split the data. Moreover, RFs can easily overfit and we need to carefully choose the number of trees. Another tree-based approach is Tree Parzen Estimator (TPE) (Bergstra et al., 2011a), an optimisation algorithm based on tree-structured Parzen density estimators. TPE uses nonparametric Parzen

kernel density estimators to model the distribution of good and bad configurations w.r.t. a reference value. Due to the nature of kernel density estimators, TPE also supports continuous and discrete spaces.

Integer-continuous Inputs

There are also other approaches for integer-continuous variables (Swiler et al., 2014; Garrido-Merchán and Hernández-Lobato, 2019; Daxberger et al., 2019; Pelamatti et al., 2020) which are related, but not essentially suitable for mixed categorical-continuous variables.

Remark. To the best of our knowledge, none of the above approaches has solved satisfactorily for the mixed type variables where (1) the continuous variables are not specific to a category and (2) we have *multiple* categorical variables each of which include *multiple* choices.

5.4 Continuous and Categorical Bayesian Optimisation

Our proposed method, Continuous and Categorical Bayesian Optimisation, harnesses both the advantages of multi-armed bandits to select categorical inputs and the strengths of GP-based BO in optimising continuous input spaces. The CoCaBO procedure is shown in Algorithm 5. CoCaBO first decides the values of the categorical inputs $\mathbf{h}_t$ by using MAB (Step 4 in Algorithm 5). Given $\mathbf{h}_t$, it then maximises the acquisition function to select the continuous part $\mathbf{x}_t$ which forms the next point $\mathbf{z}_t = [\mathbf{h}_t, \mathbf{x}_t]$ for evaluation, as illustrated in Figure 5.1.

5.4.1 CoCaBO Bandit Algorithm

Motivations

The design of the MAB system in the CoCaBO algorithm is motivated by two considerations.

Algorithm 5 CoCaBO Algorithm

Input: A black-box function f, observation data $\mathcal{D}_0$, maximum number of iterations T
Output: The best recommendation $\mathbf{z}_T = [\mathbf{x}_T, \mathbf{h}_T]$
for $t = 1, \ldots, T$ **do**
  Select $\mathbf{h}_t = [h_{1,t}, \ldots, h_{d_h,t}] \leftarrow \text{MAB}(\{\mathbf{h}_i, f_i\}_i^{t-1})$
  Select $\mathbf{x}_t = \arg\max \alpha_t(\mathbf{x}|\mathcal{D}_{t-1}, \mathbf{h}_t)$
  Query at $\mathbf{z}_t = [\mathbf{x}_t, \mathbf{h}_t]$ to obtain f_t
  $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup (\mathbf{z}_t, f_t)$ and update the reward distributions of MAB with $(\mathbf{h}_t, f_t)$
end for

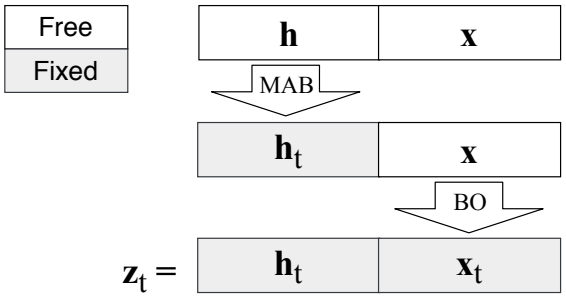


Figure 5.1: Optimisation procedure in CoCaBO.

First, our problem setting deals with *multiple* categorical inputs $\mathbf{h} = [h_1, \ldots, h_{d_h}]$ and the value of each h_j is determined by a MAB agent. This results in a multi-agent MAB system and requires coordination. We achieved the coordination across the agents by using a joint reward function. Specifically, at each iteration, each agent decides the categorical value for its own variable among the corresponding N_j choices but the objective function is evaluated based on the joint decision of all the agents. This function value is then used to update the reward distribution for the chosen category among all the agents. A concurrent work (Carlucci et al., 2020) also uses the similar coordination in deploying multi-agent learning system and shows its effectiveness on their neural architecture search application.

Second, we adopt the EXP3 (Auer et al., 2002b) method as the MAB algorithm because it makes comparatively fewer assumptions on reward distributions and can deal with the general setting of adversarially-defined rewards. The general adversarial bandit setting is important for our application as it enables the MAB part of our algorithm to handle the non-stationary nature of the rewards observed

during the optimisation (Allesiardo et al., 2017); the function value for the given values of categorical variables will improve over iterations as we apply BO on the continuous space. In contrast, Thompson Sampling (Thompson, 1933), UCB and ϵ-greedy assume that the rewards are i.i.d. samples from stationary distributions (Allesiardo et al., 2017), thus are not suitable for our application.

By using the MAB to decide the values for categorical inputs, we only need to optimise the acquisition function over the continuous subspace $\mathcal{X} \in \mathbb{R}^d$. In comparison to one-hot based methods, whose acquisition functions are defined over $\mathbb{R}^{(d+\sum_i^{d_h} N_i)}$, our approach enjoys a significant reduction in the difficulty and cost of optimising the acquisition function[1].

Regret Analyses

Assume we have c categorical variables $\mathbf{h} = [h_1, \ldots, h_{d_h}]$ and each categorical variables h_j, determined by an agent, can take one of N categories. A simplified version of the cumulative regret of such multi-agent MAB setting after T iterations can be written as:

$$\sum_j^{d_h} \mathcal{R}_T^j = \sum_j^{d_h} \sup_{\mathbf{h}_{\setminus j}} \left\{ \max_{h_j^* \in \{1,\ldots,N\}} \sum_t^T g_t(h_j^*, \mathbf{h}_{\setminus j}) - \mathbb{E}\left[\sum_t^T g_t(h_j, \mathbf{h}_{\setminus j})\right] \right\} \tag{5.3}$$

where $\mathcal{R}_T^j$ is the cumulative regret of agent j when we consider the decisions by the other agents $\mathbf{h}_{\setminus j}$ are controlled by the adversary. $g_t(\cdot)$ is the reward achieved by the joint decisions of all the agents and is equal to the normalised (between $[0, 1]$) objective function value in our case.

We first look at the cumulative regret for a single agent when the decision of all the other agents (i.e. the value of all the other categorical variables) are fixed to $\mathbf{h}_{\setminus j}$:

$$\mathcal{R}_T^j = \sup_{\mathbf{h}_{\setminus j}} \left\{ \max_{h_j^* \in \{1,\ldots,N\}} \sum_t^T g_t(h_j^*, \mathbf{h}_{\setminus j}) - \mathbb{E}\left[\sum_t^T g_t(h_j, \mathbf{h}_{\setminus j})\right] \right\} \tag{5.4}$$

[1]To optimise the acquisition function to within ζ accuracy using a grid search or branch-and-bound optimiser, our approach requires only $\mathcal{O}(\zeta^{-d})$ calls and one-hot approaches require $\mathcal{O}(\zeta^{-(d+\sum_i^c N_i)})$ calls. The cost-saving grows exponentially with the number of categories c and number of choices for each category N_i.

At iteration t with given continuous inputs, the adversary's reward function $g_t(h_j, \mathbf{h}_{\setminus j})$ (with fixed $\mathbf{h}_{\setminus j}$) is equivalent to $\hat{g}_t(h_j)$ where $g_t(\cdot)$ has N^{d_h} possible outputs while $\hat{g}_t(\cdot)$ has only N possible outputs (Carlucci et al., 2020). By substituting $\hat{g}_t(h_j) = g_t(h_j, \mathbf{h}_{\setminus j})$ in Equation (5.4), we reduce the setting for a single agent to the EXP3 setting (Auer et al., 2002b) whose regret bound is Equation (5.2):

$$\begin{aligned}\mathcal{R}_T^j &= \sup_{\mathbf{h}_{\setminus j}} \left\{ \max_{h_j^* \in \{1,\dots,N\}} \sum_t^T \hat{g}_t(h_j^*) - \mathbb{E}\left[\sum_t^T \hat{g}_t(h_j)\right] \right\} \\ &= \sup_{\mathbf{h}_{\setminus j}} \left\{ \mathcal{R}_T^{\text{EXP3},j} \right\} \leq \sup_{\mathbf{h}_{\setminus j}} \left\{ 2.63\sqrt{TN\log(N)} \right\} \end{aligned} \tag{5.5}$$

Therefore, the cumulative regret of our multi-agent EXP3 setting after T iterations has the bound:

$$\sum_j^{d_h} \mathcal{R}_T^j \leq 2.63 d_h \sqrt{TN\log(N)} \tag{5.6}$$

which is sub-linear as $\lim_{T\to\infty} \frac{\sum_j^{d_h} \mathcal{R}_T^j}{T} = 0$ and increases with the number of categorical variables d_h and the number of choices N for each categorical input.

Empirical Demonstration

In Figure 5.2, we demonstrate the effectiveness of our approach in dealing with categorical variables via a simple synthetic example Func-2C (described in Section 5.5), which comprises two categorical inputs, h_1 ($N_1 = 3$) and h_2 ($N_2 = 5$), and two continuous inputs. The optimal function value lies in the subspace when both categorical variables $h_1 = h_2 = 2$. The categories chosen by CoCaBO at each iteration, the histogram of all selections and the rewards for each category are shown for 200 iterations. We can see that CoCaBO successfully identifies and focuses on the correct categories.

5.4.2 CoCaBO Kernel Design

We propose to use a combination of two separate kernels: $k_z(\mathbf{z}, \mathbf{z}')$ will combine $k_h(\mathbf{h}, \mathbf{h}')$, a kernel defined over the categorical inputs, with $k_x(\mathbf{x}, \mathbf{x}')$, for the continuous inputs.

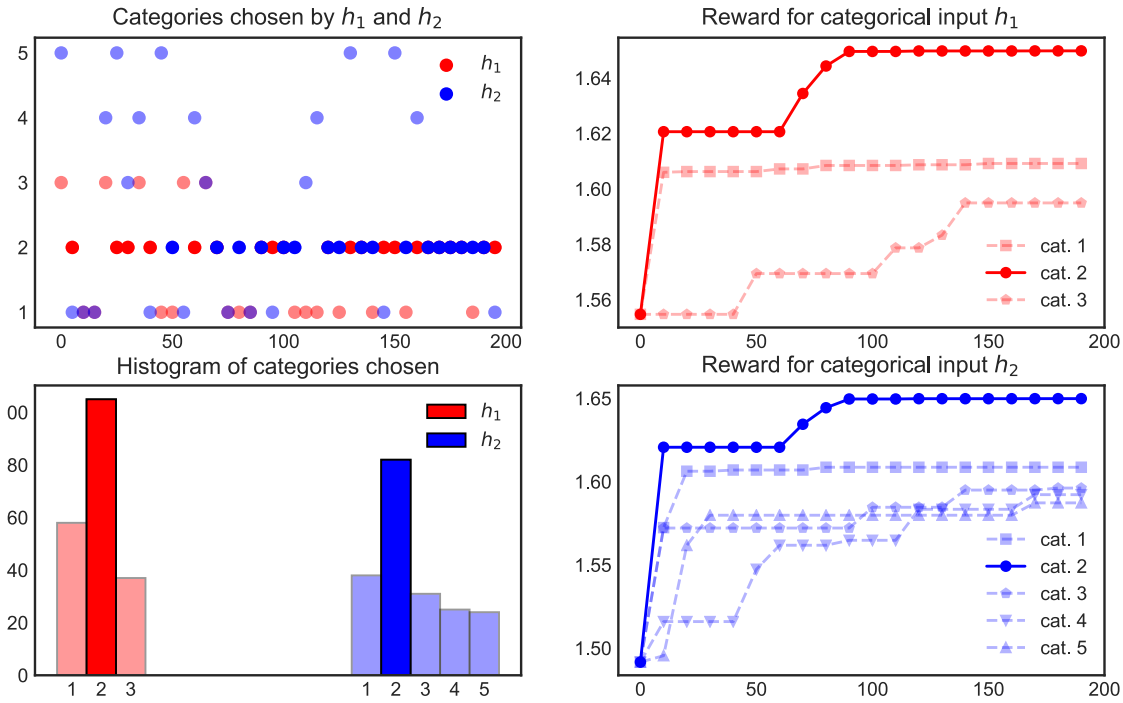


Figure 5.2: CoCaBO correctly optimises the two categorical inputs h_1 (Red) and h_2 (Blue) of the Func-2C test function over 200 iterations. The *best* category is $h_i = 2$ for both h_1 ($N_1 = 3$) and h_2 ($N_2 = 5$), and is highlighted in all plots. The left subplot shows the selections made by CoCaBO, showing how the both categorical inputs increasingly focus on the best categories as the algorithm progresses. The second left subplot shows the histogram of categories selected, with the best category being chosen the most frequently. The right subplots show the reward for each categorical value for h_1 and h_2 across iterations. Again, we see the correct category being identified for both categorical inputs for the highest rewards.

For the categorical kernel, we propose using an indicator-based similarity metric,

$$k_h(\mathbf{h}, \mathbf{h}') = \frac{\sigma}{d_h} \sum_{i=1}^{d_h} \mathbb{I}(h_i - h_i'), \tag{5.7}$$

where σ is the kernel variance and $\mathbb{I}(h_i, h_i') = 1$ if $h_i = h_i'$ and is zero otherwise. This kernel can be derived as a special case of a RBF kernel. Consider the standard RBF kernel with unit variance evaluated between two scalar locations a and a':

$$k(a, a') = \exp\left(-\frac{1}{2}\frac{(a - a')^2}{l^2}\right). \tag{5.8}$$

The lengthscale in Equation (5.8) allows us to define the similarity between the two inputs, and, as the lengthscale becomes smaller, the distance between locations that would be considered similar (i.e. high covariance) shrinks. The limiting case

$l \to 0$ states that if two inputs are not exactly the same as each other, then they provide no information for inferring the GP posterior's value at each other's locations. This causes the kernel to turn into an indicator function as in Equation (5.7) above (Kulis and Jordan, 2011):

$$k(a, a') = \begin{cases} 1, & \text{if } a = a' \\ 0, & \text{otherwise.} \end{cases} \tag{5.9}$$

By adding one such RBF kernel with $l \to 0$ for each categorical variable in h and normalising the output we arrive at the form in Equation (5.7).

There are several ways of combining kernels that result in valid kernels (Duvenaud et al., 2013). One approach is to sum them together. Using a sum of kernels, that are each defined over different subsets of an input space, has been used successfully for BO in the context of high-dimensional optimisation in the past (Kandasamy et al., 2015). Simply adding the continuous kernel to the categorical kernel $k_x(\mathbf{x}, \mathbf{x}') + k_h(\mathbf{h}, \mathbf{h}')$, though, provides limited expressiveness, as this translates in practice to learning a single common trend over $\mathbf{x}$, and an offset depending on $\mathbf{h}$.

An alternative approach is to use the product of the two kernels $k_x(\mathbf{x}, \mathbf{x}') \times k_h(\mathbf{h}, \mathbf{h}')$. This form allows the kernel to encode couplings between the continuous and categorical domains, allowing a richer set of relationships to be captured, but if there are no overlapping categories in the data, which is likely to occur in early iterations of BO, this would cause the product kernel to be zero and prevent the model from learning.

We therefore propose the CoCaBO kernel as a mixture of the sum and product kernels:

$$k_z(\mathbf{z}, \mathbf{z}') = (1 - \lambda)\Big(k_h(\mathbf{h}, \mathbf{h}') + k_x(\mathbf{x}, \mathbf{x}')\Big) + \lambda k_h(\mathbf{h}, \mathbf{h}') k_x(\mathbf{x}, \mathbf{x}'), \tag{5.10}$$

The trade-off between them is controlled by a parameter $\lambda \in [0, 1]$, which can be optimised jointly with the GP hyper-parameters (see Appendix C.2).

It is worth highlighting a key benefit of our formulation over alternative hierarchical methods discussed in Section 5.3. Rather than dividing our data

into a subset for each combination of categories, we instead use all of our acquired data at every stage of the optimisation, as our kernel is able to combine information from data within the same category as well as from different categories, which improves its modelling performance (Section 5.5.3).

Our kernel is able to learn the covariances k_z between observations in the joint input space $\mathcal{Z} = \mathcal{H} \times \mathcal{X}$. When two data points come from different categories, k_z is dominated by k_x (in the additive term). When there is some overlap in categorical choices between data points, then k_z is determined by contributions from both k_x and k_h (in both additive and multiplicative terms). In comparison, the one-hot encoded kernel converts all values of the categorical variables into additional continuous dimensions, allowing it to give nonzero covariances between points from different categories. We compare the regression performance of the CoCaBO kernel and a one-hot encoded kernel on some synthetic functions in Section 5.5.1 and show that our kernel leads to a superior predictive performance over one-hot kernel.

5.4.3 Batch CoCaBO

Algorithm 6 CoCaBO batch selection

Input: Observation data $\mathcal{D}_{t-1}$
Output: The batch $\mathcal{B}_t = \{\mathbf{z}_t^{(1)}, \ldots, \mathbf{z}_t^{(b)}\}$
$\mathbf{H}_t = \{\mathbf{h}_t^{(1)}, \ldots, \mathbf{h}_t^{(b)}\} \leftarrow$ Batch MAB$(\mathcal{D}_{t-1})$
$(\mathbf{u}_1, v_1), \ldots, (\mathbf{u}_q, v_q)$ are the unique categorical values in $\mathbf{H}_t$ and their counts
Initialise $\mathcal{B}_t = \varnothing$ and $\mathcal{D}'_{t-1} = \mathcal{D}_{t-1}$
for $j = 1, \ldots, q$ **do**
    $\{\mathbf{x}_i\}_{i=1}^{v_j} \leftarrow \text{KB}(\mathbf{u}_j, \mathcal{D}'_{t-1})$
    $\mathbf{Z}_j = \{\mathbf{u}_j, \mathbf{x}_i\}_{i=1}^{v_j}$ and $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \mathbf{Z}_j$
    $\mathcal{D}'_t \leftarrow \mathcal{D}'_{t-1} \cup \{\mathbf{Z}_j, \mu(\mathbf{Z}_j)\}$
end for
Output: $\mathcal{B}_t$

Our focus on optimising computer simulations and modelling pipelines provides a strong motivation to extend CoCaBO to select and evaluate multiple tasks at each iteration, in order to better utilise available hardware resources (Snoek et al., 2012a; Wu and Frazier, 2016b; Shah and Ghahramani, 2015a; Contal et al., 2013a).

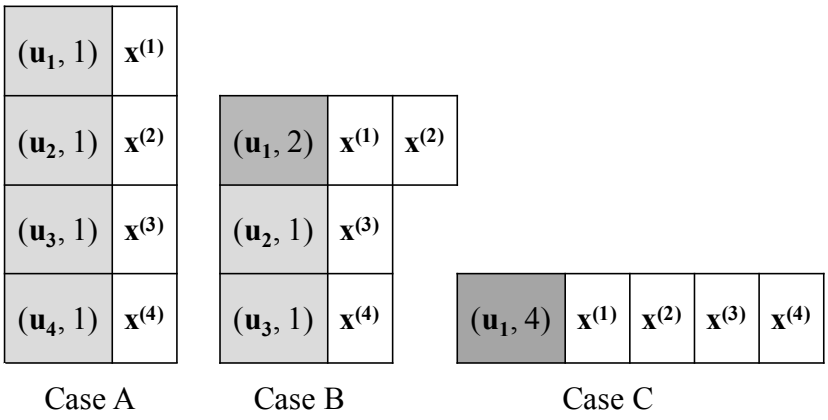


Figure 5.3: Three example cases for selecting a batch of 4 unique points ($b = 4$). $\mathbf{u}_j$ represents a unique categorical point. If all the categorical points in the batch are different, we select 1 continuous location $\mathbf{x}^{(i)}$ conditioned on each $\mathbf{u}_j$ (Case A). If 2 out of a batch of 4 categorical points are equal to $\mathbf{u_1}$, we sequentially select 2 continuous locations $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ conditioned on $\mathbf{u_1}$ using KB and collect only 1 continuous location for the other two categorical data $\mathbf{u_2}$ and $\mathbf{u_3}$ (Case B).

The batch CoCaBO algorithm uses the "multiple plays" formulation of EXP3, called EXP3.M (Auer et al., 2002b), which returns a batch of categorical choices, and combines it with the Kriging Believer (KB)[2] (Ginsbourger et al., 2010a) batch method to select the batch points in the continuous domain. We choose KB for the batch creation, as it can consider already-selected batch points, including those with different categorical values, without making restrictive assumptions as do other popular techniques, e.g. local penalisation (González et al., 2016b; Alvi et al., 2019) assumes that f is Lipschitz continuous. A detailed description of KB algorithm is in Section 2.3.1. Our novel contribution is a method for combining the batch points selected by EXP3.M with batch BO procedures for continuous input spaces. Assume we are selecting a batch of b points $\mathcal{B}_t = \{\mathbf{z}_t^{(i)}\}_{i=1}^b$ at iteration t. A simple approach is to select a batch of categorical variables $\mathbf{H}_t = \{\mathbf{h}_t^{(i)}\}_{i=1}^b$ and then choose a corresponding continuous variable for each categorical point as in the sequential algorithm above, thus forming $\{\mathbf{z}_t^{(i)}\}_{i=1}^b = \{\mathbf{h}_t^{(i)}, \mathbf{x}_t^{(i)}\}_{i=1}^b$. However, such a batch method may not identify b unique locations, as some values in $\{\mathbf{h}_t^{(i)}\}_{i=1}^b$ may be repeated and you will get identical $\mathbf{x}$ values for repeated $\mathbf{h}_t^{(i)}$ values. This is even

[2]Note that our approach can easily utilise other batch selection techniques if desired.

more problematic when the number of possible combinations for the categorical variables, $\prod_{i=1}^{d_h} N_i$, is smaller than the batch size b, as we would never identify a full batch of unique points.

Our batch selection method, outlined in Algorithm 6, allows us to create a batch of unique points $\mathcal{B}_t = \{\mathbf{z}_t^{(i)}\}_{i=1}^{b}$ (i.e. $\mathbf{z}_t^{(i)} \neq \mathbf{z}_t^{(j)}$ if $i \neq j$) by allocating multiple continuous batch points to more desirable categories. The key idea is to first collect the q unique categorical points $\{\mathbf{u}_j\}_{j=1}^{q}$ and how often they occur $\{v_j\}_{j=1}^{q}$ from the batch $\mathbf{H}_t$. The count v_j defines how many continuous batch points will be selected via KB for each unique categorical point $\mathbf{u}_j$. This process is illustrated in Figure 5.3 for three possible scenarios. The benefit of using KB here is that the algorithm can take into account selections across the different $\mathbf{h}$ to impose diversity in the batch in a consistent manner.

5.5 Experiments

We compared CoCaBO against a range of existing methods which are able to handle problems with mixed type inputs: SMAC (Hutter et al., 2011a), TPE (Bergstra et al., 2011a), GP-based Bayesian optimisation with one-hot encoding (One-hot BO) (authors, 2016) and EXP3BO (Gopakumar et al., 2018). We did not compare to the concurrent category-specific approach of (Nguyen et al., 2019) because its code is not released yet. However, the method (Nguyen et al., 2019) is highly similar to EXP3BO and suffers the same limitations as EXP3BO. For all the baseline methods, we used their publicly available Python packages[3]. CoCaBO and one-hot BO both use the UCB acquisition function (Srinivas et al., 2009) with scale parameter $\kappa = 2.0$. In all experiments, we tested four different λ values for our method[4]: $\lambda = 1.0, 0.5, 0.0, \text{auto}$, where $\lambda = \text{auto}$ means λ is optimised as a hyper-parameter. This leads to four variants of our method: CoCaBO-1.0, CoCaBO-0.5, CoCaBO-0.0 and CoCaBO-auto. We used a Matérn, $\nu = \frac{5}{2}$, kernel for k_x, as well as for One-hot

[3]One-hot BO: https://github.com/SheffieldML/GPyOpt, SMAC: https://github.com/automl/pysmac, TPE: https://github.com/hyperopt/hyperopt, EXP3BO: https://github.com/shivapratap/AlgorithmicAssurance_NIPS2018

[4]https://github.com/rubinxin/CoCaBO_code

Table 5.1: Continuous and categorical input range of the synthetic test functions

Function f	Inputs $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$	Input values
Func-2C ($d = 2$, $d_h = 2$)	h_1	$\{\text{ros}(\mathbf{x}), \text{cam}(\mathbf{x}), \text{bea}(\mathbf{x})\}$
	h_2	$\{+\text{ros}(\mathbf{x}), +\text{cam}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x})\}$
	$\mathbf{x}$	$[-1, 1]^2$
Func-3C ($d = 2$, $d_h = 3$)	h_1	$\{\text{ros}(\mathbf{x}), \text{cam}(\mathbf{x}), \text{bea}(\mathbf{x})\}$
	h_2	$\{+\text{ros}(\mathbf{x}), +\text{cam}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x})\}$
	h_3	$\{+5 \times \text{cam}(\mathbf{x}), +2 \times \text{ros}(\mathbf{x}), +2 \times \text{bea}(\mathbf{x}), +3 \times \text{bea}(\mathbf{x})\}$
	$\mathbf{x}$	$[-1, 1]^2$
Ackley-cC for $d_h = \{2, 3, 4, 5\}$ ($d = 1$, $N_i = 17$)	h_i for $i = 1, 2, \ldots, 5$	$\{z_i = -1 + 0.125 \times (j - 1), \text{ for } j = 1, 2, \ldots, 17\}$
	$\mathbf{x}$	$[-1, 1]$

BO, and used the indicator-based kernel discussed in Section 5.4.2 for k_h. For both our method and One-hot BO, we optimised the GP hyper-parameters by maximising the log marginal likelihood every 10 iterations using multi-started gradient descent.

TPE is only used in the sequential setting ($b = 1$) because its package HyperOpt does not provide a synchronous batch implementation. We started each optimisation method except EXP3BO with 24 random initial points. EXP3BO requires much more initial data to start with because as mentioned in Section 5.3, it needs to fit a GP surrogate to each category. For example, if the problem involves 2 categorical variables, each of which has 3 categorical choices, EXP3BO needs to construct 9 independent GP surrogates by dividing the observation data into 9 subsets (one for each surrogate). When applying EXP3BO for our problems, we start it with 3 random initial points for each GP surrogate. For all the problems, the continuous inputs were normalised to $\mathbf{x} \in [-1, 1]^d$ and all experiments were conducted on a 36-core 2.3GHz Intel Xeon processor with 512 GB RAM.

We tested all these methods on a diverse set of synthetic and real problems:

- Func-2C is a test problem with 2 continuous inputs ($d = 2$) and 2 categorical inputs ($d_h = 2$). The categorical inputs control a linear combination of three 2-dimensional global optimisation benchmark functions: beale, six-hump camel and rosenbrock. This is the function used for the illustration in Figure 5.2;

Table 5.2: Continuous and categorical input ranges of the real-world problems

Problems	Inputs $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$	Input values
SVM-Boston ($d = 3$, $d_h = 3$)	Kernel type h_1,	{linear, poly, RBF, sigmoid}
	Kernel coefficient h_2,	{scale, auto }
	Shrinking h_3,	{shrinking on, shrinking off}
	Penalty parameter x_1,	$[0, 10]$
	Tolerance for stopping x_2,	$10^{[10^{-6},1]}$
	Lower bound of the fraction of support vector x_3	$[0, 1]$
XG-MNIST ($d = 5$, $d_h = 3$)	Booster type h_1,	{gbtree, dart}
	Grow policies h_2,	{depthwise, loss}
	Training objective h_3,	{softmax, softprob}
	Learning rate x_1,	$[0, 1]$
	Maximum dept x_2,	$[1, 2, \ldots, 10]$
	Minimum split loss x_3,	$[0, 10]$
	Subsample x_4,	$[0.001, 1]$
	Regularisation x_5	$[0, 5]$
NAS-CIFAR10 ($d = 22$, $d_h = 5$)	Operations for the 5 intermediate nodes in the DAG $\{h_i\}_{i=1}^{5}$,	{3×3 conv, 1×1 conv, and 3×3 max-pool}
	Probability values for the 21 possible edges in the DAG $\{x_i\}_{i=1}^{2}1$,	$[0, 1]$
	Number of edges present in the DAG x_{22}	$[0, 9]$

- Func-3C is similar to *Func-2C* but with 3 categorical inputs ($d_h = 3$) which leads to more complicated combinations of the three functions;

- Ackley-cC, with $d_h = \{2, 3, 4, 5\}$ and $d = 1$ is generated to test the performance of CoCaBO on problems with large numbers of categorical inputs and inputs with large numbers of categorical choices. Here, we convert c dimensions of the $(d_h + 1)$-dimensional Ackley function into 17 categories each;

- SVM-Boston outputs the negative mean square test error of using a support vector machine (SVM) for regression on the Boston housing dataset (Dua and Graff, 2017);

- XG-MNIST returns classification test accuracy of a XGBoost model (Chen and Guestrin, 2016) on MNIST (LeCun and Cortes, 2010);
- NAS-CIFAR10 performs the architecture search on convolutional neural network topology for CIFAR10 classification. We conducted the search using the NAS-Bench-101 dataset (Ying et al., 2019) and adopted the search space proposed in Ying et al. (2019) which encodes each unique architecture with 5 categorical variables and 22 continuous variables.

The input space of each problem involves both continuous variables and multiple categorical variables. We summarise the input description of all the synthetic problems in Table 5.1 and that of real problems in Table 5.2.

5.5.1 Predictive Performance of the CoCaBO Posterior

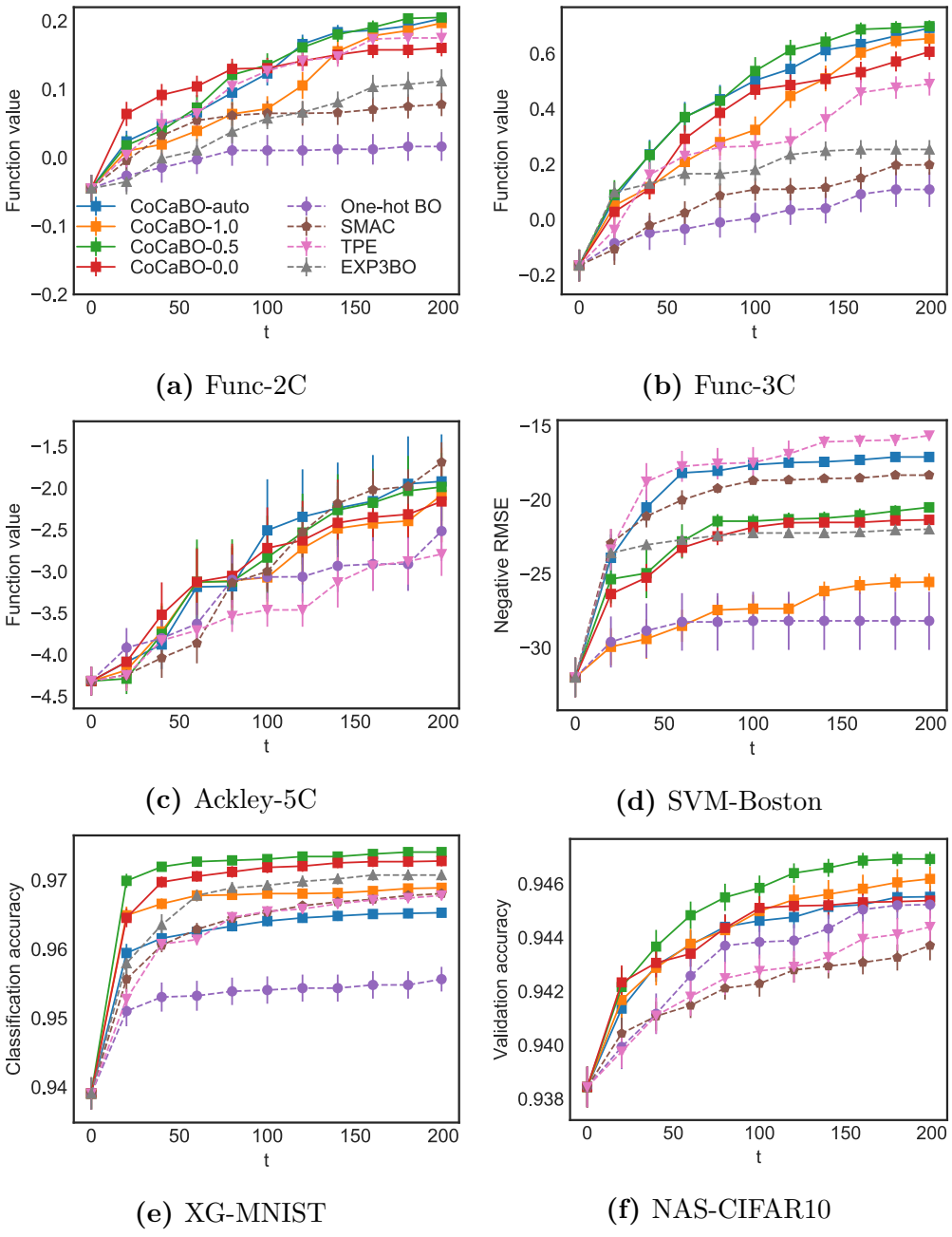
Table 5.3: Mean and standard error of the predictive log likelihood of the CoCaBO and the One-hot BO surrogates on synthetic test functions. Both models were trained on 250 samples and evaluated on 100 test points. The CoCaBO surrogate can model the function surface better than the One-hot surrogate as the number of categorical variables increases.

	CoCaBO	One-hot
Ackley-2C	$-69.1(\pm 9.47)$	$\mathbf{-60.4(\pm 5.91)}$
Ackley-3C	$\mathbf{-74.9(\pm 3.17)}$	$-107(\pm 23.7)$
Ackley-4C	$\mathbf{-92.0(\pm 6.31)}$	$-102(\pm 7.94)$
Ackley-5C	$\mathbf{-114(\pm 8.31)}$	$-120(\pm 10.3)$
Func-2C	$\mathbf{115(\pm 14.1)}$	$12.4(\pm 10.7)$
Func-3C	$\mathbf{167(\pm 6.54)}$	$-81.0(\pm 4.69)$

We first investigate the quality of the CoCaBO-auto surrogate by comparing its modelling performance against a standard GP with one-hot encoding. We train each model on 250 uniformly randomly sampled data points and evaluate the predictive log likelihood on 100 test data points. The mean and standard error over 20 random initialisations are presented in Table 5.3. The results showcase the benefit of using the CoCaBO kernel over a kernel with one-hot encoded inputs,

especially when the number of categorical inputs grows. The CoCaBO kernel, which allows it to learn a richer set of variations from the data, leads to consistently better out-of-sample predictions.

5.5.2 Performance of CoCaBO in Sequential Setting



(a) Func-2C **(b)** Func-3C

(c) Ackley-5C **(d)** SVM-Boston

(e) XG-MNIST **(f)** NAS-CIFAR10

Figure 5.4: Performance of CoCaBOs against existing methods on various tasks in the sequential setting ($b = 1$).

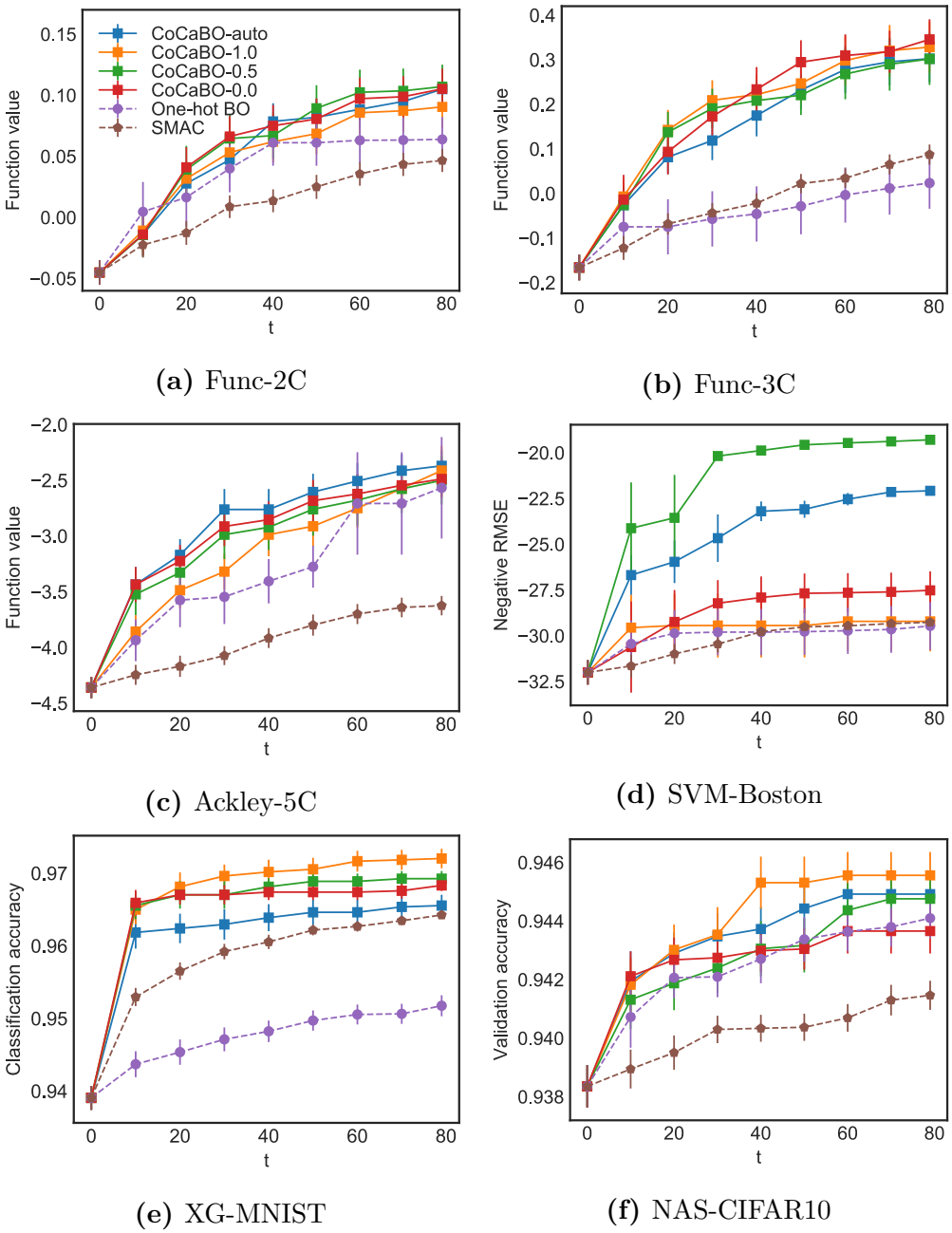
We evaluated the optimisation performance of our proposed CoCaBO methods and other existing methods in the sequential setting. We ran each sequential optimisation method for $T = 200$ iterations. We performed 20 random repetitions for synthetic problems and 10 random repetitions for the real-world problems. The mean and standard error over all repetitions are presented in Figure 5.4. For almost all synthetic and real-world problems, CoCaBO methods outperform other competing approaches with CoCaBO-0.5 and CoCaBO-auto consistently demonstrating the most competitive performance.

Note that we did not apply EXP3BO for Ackley-5C and NAS-CIFAR10 because both cases involve multiple categorical variables and each categorical variable has multiple categories, thus requiring EXP3BO to build a huge number of independent GP surrogates (17^5 and 3^5 respectively) to model the objective problem. And this in turn demands a large amount of observation data to start the optimisation. Hence, EXP3BO is not suitable for problems involving *multiple* categorical inputs with *multiple* possible values. Even in the cases where EXP3BO is compared (Figure 5.4a, 5.4b, 5.5d, 5.5e), CoCaBO outperformed EXP3BO. This justifies the gain in efficiency by using CoCaBO kernel to leverage all the data in a single surrogate in contrast with subdividing data by categories to learn multiple separate surrogates.

Another observation is that TPE performs relatively well in the case when the number of categorical combinations ($\prod_i^{d_h} N_i$) is small e.g. Func-2C(15), SVM-Boston(16) but performs clearly worse than CoCaBO when categorical combinations are large e.g. Func-3C(60), NAS-CIFAR10(243), Ackley-5C(17^5).

5.5.3 Performance of CoCaBO in Batch Setting

Now we move to experiments in the batch setting. We ran each batch optimisation with $b = 4$ for $T = 80$ iterations. The mean and standard error of the optimisation performance over 20 random repetitions are presented in Figure 5.5. CoCaBO methods again show competitive performance over other batch methods with CoCaBO-0.5 and CoCaBO-auto again remaining the top performing λ options. This supports our hypothesis that combining the sum and product kernels is



(a) Func-2C **(b)** Func-3C

(c) Ackley-5C **(d)** SVM-Boston

(e) XG-MNIST **(f)** NAS-CIFAR10

Figure 5.5: Performance of CoCaBOs against existing methods on synthetic and real-world tasks in the batch setting ($b = 4$).

beneficial for search over the mixed-type input space. Therefore, we recommend $\lambda = 0.5$ or $\lambda =$auto as the default options for using our algorithm.

One point to note is that CoCaBO-1.0 (product kernel only) performs quite competitively in the batch setting for problems like XG-MNIST and NAS-CIFAR10 whose categorical variables have few category choices. At the same iteration number,

the batch setting provides more sample data for the BO algorithm than the sequential setting, leading to more frequent overlapping categories in the data, especially when the category choices are small. Hence, the product kernel, which captures the couplings between the continuous and categorical data, can perform more effectively.

5.6 Extension Work

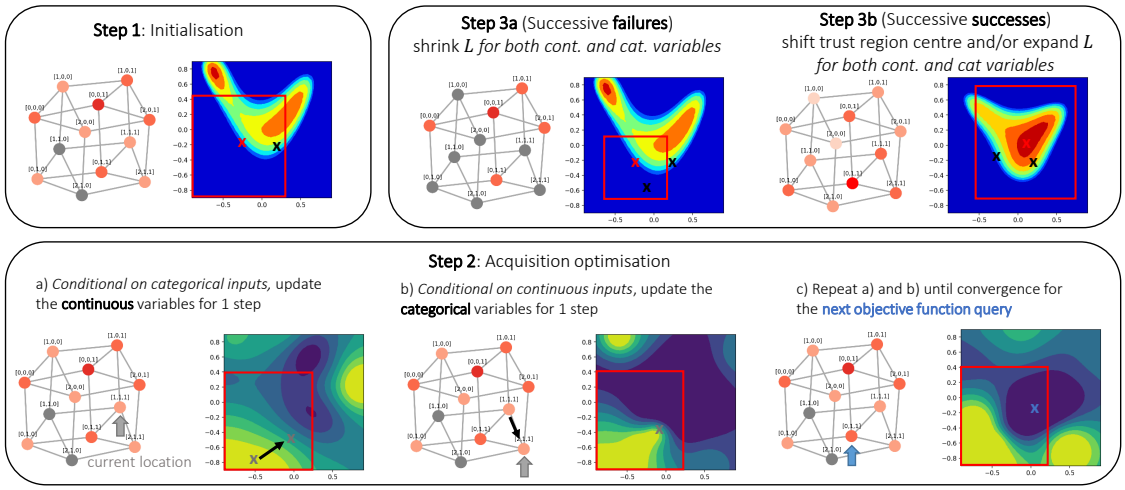


Figure 5.6: Illustration of CASMOPOLITAN in mixed space. Note that in **Steps 1** & **3** we show the *GP posterior* on $\mathcal{X}$ conditioned on the incumbent $\mathbf{h}_\mathbf{T}^*$, and in **Step 2** we show the *acquisition function* on $\mathcal{X}$ conditioned on $\mathbf{h}$ at various optimisation steps. Suppose we optimise over a 5-dimensional mixed problem with 3 categorical dimensions with $\{3, 2, 2\}$ choices for each dimension respectively, and 2 additional continuous dimensions. Initially (**Step 1**), the best location so far $\mathbf{z}_t^* = \arg\max_\mathbf{z}\{y_j\}_{z=1}^t = [\mathbf{h}_\mathbf{t}^*, \mathbf{x}_\mathbf{t}^*]$ (with the continuous TR and $\mathbf{x}_T^*$ in red box and cross). In optimisation of acquisition function (**Step 2**), we interleave the local search on $\mathcal{H}$ described in Section 5.6.3 with gradient-based optimisation on $\mathcal{X}$ until convergence. In **Steps 3a/3b**, we adjust both the continuous and categorical TRs correspondingly and restart if/when either shrinks below its minimum length.

In a more recent work (Wan et al., 2021b), we extend the CoCaBO to the more challenging setting of high-dimensional problems involving a mixture of categorical and continuous inputs. However, instead of using MABs for categorical inputs, which is less scalable to high dimensions due to the need for pulling each arm at least once, we resort back to GP-based BO with the kernel design in CoCaBO and leverage the concept of local trust region (TR) (Eriksson et al., 2019) to effectively handle high dimensions. We name the new method CASMOPOLITAN (CAtegorical Spaces, or Mixed, OPtimisatiOn with Local-trust-regIons & TAilored Non-parametric). We

Algorithm 7 CASMOPOLITAN

Input: #init (the number of random initialing points at initialisation or restarts), #iter T, initial TR size for categorical $L_0^h \in \mathbb{Z}^+$, and continuous variables $L_0^x \in \mathbb{R}^+$
Output: The best recommendation $\mathbf{z}_T = [\mathbf{x}_T, \mathbf{h}_T]$
Set restart to True initially, `restart = True`
for $t = 1, \ldots, T$ **do**
  if `restart` **then**
    Reset TR $L^h = L_0^h$ and $L^x = L_0^x$ and reset GP. Randomly select #init points in the search space as $\mathbf{z}_t$ *(if at initialisation)*,
    or set the TR center as the point determined by Equation (5.13) and randomly select #init points within the newly constructed TR as $\mathbf{z}_t$ *(if at subsequent restarts)*
  else
    Construct a TR $\mathrm{TR_h}(\mathbf{h_t^*})$ around the categorical dimensions of the best point $\mathbf{h}_t^*$ using Equation (5.12)
    Construct a hyper-rectangular TR of length L^x, $\mathrm{TR_x}(\mathbf{x_t^*})$ for the continuous variables
    Select next query point(s) *within* TRs
    $\mathbf{z}_t = \mathrm{argmax}_{\mathbf{z}} \alpha(\mathbf{z})$ s.t. $\mathbf{x} \in \mathrm{TR}_x(\mathbf{x_t^*}), \mathbf{h} \in \mathrm{TR}_h(\mathbf{h_t^*})$
  end if
  Query at $\mathbf{z}_t$ to obtain y_t
  $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup (\mathbf{z}_t, y_t)$ and update the GP surrogate
  Update the TRs and decide whether to `restart`.
end for

present the overall algorithm of CASMOPOLITAN in Algorithm 7 and include an illustration in Figure 5.6. We highlight the key design features of CASMOPOLITAN in the following subsections and recommend the readers to refer to (Wan et al., 2021b) for more details and results.

5.6.1 Modified Kernel Design

To handle categorical variables, we modify the categorical kernel in Equation (5.7):

$$k_h(\mathbf{h}, \mathbf{h}') = \exp\left(\frac{1}{d_h}\sum_{i=1}^{d_h} l_i \delta(h_i, h_i')\right), \tag{5.11}$$

where $\delta(\cdot,\cdot)$ is the Kronecker delta function and $\{l_i\}_i^{d_h}$ are the lengthscales, which will be different for each categorical dimension if we enable automatic relevance determination as we described in Section 2.1.1. This modification affords additional expressiveness in modelling more complicated functions: for example, the kernel in

Equation (5.11) can discern the dimensions to which the objective function value is more sensitive via learning different lengthscales but the original categorical kernel (Equation (5.7)) treats all dimensions equally. We empirically validate the performance gain of the modified kernel in Figure 5.9, and we prove this kernel is positive semi-definite in Lemma 1 in Appendix C.3. When modelling over high-dimensional spaces with both categorical and continuous inputs, we again adopt the CoCaBO kernel in Equation (5.10).

5.6.2 Trust Region

As mentioned in Section 2.3.2, one key difficulty in applying GP-based BO in high-dimensional problems is that the surrogate attempts to model the entire function landscape and tend to over-explore due to the existence of large regions of high posterior variance. To circumvent this, TurBO (Eriksson et al., 2019) constrains BO search on local TRs centred around the best observed inputs so far and resizes these TRs to focus on promising local regions.

The TR definition in TurBO (Eriksson et al., 2019) is based on the Euclidean distance which works for continuous space only. Here we adapt it to categorical search space (Line 8 in Algorithm 7) by defining TRs in terms of *Hamming distance*, i.e. a TR of radius L^h from the best location, $\mathbf{h}^*$, observed at iteration t includes all points that are up to L^h variables different from $\mathbf{h}^*$:

$$\mathrm{TR}_h(\mathbf{h}^*)_{L^h} = \left\{ \mathbf{h} \mid \sum_{i=1}^{d_h} \delta(h_i, h_i^*) \leq L^h \right\}. \tag{5.12}$$

The TR radius is adjusted dynamically during optimisation, expanding on successive successes (if best function value f_t^* improves) and shrinking otherwise. Since Hamming distance is integer-valued bounded in $[0, d_h]$, we set the minimum and maximum TR radius to $L^h_{\min} = 0$ and $L^h_{\max} = d_h$ respectively.

TRs in local optimisation are typically biased toward the starting points. Therefore, most local optimisation approaches rely on a restarting strategy to attain good performance (Shylo et al., 2011; Kim and Fessler, 2018). In our case, we restart the optimisation when the TR length L^h reaches the smallest

possible value (Line 13 in Algorithm 7). However, rather than restarting randomly as in Eriksson et al. (2019), we propose to restart our method using GP-UCB principle (Srinivas et al., 2009).

Specifically, we introduce an auxiliary global GP model to achieve this. Suppose we are restarting the i-th time, we first fit the global GP model on a subset of data $\mathcal{D}^*_{i-1} = \left\{\mathbf{h}^*_j, y^*_j\right\}_{j=1}^{i-1}$, where $\mathbf{h}^*_j$ is the local maximiser found after the j-th restart or a random input point, if the found local maximiser after the j-th restart is same as one of previous restarts. The posterior of the global GP at a new categorical input $\mathbf{h}$ is $p(y|\mathcal{D}^*_{i-1}) = \mathcal{N}\left(y; \mu_{global}(\mathbf{h}), \sigma_{global}(\mathbf{h})^2\right)$. Thus, at the i-th restart, we select the following location $\mathbf{h}^{(0)}_i$ as the initial centre of the new TR:

$$\mathbf{h}^{(0)}_i = \arg\max_{\mathbf{h}\in\mathcal{H}} \mu_{global}(\mathbf{h}) + \sqrt{\beta_i}\sigma_{global}(\mathbf{h}), \tag{5.13}$$

where β_i is the trade-off parameter. As formally shown in Appendix C.3, this strategy is optimal in deciding the next TR by balancing exploration against exploitation (Srinivas et al., 2009) and leads to theoretical guarantee for CASMOPOLITAN. When applied to the categorical-continuous input space, we adopt the GP-UCB criteria in Equation (5.13) to choose the next restart location for continuous inputs as well and will make a restart when the length of either categorical TR TR_h or continuous TR TR_x reaches the smallest possible value.

5.6.3 Acquisition Function Optimisation

For the categorical space, we cannot optimise the acquisition function via gradient-based methods because we preserve the discrete nature of the variables in our method like CoCaBO does. Instead, we use the simple strategy of local search within the TRs defined above: at each BO iteration, we randomly sample an initial configuration $\mathbf{h}' \in \text{TR}_h(\mathbf{h}^*)$. We then randomly select a neighbour point of Hamming distance 1 from $\mathbf{h}'$, evaluate its acquisition function $\alpha(\cdot)$, and move from $\mathbf{h}'$ if the neighbour has a higher acquisition function value and is still within the TR. We repeat this process until a pre-set budget of queries is exhausted and dispatch the best configurations for objective function evaluation.

When optimising the acquisition function over the mixed input space (Line 10 of Algorithm 7), at each optimisation step, we simply do one step of local search on the categorical variables, followed by one step of gradient-based optimisation of the acquisition function on the continuous variables. We repeat this procedure until convergence or when a maximum number of steps is reached.

5.6.4 Experiments

We first evaluate our proposed method, CASMOPOLITAN, on several high-dimensional problems involving categorical variables only to demonstrate the effectiveness of our modified categorical kernel and adapted TR approach for categorical space. We then compare CASMOPOLITAN on problems involving a mix of both categorical and continuous input variables as the setting for CoCaBO. For fair comparison with other baselines, we use sequential version of CASMOPOLITAN and CoCaBO in the following experiments.

Categorical Problems

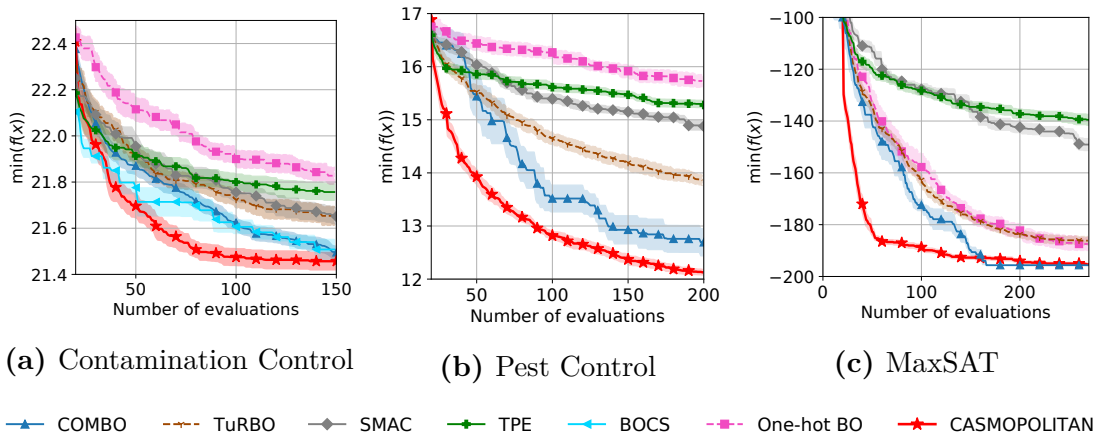


Figure 5.7: Results on various categorical optimisation problems. Lines and shaded area denote mean ± 1 standard error.

We first compare our proposed method against a number of competitive baselines, including TPE (Bergstra et al., 2011a), SMAC (Hutter et al., 2011a), BOCS

(Baptista and Poloczek, 2018) [5] and COMBO (Oh et al., 2019) which claims the state-of-the-art performance on categorical space. We also include two additional baselines: One-hot BO, which performs the naïve GP-based BO approach after converting the categorical variables into one-hot representations, and TuRBO, which is identical to One-hot BO except that we additionally incorporate the TR approach in Eriksson et al. (2019). We experiment on following real-life problems:

- Contamination Control ($d_h = 25$): Contamination Control is a *binary* optimisation problem in food supply chain (Hu et al., 2010; Oh et al., 2019) with 25 stages: at each stage, we have the choice of whether to introduce contamination control, but early use of contamination control could inevitably lead to increase in cost and as such our objective is to minimise food contamination with the smallest monetary cost (hence a minimisation problem). It is worth noting that in this problem and the Pest Control problem described below, the actions taken by the previous stage have implications on the following stages, thus leading to highly complicated interactions amongst the different variables.

- Pest Control ($d_h = 25$): Oh et al. (2019) expands the contamination control problem into a multi-categorical optimisation problem: at each stage, we now need not only to determine whether to take an action (to use pesticide or not), but also the type of the pesticide (4 choices in total). This thus gives rise to 5 *potential choices* for each stage. Similar to contamination control, we again set the total number of stages to 25.

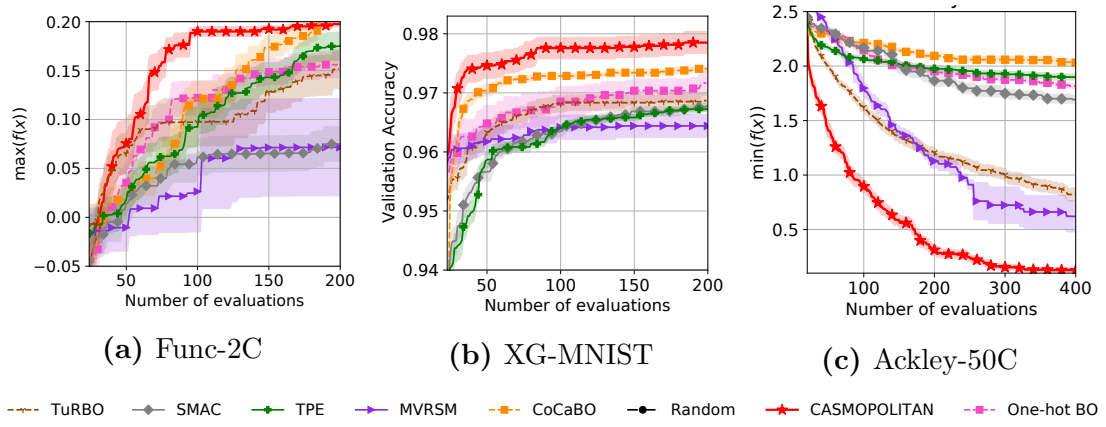
- MaxSAT ($d_h = 60$): Maximum satisfiability problem is a classical combinatorial optimisation problem that aims to determine the maximum number of clauses of a given Boolean formula in conjunctive normal form that can be made true by an assignment of truth values to the variables. Similar to Oh et al. (2019), we take the same 60-variable *binary* benchmark from Maximum

[5] BOCS is only run in Contamination, as it by default does not support multi-categorical optimisation and on MaxSAT, a single trial takes more than 100 hours, rendering comparison infeasible within our computing constraints.

Satisfiability Competition 2018 problem from https://maxsat-evaluations.github.io/2018/benchmarks.html)

The results in Figure 5.7 show that our method achieves the best convergence speed and query efficiency in general, and in terms of the performance at termination, our method again outperforms the rest except in Contamination and MaxSAT where it performs on par with COMBO. However, it is worth noting that in terms of wall-clock speed, our method is $2-3$ times faster than COMBO in the problems considered.

Mixed-input Problems



(a) Func-2C **(b)** XG-MNIST **(c)** Ackley-50C

Figure 5.8: Results on various mixed optimisation problems. Lines and shaded area denote mean $\pm$ 1 standard error.

We then consider the optimisation problems involving a mix of continuous and categorical input variables. In these experiments, in addition to SMAC, TPE, One-hot BO and TurBO compared above, we also include MVRSM Bliek et al. (2020), a recent advancements in this setup, and our CoCaBO. Note that we do not compare against BOCS and COMBO here since they are suitable for purely categorical spaces only. Under this setup, we consider three synthetic and real-life problems described in Table 5.1 and 5.2: Func-2C ($d = 2, d_h = 2$), XG-MNIST ($d = 5, d_h = 3$), Ackley-50C ($d = 3, d_h = 50$).

The results in Figure 5.8 again show that CASMOPOLITAN achieves the best performance. It is interesting to observe that in lower dimensions (Func-2C and

XG-MNIST), CoCaBO featuring tailored categorical kernels clearly outperform MVRSM and TuRBO, both focusing on high dimensions. However, in high-dimensional problems (Ackley-50C), the relative performance switches completely, suggesting that the focus on dimensionality now outweighs the importance of better treatment on different input types. Nonetheless, with *both* tailored kernels and focus on scaling to high dimensions, CASMOPOLITAN consistently outperform other baselines in all problems.

Ablation Studies

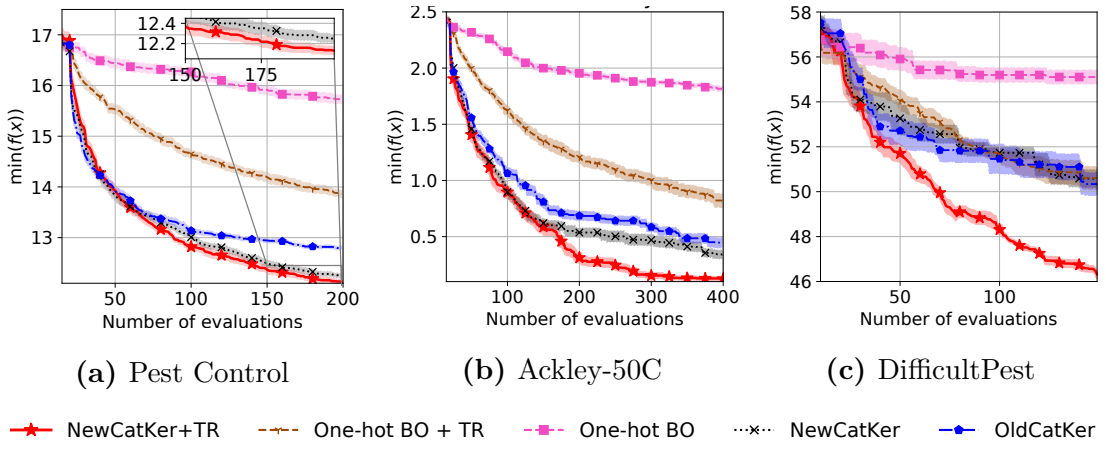


Figure 5.9: Ablation studies of our method on categorical and mixed optimisation problems.

Our method introduces a number of modifications over the naïve GP approach. To understand the benefits of these, we conduct ablation studies in both the categorical and mixed problems. Specifically, we include the following setups.

- The naïve BO approach with global GP surrogate and one-hot transformation on the categorical variables (One-hot BO);
- One-hot transformed BO, but with *local* TRs i.e. TurBO (One-hot BO+TR);
- BO with *global* surrogates, but with the original categorical kernel in Equation (5.7) where applicable (OldCatKer);
- BO with *global* GP surrogate, but with the kernel defined in Equation (5.11) (NewCatKer);

- Our approach that incorporates both local modelling and the kernel in Equation (5.11) (NewCatKer+TR).

We use Pest Control and the Ackley-50C problems as representative problems for the categorical and mixed setups for the ablation studies. To further understand the relative importance of the various features of CASMOPOLITAN especially as the dimensionality of the problems gets really large, we also include Pest control with number of stages expanded to 80, which we denote as DifficultPest (the number of possible configurations is more than 8.27×10^{55}).

We show the results in Figure 5.9: in most problems, the usage of the categorical kernel leads to improvements over baselines, with modified categorical kernel (Equation (5.11)) generally outperforming the original categorical kernel (Equation (5.7)). Unsurprisingly, the additional benefits of local optimisation and the use of trust regions increase with increasing dimensionality and complexity of the problems, with largest benefits coming from the higher-dimensional problem, DifficultPest.

5.7 Conclusion

Existing BO literatures use one-hot transformations or hierarchical approaches to encode real-world problems involving mixed continuous and categorical inputs. We presented a solution from a novel perspective, called Continuous and Categorical Bayesian Optimisation (CoCaBO), that harnesses the strengths of MABs and GP-based BO to tackle this problem. Our method uses a new kernel structure, which allows us to capture information within categories as well as across different categories. This leads to more efficient use of the acquired data and improved modelling power. We extended CoCaBO to the batch setting, enabling parallel evaluations at each stage of the optimisation. Furthermore, building on the kernel design in CoCaBO and the idea of local trust region, we develop a unified GP-based BO solution, CASMOPOLITAN, to the more challenging setting of high-dimensional problems with mixed input types. Both CoCaBO and CASMOPOLITAN demonstrate strong performance over existing methods on a variety of synthetic and

real-world optimisation tasks involving *multiple* continuous and categorical inputs. We believe they will serve as very competitive alternatives to existing approaches for practical optimisation tasks within and beyond AutoML domain.

Future directions In CoCaBO, we adopts a classic adversarial bandit algorithm, EXP3, and achieves implicit coordination in the multi-agent system via the use of a joint reward function. There exist many alternatives to these design choices, and a better bandit algorithm or multi-agent coordination strategy can help improve the performance of CoCaBO. Another interesting direction is to explore other categorical kernel choices. The categorical kernels used in CoCaBO and CASMOPOLITAN assume independence among different categories of a categorical variable. Alternative options, like the one proposed in Zhang et al. (2020) which maps categories to a low-dimensional latent space and assumes covariance in the latent space, might give better modelling performance in specific applications. In addition, the problem setting studied in this chapter does not assume conditioning among the different variables but in real practice, that might be the case. For example, the choice of continuous kernel hyper-parameters in a SVM might depend on the categorical kernel choice. The adaption of conditional GP kernels into our CoCaBO or CASMOPOLITAN frameworks can be investigated to deal with such conditional search space.

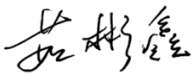
Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Bayesian optimisation over multiple continuous and categorical inputs
Publication Status	☒Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Binxin Ru*, Ahsan Alvi*, Vu Nguyen, Michael A. Osborne, and Stephen Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning (ICML 2020)*, 2020.

Student Confirmation

Student Name:	Binxin Ru		
Contribution to the Paper	Alvi and I contributed equally to the research idea discussions, experiments, code reviews and result analyses as well as the write-up of the paper. Specifically, I worked on the multi-armed bandits part of the algorithm and its regret analysis, and Alvi worked on the continuous BO part. We jointly designed the CoCaBO kernel and the batch algorithm. All the experiments are performed jointly by both of us.		
Signature		Date	17/09/2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof. Michael A. Osborne		
Supervisor comments To the best of my knowledge, all above seems fair and correct.		
Signature 	Date	17/09/2021

This completed form should be included in the thesis, at the end of the relevant chapter.

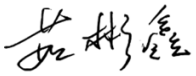
Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces
Publication Status	☒Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A. Osborne. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. In *International Conference on Machine Learning (ICML 2021)*, 2021.

Student Confirmation

Student Name:	Binxin Ru		
Contribution to the Paper	I contributed to the discussion on how to extend the categorical kernel and combined kernel in CoCaBO for the proposed method. I also contributed to running the baselines for experiments on problems with mixed inputs as well as to setting up the black-box adversarial attack experiment. I helped with the write-up of the paper.		
Signature 		Date	17/09/2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof. Michael A. Osborne		
Supervisor comments To the best of my knowledge, all above seems fair and correct.		
Signature 	Date	17/09/2021

This completed form should be included in the thesis, at the end of the relevant chapter.

6

Bayesian Optimisation for Adversarial Attack

The content of this chapter is based on the following paper:

> Binxin Ru, Adam Cobb, Arno Blaas, and Yarin Gal. BayesOpt adversarial attack. In *International Conference on Learning Representations (ICLR 2020)*. 2020.
>
> Xingchen Wan, Henry Kenlay, Binxin Ru, Arno Blaas, Michael A. Osborne, and Xiaowen Dong. Attacking graph classification via Bayesian optimisation. In *International Conference on Machine Learning Workshop on Adversarial Machine Learning*, 2021.

where our contributions were listed in the acknowledgements section of this thesis.

The extensive adoption of deep learning models in important real applications, fuelled by the development of AutoML, has led to escalating costs and risks associated with potential model failure. How to efficiently assess the adversarial robustness of such models becomes an urgent question. This is especially important for models suggested by the AutoML pipeline because they are unique for each task and we often have little prior knowledge on their design logic or potential failure modes.

One general way to assess model safety/robustness is to simulate model-agnostic adversarial attacks under realistic scenarios, such as the adversary having no access to the victim model (black-box setting) and the adversarial examples being visually indistinguishable from the original input.

However, current black-box attacks, which reply on substitute model training, gradient estimation or genetic algorithms, often require an excessive number of queries. Therefore, they are not suitable for real-world systems where the maximum query number is limited due to cost. We propose a query-efficient black-box attack, BayesOpt attack, which uses Bayesian optimisation (BO) in combination with Bayesian model selection to optimise over the adversarial perturbation and the optimal degree of search space dimension reduction. We demonstrate empirically that our method [1] can achieve comparable success rates with $2-5$ times fewer queries compared to previous state-of-the-art black-box attacks. In a recent extension work, we further adapt our BO-based attack framework to attacking graph-based machine learning models, which again achieves the state-of-the-art query efficiency under the black-box attack setting.

6.1 Introduction

Deep learning algorithms have been widely deployed in many real-world systems, ranging from identity verification (Liu et al., 2018c), to financial services (Heaton et al., 2017) to autonomous driving (Bojarski et al., 2016). The advancements in AutoML techniques free the human involvement in the painful processes of hyper-parameter tuning and architecture design/selection, thus significantly lowering the expertise required for applying such models. This is going to encourage the use of deep learning algorithms in more important tasks. However, despite their impressive performance on the designated tasks such as image classification, even the most accurate deep learning models can be easily deceived by perturbations which are visually imperceptible to the human eye (Szegedy et al., 2013; Carlini et al., 2016). The growing costs and risks associated with the potential model

[1] Our code is available at https://github.com/rubinxin/BayesOpt_Attack.git

failures has led to the importance of studying adversarial attacks, both in assessing their robustness and their ability to detect such attacks. In this chapter we focus on highly practical adversarial attacks that fulfil the following two criteria. First, the attack is designed for a black-box setting because in real-world examples, the attacker would normally have no knowledge of the target deep learning model and can only interact with the model by querying it. Second, query efficiency is highly prioritised because in practical cases where the damage caused by the attack is high, the query budget available to the attacker will be highly limited due to the risk of being detected by the defence system or other high inherent costs (monetary or computational) of model evaluations.

Despite of the large array of adversarial attacks proposed in the literature, many of them are white-box approaches that assume full access to the victim model architecture and the ability of performing back-propagation to get gradient information (Moosavi-Dezfooli et al., 2016; Kurakin et al., 2016; Gu and Rigazio, 2014; Goodfellow et al., 2014; Chen et al., 2018a; Carlini and Wagner, 2017). On the other hand, for black-box attacks, there are various techniques that have been used which do not require access to the model architecture. One class of methods trains a white-box substitute model and attacks the victim model with adversarial examples that successfully fool the substitute (Papernot et al., 2017). However, this type of method requires the availability of the original training data or large query data to train the substitute network and the performance is often limited by the mismatch between the substitute and the victim models (Su et al., 2018). The second class of black-box attacks, which show better empirical performance than the substitute model approaches, numerically estimate the gradient of the victim model by repeatedly querying it (Chen et al., 2017; Ilyas et al., 2018; Tu et al., 2018) and attack with the estimated gradient. Although various techniques are employed to increase the query efficiency for the gradient estimation, they need an excessively large query budget to achieve a successful attack (Alzantot et al., 2018). Another line of work removes the need for gradient estimation and uses decision-based techniques (Brendel et al., 2017) or genetic algorithms (Alzantot et al., 2018) to

generate adversarial examples. One popular technique that is adopted by many black-box attacks (Chen et al., 2017; Alzantot et al., 2018; Tu et al., 2018) to significantly improve the query efficiency is to search for adversarial perturbations in a low-dimensional latent space (search dimensionality reduction). However, learning the effective dimensionality of the latent search space can be challenging by itself, and has not been investigated by the prior works to the best of our knowledge.

In light of the above limitations, we propose a query efficient black-box attack that iteratively optimises over both the adversarial perturbation and the effective dimensionality of the latent search space. Our main contributions are summarised as follows:

- We introduce a novel gradient-free black-box attack method, BayesOpt attack, which uses BO with Gaussian process surrogate models to find the effective adversarial example and is capable of dealing with high-dimensional image inputs.

- We propose a Bayesian technique which learns the optimal degree of search dimensionality reduction by harnessing our statistical surrogate and information from query data. This technique can be incorporated naturally into our attack procedure, leading to efficient optimisation over both adversarial perturbation and the latent search space dimension.

- We empirically demonstrate that under the L_∞ constraint, our proposed attack method can achieve comparable success rate with about *2 to 5 times* fewer model queries in comparison to current state-of-the-art query-efficient black box attack methods.

- We extend our BayesOpt attack framework to graph classification tasks and develop the first query-efficient BO-based black box attack for graph-based machine learning models.

6.2 Related Work

Most of the existing adversarial attacks (Moosavi-Dezfooli et al., 2016; Kurakin et al., 2016; Gu and Rigazio, 2014; Goodfellow et al., 2014; Chen et al., 2018a; Carlini and Wagner, 2017) focus on white-box settings, where the attacker can get full access to the victim model and has complete knowledge of the architecture, weights and gradients to generate successful adversarial examples. However, real-world systems are usually attacked in a black-box setting, where one has no knowledge about the model and can only observe the input-output correspondences by querying the model. Here we give a brief overview of various existing black-box attacks.

Substitute model One class of black-box attacks uses data acquired from querying the target black-box model to train a substitute model (Papernot et al., 2017), which mimics the classification behaviour of the victim model. The adversary can then employ any white-box method to attack the fully observable substitute model and apply the successful adversarial example to the victim model. Such approaches rely on the transferability assumption that adversarial examples which are effective to the substitute model are also very likely to conceive the victim model given their similar classification performance on the same data (Szegedy et al., 2013). Moreover, to provide training data for the substitute model, the adversary either requires information on the victim model's training set, which is highly unrealistic in real-world applications, or needs to build a synthetic training set by querying the victim model (Papernot et al., 2017), which implies a large number of model evaluations and becomes hardly feasible for large models or complex datasets (Brendel et al., 2017).

Gradient estimation An alternative to training a substitute model is to estimate the gradient via finite differences and use the estimated gradient information to produce attacks. However, the naive coordinate-wise gradient estimation requires excessive queries to the victim model (2 queries per coordinate per descent/attack step) and thus is not feasible for attacking models with high dimensional inputs

(e.g. classifiers on ImageNet). Chen et al. (2017) overcome this limitation by using stochastic coordinate gradient descent and selecting the batch of coordinates via importance sampling, introducing the state-of-the-art zeroth-order attack, ZOO, which achieves comparable attack success rate and perturbation costs as many white-box attacks. Although ZOO makes it computationally tractable to perform black-box attack on high-dimensional image data, it still requires millions of queries to generate a successful adversarial example, making it impracticable for attacking real-world systems where model query can be expensive and the budget limited. Improving on ZOO, AutoZOOM (Tu et al., 2018) significantly enhances the query efficiency by using random vectors to estimate the full gradient and adjusting the estimation parameter adaptively to trade off query efficiency vs. input perturbation cost. More importantly, AutoZOOM shows the benefits of employing dimension reduction techniques in accelerating attacks (i.e. searching for adversarial perturbation in a low-dimensional latent space and decoding it back to the high-dimensional input space). Parallel work by Ilyas et al. (2018) estimate the gradient through a modified version of natural evolution strategy which can be viewed as a finite-difference method over a random Gaussian basis. The estimated gradient is then used with projected gradient descent, a white-box attack method, to generate adversarial examples.

Gradient-free optimisation As discussed, gradient-estimation approaches in general need an excessive number of queries to achieve successful attack. Moreover, their dependence on the gradient information makes them less robust to defences which manipulate the gradients (Athalye et al., 2018; Brendel et al., 2017; Guo et al., 2017). Thus, truly gradient-free methods are more likely to bypass such defences. One recent example, which has demonstrated state-of-the-art query efficiency, is GenAttack (Alzantot et al., 2018). GenAttack uses genetic algorithms to iteratively evolve a population of candidate adversarial examples. Besides having an annealing scheme for mutation rate and range, GenAttack also adopts dimensional reduction techniques, similar to AutoZOOM, to improve the query efficiency. In parallel to GenAttack, Brendel et al. (2017) introduce a decision-based attack,

Boundary Attack, which only requires access to the final model decision. Boundary Attack starts from a huge adversarial perturbation and then iteratively reduces the perturbation through a random walk along the decision boundary. However, Boundary Attack takes about 72 times more queries than GenAttack to fool an undefended ImageNet model (Alzantot et al., 2018). Another recent approach introduced in Moon et al. (2019) reduces the search space to a discrete domain and subsequently uses combinatorial optimisation to find successful attacks, mostly focusing on the less challenging setting of untargeted attacks. Finally, the prior works that use BO for adversarial attacks (Suya et al., 2017; Zhao et al., 2019) only investigate the use of simple Gaussian process as the surrogate model and are limited in their performance. The method proposed in Suya et al. (2017) deals with the untargetted attack setting and only demonstrates the effectiveness of BO in comparison to random search on a low-dimensional ($d = 57$) email attack task. BO-ADMM proposed in Zhao et al. (2019) works on image data but it applies BO directly on the search space of image dimension to minimise the joint objective of attack loss and distortion loss. Despite its query efficiency, BO-ADMM leads to poor-quality adversarial examples of large distortion loss.

6.3 Preliminaries

6.3.1 Problem Definition

We focus on the black-box attack setting, where the adversary has no knowledge about the network architecture, weights, gradient or training data of the victim model f, and can only query the victim model with an input $\mathbf{x}$ to observe its prediction scores on all C classes (i.e. $f : \mathbb{R}^d \longrightarrow [0,1]^C$) (Tu et al., 2018; Alzantot et al., 2018). Moreover, we aim to perform *targeted* attacks, which is more challenging than untargeted attacks, subject to a constraint on the maximum change to any of the coordinates (i.e., a L_∞ constraint) (Warde-Farley and Goodfellow, 2016; Alzantot et al., 2018). Specifically, *targeted* attacks refer to the case where given a valid input $\mathbf{x}_{origin}$ of class t_{origin} (i.e. $\arg\max_{i\in\{1,\dots,C\}} f(\mathbf{x}_{origin})_i = t_{origin}$) and a target $t \neq t_{origin}$, we aim to find an adversarial input $\mathbf{x}$, which is close to $\mathbf{x}_{origin}$ according

to the $L_\infty-$norm, such that $\arg\max_{i\in\{1,\dots,C\}} f(\mathbf{x})_i = t$. *Untargeted* adversarial attacks refer to the case that instead of classifying $\mathbf{x}_{origin}$ as t_{origin}, we try to find an input x so that $\arg\max_{i\in\{1,\dots,C\}} f(\mathbf{x})_i \neq t_{origin}$.

In our approach, we follow the convention to optimise over the perturbation $\boldsymbol{\delta}$ instead of the adversarial example $\mathbf{x}$ directly (Chen et al., 2017; Alzantot et al., 2018; Tu et al., 2018). Therefore, our problem can be formulated as:

$$\arg\max_{i\in\{1,\dots,C\}} f(\mathbf{x}_{origin}+\boldsymbol{\delta})_i = t \quad \text{s.t.} \quad \|\boldsymbol{\delta}\|_\infty \leq \delta_{max} \tag{6.1}$$

6.3.2 Bayesian Optimisation

BO is a query-efficient approach to tackle global optimisation problems (Brochu et al., 2010a). It is particularly useful when the objective function is a black-box and is costly to evaluate. Please refer to Chapter 2 for detailed explanation on BO.

6.4 BayesOpt Attack

6.4.1 BayesOpt Attack Objective

It has been shown that reducing the dimensionality of the search space in Equation (6.1) increases query efficiency significantly (Chen et al., 2017; Tu et al., 2018; Alzantot et al., 2018). Due to our focus on the small query regime where our surrogate model needs to be trained with a very small number of observation data, we adopt the previously suggested dimensionality reduction technique, bilinear resizing, to reduce the challenging problem of optimising over high-dimensional input space of $\mathbf{x} \in \mathbb{R}^d$ to one over a relatively low-dimensional input space, setting $\mathbf{x} = \mathbf{x}_{origin} + g(\boldsymbol{\delta})$ where $\boldsymbol{\delta} \in \mathbb{R}^{d^r}$ with $d^r < d$ and $g : \mathbb{R}^{d^r} \longrightarrow \mathbb{R}^d$ being the bilinear resizing decoder[2].

Furthermore, we follow the approach of smoothing the discontinuous objective function in Equation (6.1), which has been found to be beneficial in previous work (Chen et al., 2017; Tu et al., 2018; Alzantot et al., 2018). Together with the

[2]Note that our method is not dependent on a particular choice of decoder, i.e. it could also be used together with an autoencoder- or PCA-based dimensionality reduction technique.

dimensionality reduction, this leads to the following black-box objective problem for our BO:

$$\boldsymbol{\delta}^* = \arg\max_{\boldsymbol{\delta}} y(\boldsymbol{\delta}) \quad \text{s.t.} \quad \boldsymbol{\delta} \in [-\delta_{max}, \delta_{max}]^{d'} \tag{6.2}$$

$$\text{where} \quad y(\boldsymbol{\delta}) = \left[\log f(\mathbf{x}_{origin} + g(\boldsymbol{\delta}))_t - \log \sum_{j \neq t}^{C} f(\mathbf{x}_{origin} + g(\boldsymbol{\delta}))_j \right]$$

6.4.2 GP-based BayesOpt Attack

We first use BO with a standard GP as the surrogate to solve for the black-box attack objective in the reduced input dimension in Equation (6.2). The GP encodes our prior belief on the objective y :

$$y \sim \mathcal{GP}(\mu(\boldsymbol{\delta}), k(\boldsymbol{\delta}, \boldsymbol{\delta}')) \tag{6.3}$$

which is specified by a mean function μ and a kernel/covariance function k (we use the Matern-5/2 kernel in this work). In this work, we normalise the objective function value and thus use a zero-mean prior $\mu(\cdot) = 0$. The predictive posterior distribution for y_t at a test point δ_t conditioned on the observation data $\mathcal{D}_{t-1} = \{(\boldsymbol{\delta}_i, y_i)\}_{i=1}^{t-1} = \{\boldsymbol{\delta}_{1:t-1}, \mathbf{y}_{1:t-1}\}$ then has the form:

$$p(y_t|\boldsymbol{\delta}_t, \mathcal{D}_{t-1}) = \mathcal{N}(y; \mu_y(\cdot), \sigma_y(\cdot,\cdot)^2) \tag{6.4}$$

where

$$\mu_y(\boldsymbol{\delta}_t) = K(\boldsymbol{\delta}_t, \boldsymbol{\delta}_{1:t-1})\mathbf{K}_{1:t-1}^{-1}\mathbf{y}_{1:t-1}, \tag{6.5}$$

$$\sigma_y(\boldsymbol{\delta}_t, \boldsymbol{\delta}_t')^2 = K(\boldsymbol{\delta}_t, \boldsymbol{\delta}_t') - K(\boldsymbol{\delta_t}, \boldsymbol{\delta}_{1:t-1})\mathbf{K}_{1:t-1}^{-1}K(\boldsymbol{\delta}_{1:t-1}, \boldsymbol{\delta}_t'). \tag{6.6}$$

where $\mathbf{K}_{1:t-1} = K(\boldsymbol{\delta}_{1:t-1}, \boldsymbol{\delta}_{1:t-1})$ is the $t-1 \times t-1$ matrix with pairwise covariances $[\mathbf{K}_{1:t-1}]_{l,m} = k(\boldsymbol{\delta}_l, \boldsymbol{\delta}_m')$, $l, m \leq t-1$ as entries. The optimal GP hyper-parameters $\boldsymbol{\theta}^*$ such as the length scales and variance of the kernel function k can be learnt by maximising the marginal likelihood $p(\mathcal{D}_{t-1}|\boldsymbol{\theta})$ which has analytic form as presented in Section 2.1.2. Based on the predictive posterior distribution, we construct the acquisition function $\alpha(\boldsymbol{\delta}|\mathcal{D}_{t-1})$ to help select the next query point $\boldsymbol{\delta}_t$. The approach of using BO with a GP surrogate to attack the victim model is described in Algorithm 8.

Algorithm 8 BayesOpt Attack

1: **Input:** A black-box function y, observation data $\mathcal{D}_0$, iteration budget T, a decoder $g(\cdot)$
2: **Output:** The best recommended adversarial example $x^* = \mathbf{x}_{origin} + g(\boldsymbol{\delta}^*)$
3: **for** $t = 1, \ldots, T$ **do**
4: Select $\boldsymbol{\delta}_t = \arg\max \alpha_t(\boldsymbol{\delta}|\mathcal{D}_{t-1})$
5: $y_t = y(\boldsymbol{\delta}_t)$ and $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup (\boldsymbol{\delta}_t, y_t)$
6: Update the surrogate model with $\mathcal{D}_t$
7: **end for**

6.4.3 Additive GP-based BayesOpt Attack

Although techniques such as bilinear resizing are able to reduce the input dimension from the original image size (e.g. $d = 3072$ for CIFAR10 image) to a much lower dimension (e.g. $d' = 192$), the reduced search space for the adversarial attack is still considered very high dimensional for GP-based BO (which is usually applied on problems with $d' \leq 20$).

To alleviate the challenges of using BO for high-dimensional problems (as discussed in Section 2.3.2), we adopt the additive-GP model (Duvenaud et al., 2011; Kandasamy et al., 2015) to search for adversarial perturbation in the high dimensional space. The key assumption we make is that the objective can be decomposed into a sum of low-dimensional composite functions:

$$y(\boldsymbol{\delta}) = \sum_{j=1}^{M} y^{(j)}(\boldsymbol{\delta}^{(A_j)}) \tag{6.7}$$

where $\boldsymbol{\delta}^{(A_j)}$ denotes disjoint low-dimensional subspaces, $\boldsymbol{\delta} = \cup_{j=1}^{M} \boldsymbol{\delta}^{(A_j)}$ and $\boldsymbol{\delta}^{(A_j)} \cap \boldsymbol{\delta}^{(A_i)} = \emptyset$ for all $j \neq i$. If we impose a GP prior for each $y^{(j)}$, the prior for the overall objective y is also a GP: $y \sim \mathcal{GP}(\mu(\boldsymbol{\delta}), k(\boldsymbol{\delta}, \boldsymbol{\delta}'))$ where $\mu(\boldsymbol{\delta}) = \sum_{j=1}^{M} \mu^{(j)}\left(\boldsymbol{\delta}^{(A_j)}\right)$ and $k(\boldsymbol{\delta}, \boldsymbol{\delta}') = \sum_{j=1}^{M} k^{(j)}\left(\boldsymbol{\delta}^{(A_j)}, \boldsymbol{\delta}'^{(A_j)}\right)$. The predictive posterior distribution for each subspace $p(y_{1:t-1}^{(j)}|\boldsymbol{\delta}_t^{(A_j)}, \mathcal{D}_t)$ is:

$$\mu_y^{(j)}(\boldsymbol{\delta}_t^{(A_j)}) = K^{(j)}(\boldsymbol{\delta}_t^{(A_j)}, \boldsymbol{\delta}_{1:t-1}^{(A_j)})\mathbf{K}_{1:t-1}^{-1}\mathbf{y}_{1:t-1},$$
$$\sigma_y^{(j)}(\boldsymbol{\delta}_t^{(A_j)}, \boldsymbol{\delta}_t'^{(A_j)})^2 = K^{(j)}(\boldsymbol{\delta}_t^{(A_j)}, \boldsymbol{\delta}_t'^{(A_j)}) - K^{(j)}(\boldsymbol{\delta}_t^{(A_j)}, \boldsymbol{\delta}_{1:t-1}^{(A_j)})\mathbf{K}_{1:t-1}^{-1}K^{(j)}(\boldsymbol{\delta}_{1:t-1}^{(A_j)}, \boldsymbol{\delta}_t'^{(A_j)}).$$

In our case, the exact decomposition (i.e. which input dimension belongs to which low-dimensional subspace) is unknown but we can learn it together with other GP

hyperparameters by maximising marginal likelihood (Kandasamy et al., 2015). We demonstrate the effectiveness of our decomposition learning in Section 6.5.4. Note that in this case, the acquisition function is formulated based on $p(y_t^{(j)}|\boldsymbol{\delta}_t^{(A_j)}, \mathcal{D}_t)$ for each subspace and is also optimised in the low-dimensional subspace, thus leading to much more efficient optimisation task. The optimal perturbations in all the subspaces $\{\boldsymbol{\delta}^{(A_j)*}\}_{j=1}^{M}$ are then combined to give the next query point $\boldsymbol{\delta}_t$.

6.4.4 Learning The Optimal d^r

Generating the successful adversarial example $\mathbf{x} \in \mathbb{R}^d$ by searching perturbation in a reduced dimension $\boldsymbol{\delta} \in \mathbb{R}^{d^r}$ has become a popular practice that leads to significant improvement in query efficiency (Chen et al., 2017; Tu et al., 2018; Alzantot et al., 2018). However, what the optimal d^r is and how to decide it efficiently have not been investigated in previous work (Chen et al., 2017; Tu et al., 2018; Alzantot et al., 2018). As we shown empirically in Section 6.5.1, setting d^r arbitrarily can lead to suboptimal attack performance in terms of query efficiency, attack success rate as well as perturbation costs while finding a good d^r by trial and error or progressive increasing is computationally expensive and highly inefficient because the optimal d^r varies with different attack input $\mathbf{x}_{origin}$. An effective way of learning d^r is thus very important for adversarial attacks. In this section, we propose a rigorous method, which is neatly compatible with our attack technique, to learn the optimal d^r from the query information.

The optimal d^r should be the one that both takes into consideration our prior knowledge on the discrete d^r choices and at the same time best explain the observed query data. Given that our BayesOpt attack uses a statistical surrogate (i.e. GP in our case) to model the unknown relation between the attack objective score y and the adversarial perturbation $\boldsymbol{\delta}$, this naturally translates to the criterion of maximising the posterior for d^r:

$$p(d_j^r|\mathcal{D}_{t-1}) = \frac{p(\mathcal{D}_{t-1}|d_j^r)p(d_j^r)}{p(\mathcal{D}_{t-1})} \tag{6.8}$$

where $p(\mathcal{D}_{t-1}|d_j^r)$ is the marginal likelihood (evidence) of adopting a specific d_j^r, $p(d_j^r)$ is our prior over the possible d^r values and $p(\mathcal{D}_{t-1}) = \sum_i p(\mathcal{D}_{t-1}|d_j^r)p(d_j^r)$ is a normalising constant. If we match different d_j^r to different model choice, the problem of choosing d^r then becomes the classic *Bayesian model selection* task (Rasmussen, 2003). In most cases, our prior assumption is that we do not prefer one d^r over another (i.e. flat prior $p(d_j^r) = p(d_i^r)$ for $j \neq i$) and thus we can select d^r by comparing their evidence(MacKay, 2003):

$$\frac{p(d_i^r|\mathcal{D}_{t-1})}{p(d_j^r|\mathcal{D}_{t-1})} = \frac{p(\mathcal{D}_{t-1}|d_i^r)p(d_i^r)}{p(\mathcal{D}_{t-1}|d_j^r)p(d_j^r)} = \frac{p(\mathcal{D}_{t-1}|d_i^r)}{p(\mathcal{D}_{t-1}|d_j^r)}. \tag{6.9}$$

The exact computation of the evidence term requires marginalisation over model hyper-parameters, which is intractable. We approximate the integral with point estimates (i.e. the maximum marginal likelihood of our GP model): $p(\mathcal{D}_{t-1}|d_j^r) = \int p(\mathcal{D}_{t-1}|\boldsymbol{\theta}, d_j^r)p(\boldsymbol{\theta}|d_j^r)d\boldsymbol{\theta} \approx p(\mathcal{D}_{t-1}|\boldsymbol{\theta}^*, d_j^r)$ where $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathcal{D}_{t-1}|\boldsymbol{\theta}, d_j^r)$. For query efficiency, we project the same perturbation query data in the original image dimension $\mathcal{D}_{t-1}^d = \{g(\boldsymbol{\delta}_i), y_i\}_{i=1}^{t-1}$ to different latent spaces to get corresponding low-dimensional training data sets for separate GP models. The GP model that corresponds to d_j^r is trained on $\mathcal{D}_{t-1}^{d_j^r} = \{(g^{-1}(g(\boldsymbol{\delta}_i)), y_i)\}_{i=1}^{t-1}$ where $g^{-1}(g(\boldsymbol{\delta}_i)) \in \mathbb{R}^{d_j^r}$. Then we select the optimal d^r by comparing the marginal likelihood $p(\mathcal{D}_{t-1}^{d_j^r}|\boldsymbol{\theta}^*, d_j^r)$ of each GP surrogate. The overall procedure for d^r selection is described in Algorithm 9. We would like to highlight that the use of the statistical surrogate in our BayesOpt attack approach enables us to naturally use the Bayesian model selection technique to learn the optimal d^r that automatically enjoy the trade-off between data-fit quality and model complexity. And we also show empirically in Section 6.5.3 that by automating and incorporating the learning of d^r into our BayesOpt attack, we can gain higher success rate and query efficiency. Other adversarial attacks methods can also use our proposed approach to decide d^r but it would require the additional efforts of constructing statistical models to provide $p(\mathcal{D}_{t-1}|d_j^r)$.

Algorithm 9 Bayesian selection of d^r

1: **Input:** A decoder $g(\cdot)$, observation data $\mathcal{D}_{t-1}^d = \{(g(\boldsymbol{\delta}_i), y_i)\}_{i=1}^{t-1}$ where $g(\boldsymbol{\delta}_i) \in \mathbb{R}^d$ and a set of possible $d^r : \{d_j^r\}_{j=1}^N$
2: **Output:** The optimal reduced dimension d^{r*} and the corresponding GP model
3: **for** $j = 1, \ldots, N$ **do**
4: $\quad \mathcal{D}_{t-1}^{d_j^r} = \{(g^{-1}(g(\boldsymbol{\delta}_i)), y_i)\}_{i=1}^{t-1}$ where $g^{-1}(g(\boldsymbol{\delta}_i)) \in \mathbb{R}^{d_j^r}$
5: $\quad$ Fit a GP model to $\mathcal{D}_{t-1}^{d_j^r}$ and computes its maximum marginal likelihood $p(\mathcal{D}_{t-1}^d | \boldsymbol{\theta}^*, d_j^r)$
6: **end for**
7: Select $d^{r*} = \arg\max_{d_j^r \in \{d_j^r\}_{j=1}^N} p(\mathcal{D}_{t-1}^d | \boldsymbol{\theta}^*, d_j^r)$ and its correspond GP model

6.5 Experiments

We empirically compare the performance of our BayesOpt attacks against the state-of-the-art black-box methods such as ZOO (Chen et al., 2017), AutoZOOM(Tu et al., 2018) and GenAttack (Alzantot et al., 2018). We denote the BayesOpt attack with standard GP surrogate as GP-BO, its variant that learns the d^r automatically is GP-BO-auto-d^r as well as the attack with additive GP surrogate as ADDGP-BO. For all the BO methods, we use GP-UCB as the acquisition function [3] and update the GP hyperparameters every 5 BO iterations. We relearn the optimal d^r for GP-BO-auto-d^r and the search space decomposition for ADDGP-BO every 40 iterations. For ADDGP-BO, we decompose the search spaces into $M = 12$ sub-spaces of equal dimensions for MNIST and CIFAR10 and into $M = 27$ sub-spaces for ImageNet.

The victim models that we attack follow the same architectures as that used in AutoZOOM and GenAttack; These are image classifiers for MNIST (a CNN with 99.5% test accuracy), CIFAR10 (a CNN with 80% test accuracy) and ImageNet (the InceptionV3 with 78% test accuracy). Following the experiment design in Tu et al. (2018), we randomly select 50 correctly classified images from CIFAR10 test data and MNIST test data. We then perform targeted attacks on these images. Each selected image is attacked 9 times, targeting at all but its true class and this gives a total of 450 attack instances for both CIFAR10 and MNIST. For ImageNet, we also select 50 correctly classified images from the test set but perform

[3] The parameter that trades off exploration and exploitation is set to 2.

one random targeted attack for each image. We set $\delta_{max} = 0.3$ for attacking MNIST and $\delta_{max} = 0.05$ for CIFAR10 and ImageNet, which are used in Alzantot et al. (2018). We use the recommended parameter setting and their open sourced implementations for performing all competing attack methods (Chen et al., 2017; Tu et al., 2018; Alzantot et al., 2018).

6.5.1 Effect of d^r

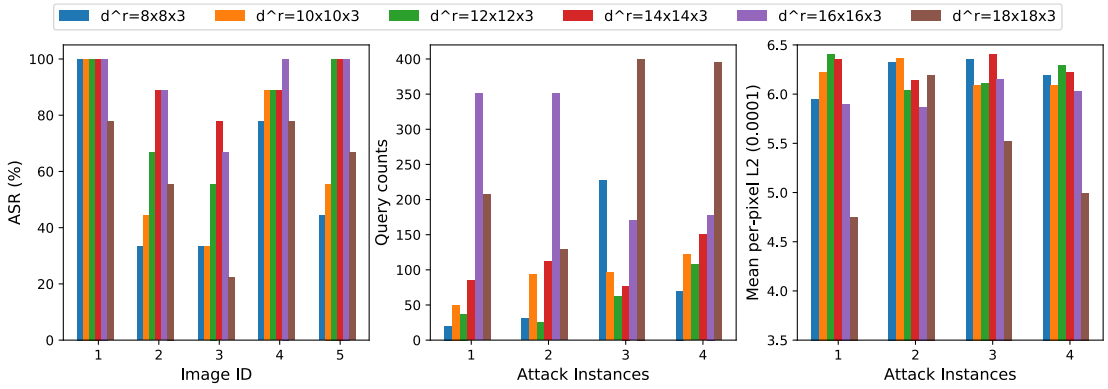


Figure 6.1: Performance of BayesOpt attack with GP surrogate in different reduced dimension d^r. Each color represents a particular d^r. The **left** subplot shows the attack success rates(ASR) for attacking all the other 9 classes on each of the 5 original images. The d^r that leads to the highest ASR varies with the original images. The **middle** subplot shows the number of queries needed to attack the same attack instances (i.e. the same original image and the same attack target) by using different d^r. For the same 4 attack instances, the **right** subplot show the effect of the choice of d^r on the L_2 distances between the resultant adversarial examples and the original image.

We first empirically investigate the effect of the reduced dimensionality d^r of the latent space in which we search for the adversarial perturbation. We experiment with the GP-based BayesOpt attacks for the CIFAR10 classifier using reduced dimension of $d_r = \{6\times6\times3, 8\times8\times3, 10\times10\times3, 12\times12\times3, 14\times14\times3, 16\times16\times3, 18\times18\times3\}$ and perform targeted attacks on 5 target images with each image being attacked 9 times, leading to 45 attack instances. We first investigate the attack success rate (ASR) achieved at different d^r for all the 5 images. The results on ASR out of the 9 targeted attacks for each of the 5 images, which are indicated by Image ID 1 to 5, are shown in the left subplot of Figure 6.1. It's evident that the d^r

which leads to highest attack success rate varies for different original images $\mathbf{x}_{origin}$. More results are shown in Appendix A.4.

We then examines how the d^r affects the query efficiency and attack quality (average L_2 distance of the adversarial image from the original image) for the attack instances (e.g. make the classifier to mis-classify a airplane image as a cat) that GP-based BO can attack successfully at all d^r. We present the results on 4 attack instances of an airplane image in the middle and right subplots of Figure 6.1. We can see that even for the same original image and attack instances, varying d^r can impact query efficiency and L_2 norm of the successful adversarial perturbation significantly. For example, $d^r = 8 \times 8 \times 3$ is most query efficient for attack instance 1 and 4 but is outperformed by other dimensions in attack instance 2 and 3. Therefore, the importance of d^r and the difficulty of finding the optimal d^r for a specific target/image motivates us to derive our method for learning it automatically from the data. As shown in the following sections, our d^r learner does lead to more superior attack performance.

6.5.2 Performance Under a Fixed Query Budget

Table 6.1: Attack results on 50 random MNIST images by using $d^r = 14 \times 14 \times 1$. Q denotes the query count. *ASR* denotes attack success rate. The standard errors are in parentheses.

Attack method	*ASR*	*Q (Max, Median, Mean)*	Average L_2 perturbation (per pixel)
ADDGP-BO	97%	$829, 46, 95(\pm 6)$	$6.81 \times 10^{-3}(\pm 3.12 \times 10^{-5})$
GP-BO-auto-d^r	98%	$833, 75, 121(\pm 6)$	$6.89 \times 10^{-3}(\pm 5.55 \times 10^{-5})$
GP-BO	82%	$899, 42, 77(\pm 5)$	$7.15 \times 10^{-3}(\pm 4.17 \times 10^{-5})$
GenAttack	92%	$986, 146, 217(\pm 9)$	$6.66 \times 10^{-3}(\pm 2.81 \times 10^{-5})$
AutoZOOM	99%	$998, 229, 265(\pm 9)$	$5.05 \times 10^{-3}(\pm 7.71 \times 10^{-5})$
ZOO	1%	$256, 128, 128(\pm 64)$	$2.78 \times 10^{-3}(\pm 3.53 \times 10^{-5})$

In this experiment, we limit the total query number to be 1000, which is slightly above the median query counts needed for GenAttack to make a successful attack on MNIST and CIFAR10 (Alzantot et al., 2018) and 2000 for ImageNet. We adopt the reduced search dimension recommended by AutoZOOM, $d^r = 14 \times 14 \times 3$ for

Table 6.2: Attack results on 50 randomly selected CIFAR10 images with $d^r = 14 \times 14 \times 3$. Q denotes the query count. *ASR* denotes attack success rate.The standard errors are in parentheses.

Attack method	*ASR*	*Q (Max, Median, Mean)*	Average L_2 perturbation (per pixel)
ADDGP-BO	87%	885, 154, 222($\pm$10)	$5.87 \times 10^{-4}(\pm 3.22 \times 10^{-6})$
GP-BO-auto-d^r	87%	891, 159, 234($\pm$10)	$5.96 \times 10^{-4}(\pm 5.00 \times 10^{-6})$
GP-BO	72%	899, 141, 190($\pm$8)	$5.78 \times 10^{-4}(\pm 4.67 \times 10^{-6})$
GenAttack	68%	991, 246, 329($\pm$14)	$7.38 \times 10^{-4}(\pm 4.14 \times 10^{-6})$
AutoZOOM	38%	896, 102, 154($\pm$11)	$9.97 \times 10^{-4}(\pm 1.53 \times 10^{-5})$
ZOO	4%	768, 256, 369($\pm$57)	$1.77 \times 10^{-4}(\pm 1.16 \times 10^{-5})$

Table 6.3: Attack results on 50 randomly ImageNet images with random target label under a query budget of 2000. Q denotes the query count. *ASR* denotes attack success rate. The standard errors are in parentheses. ZOO fails to make any successful attack within this budget.

Attack method	*ASR*	*Q (Max, Median, Mean)*	Average L_2 perturbation (per pixel)
ADDGP-BO	60%	1985, 1247, 1206($\pm$90)	$1.74 \times 10^{-4}(\pm 1.89 \times 10^{-6})$
GP-BO-auto-d^r	32%	1999, 922, 938($\pm$128)	$1.96 \times 10^{-4}(\pm 8.07 \times 10^{-6})$
GP-BO	16%	1947, 1113, 1232($\pm$162)	$1.62 \times 10^{-4}(\pm 7.94 \times 10^{-6})$
GenAttack	12%	1971, 1354, 1266($\pm$259)	$2.03 \times 10^{-4}(\pm 9.81 \times 10^{-6})$
AutoZOOM	2%	1451, 1451, 1451($\pm$0.00)	$1.21 \times 10^{-5}(\pm 0.00)$

CIFAR10 and $d^r = 14 \times 14 \times 1$ for MNIST and allow GP-BO-auto-d^r to automatically learn the reduced dimension d^r in a range between $6 \times 6 \times 1$ and $28 \times 28 \times 1$ for MNIST, and between $6 \times 6 \times 3$ and $32 \times 32 \times 3$ for CIFAR10.

For ImageNet, due to the high dimensionality of its images, we adopt a hierarchical decoding process: 1) first performance BayesOpt attacks(ADDGP-BO and GP-BO) on a reduced dimension of $d_1^r = 48 \times 48 \times 3$ and then 2) decode the adversarial perturbation found in d_1^r to $d_2^r = 96 \times 96 \times 3$ via bilinear upsampling. 3) This is followed by another bilinear decoder projecting the adversarial perturbation in d_2^r back to image dimension of $d = 299 \times 299 \times 3$. As for the GP-BO-auto-$d^r$, we allow the algorithm to automatically learn the reduced dimension d_1^r in the phase 1) in the range between $6 \times 6 \times 3$ to $60 \times 60 \times 3$. GenAttack is performed with the same upsampling as our methods. As for AutoZOOM and ZOO, we uses the optimal d^r

settings recommended in Tu et al. (2018); Chen et al. (2017) for ImageNet.

For BayesOpt attack, each iteration requires 1 query to the objective function so we limit its iteration to 1000 and early terminate the BayesOpt attack algorithm when successful adversarial example is found. AutoZOOM comprises 2 stages in their attack algorithm. The first exploration stage aims to find the successful adversarial example. Once a successful attack is found, it switches to the fine-tuning stage to reduce the perturbation cost (e.g. L_2 norm) of the successful attack. We report its performance on the attack success rate and L_2 norm of adversarial perturbations after a budget of 1000 queries, which allows it to fine-tune the successful adversarial perturbations found. Moreover, AutoZOOM uses L_2 norm to measure the perturbation costs but our method and GenAttack limit the search space via L_∞ norm. We observe that the successful attacks found by AutoZOOM incur much higher perturbation costs in terms of L_∞ norm than GenAttack and our method. Therefore, we only consider the final adversarial examples whose L_∞ distances from the original images lie within $[-\delta_{max}, \delta_{max}]$ as the successful attacks [4].

The results on MNIST in Table 6.1 show that all our BayesOpt attacks can achieve a comparable success rate but at a much lower query count in comparison to GenAttack (Alzantot et al., 2018) and AutoZOOM (Tu et al., 2018). Specifically, the median query count of ADDGP-BO is 68% less than GenAttack and 80% less than AutoZOOM while that of GP-BO-auto-d^r is 49% less than GenAttack and 67% less than AutoZOOM.

As for results on attacking on CIFAR10, Table 6.2 shows that all our BayesOpt attack can achieve significantly higher success rate but again at a remarkably lower query counts than the existing black-box approaches. For example, our ADDGP-BO can achieve 18% higher success rate while using 37.4% less queries in terms of the median as compared to GenAttack. In addition, our approaches also lead to better quality adversarial examples which are closer to the original image than the

[4]This still gives an advantage to ZOO and AutoZOOM because it allows them to inject perturbation whose magnitude is larger than $\boldsymbol{\delta}_{max}$ at certain pixels but our BayesOpt methods and GenAttack limit perturbation at all the pixels to be less than $\boldsymbol{\delta}_{max}$.

benchmark methods as reflected by the lower average L_2 perturbation (20.5% less). Figure 6.2 shows some CIFAR10 adversarial examples found by our BayesOpt attack.

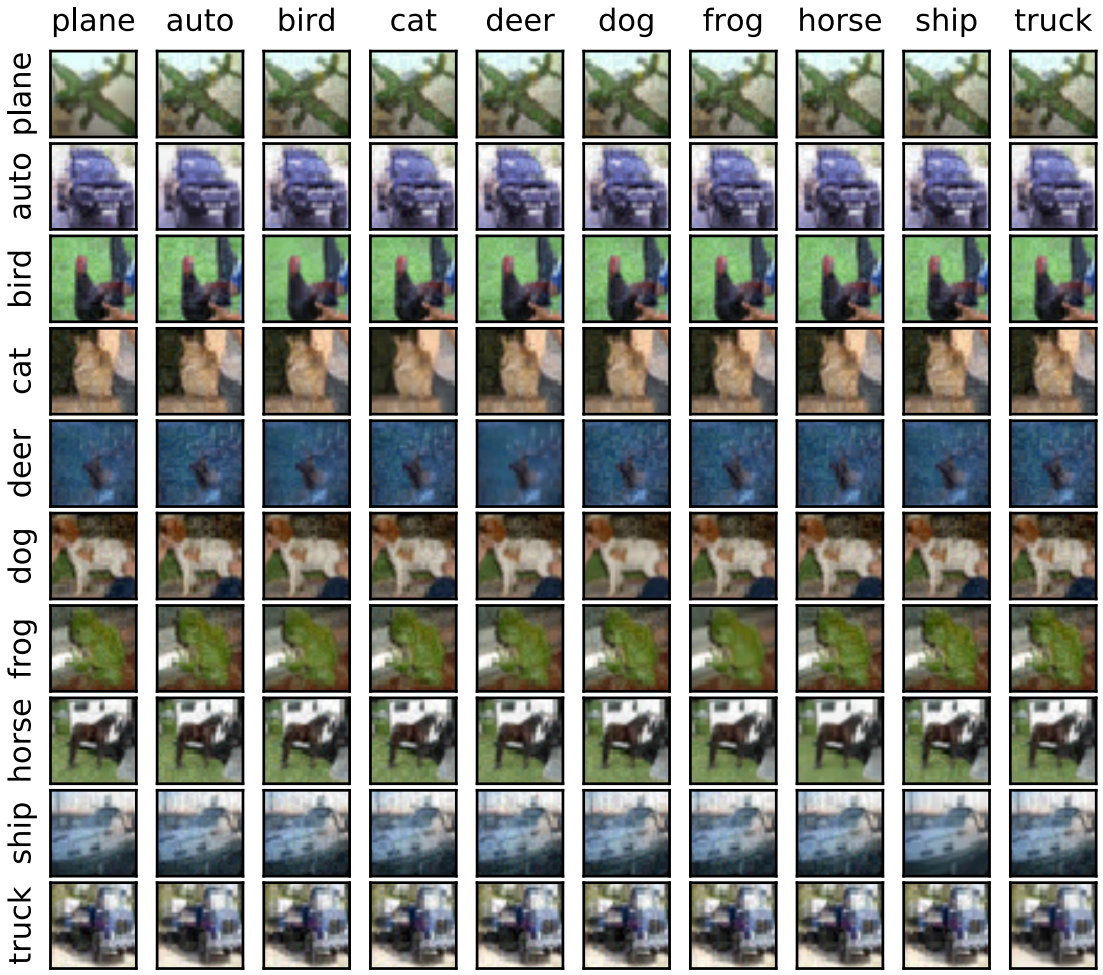


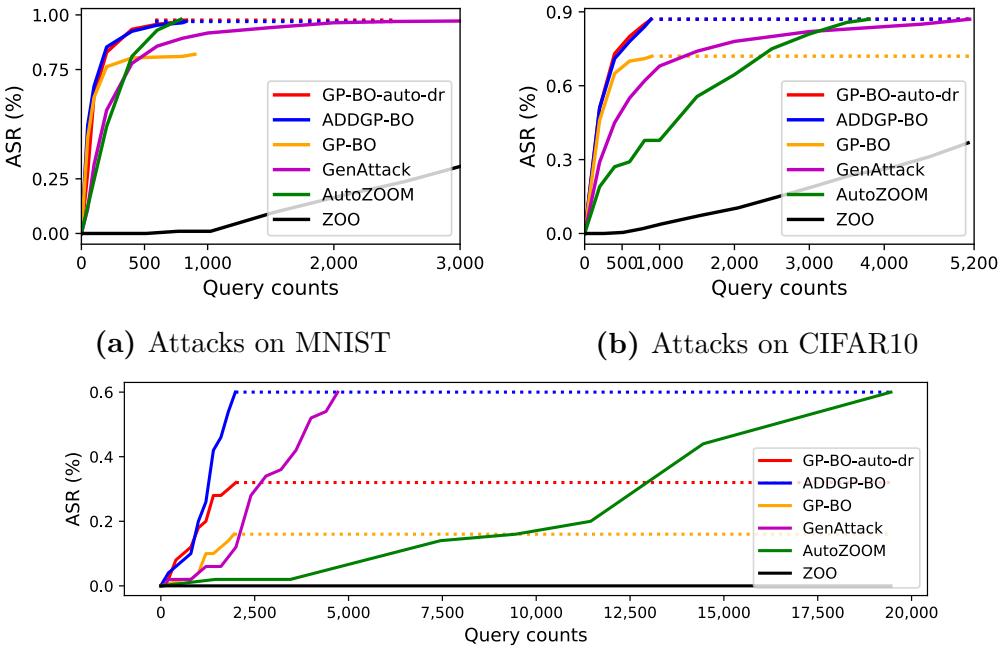
Figure 6.2: CIFAR10 adversarial examples generated by our BayesOpt attack. True labels and target labels correspond to the rows and columns.

More importantly, this set of experiments also demonstrate the effectiveness of our Bayesian method for learning d^r as GP-BO-auto-d^r leads to 15% increase in attack success rate compared to GP-BO while maintaining the competitiveness in query efficiency and L_2 distance.

Similarly, the results on ImageNet in Table 6.3 shows that all our BayesOpt attacks can achieve higher attack success rate than the competing methods given a query budget of 2000. ADDGP-BO is the clearly best performing method in this very high-dimensional setting, achieving 5 times higher success rate than the best competing method, GenAttack, while again obtaining successful adversarial

perturbations with lower average L_2 perturbation (14 % less) than GenAttack. This confirms our hypothesis that the additive-GP surrogate together with decomposition learning is very effective for high-dimensional optimisation.

6.5.3 Query Efficiency Comparison



(a) Attacks on MNIST **(b)** Attacks on CIFAR10

(c) Attacks on ImageNet

Figure 6.3: Query efficiency of BayesOpt Attacks. The plots show the attack success rate (ASR) of different methods up to certain query counts. The best BayesOpt attacks (i.e. GP-BO-auto-d^r (red) and ADDGP-BO (blue)) can achieve an ASR of 98% on MNIST(a) and 87% on CIFAR10 (b) within a budget of 1000 queries. To achieve the same success rates, GenAttack (purple) takes 3500 for MNIST and 5141 for CIFAR10, and AutoZOOM (green) takes 800 for MNIST and 3880 for CIFAR10. For ImageNet (c), the best BayesOpt attack is ADDGP-BO (blue), which achieves an ASR of 60% with 1985 queries. To achieve the same success rates, GenAttack (purple) takes a budget of 4711 queries and AutoZOOM (green) takes 19451 queries. ZOO fails to make any attack on ImageNet within the given budget.

We finish by comparing the query efficiency, measuring the change in the attack success rate over query counts for all the methods. We limit the query budget of our BayesOpt attacks to 1000 but let the competing methods to continue running until they achieve the same best attack success rate as our best BayesOpt attacks or exceeds a much higher limit (2000 queries for MNIST and 5000 queries for

CIFAR10 and 14000 queries for ImageNet).

As shown in Figure 6.3, BayesOpt attacks converge much faster to the high attack success rates than the other methods. Specifically, GP-BO-auto-d^r take less than 833 queries to achieve the success rate of 98% for MNIST, which is 24% of that by GenAttack (2500). As for CIFAR10, both ADDGP-BO and GP-BO-auto-d^r takes around 890 queries to achieve a success rate of 87% which is 23% of the query count by AutoZOOM and 17% of that by GenAttack. One point to note is that AutoZOOM appears to be slightly more query efficient than GenAttack in this set of experiments. However, we need to bear in mind that although we limit the mean L_{inf} norm of the adversarial perturbation found by AutoZOOM to δ_{max}, AutoZOOM still have the advantage of exploring beyond the δ_{max} for many dimensions. In the case of ImageNet, all our BayesOpt attacks again enjoy faster convergence, with ADDGP-BO taking only 1985 queries to achieve 60% ASR. Meanwhile, GenAttack, takes a budget of 4711 queries, 2.4 times that of ADDGP-BO, to achieve the same ASR and AutoZOOM costs 19451 queries which is 9.8 times that of ADDGP-BO.

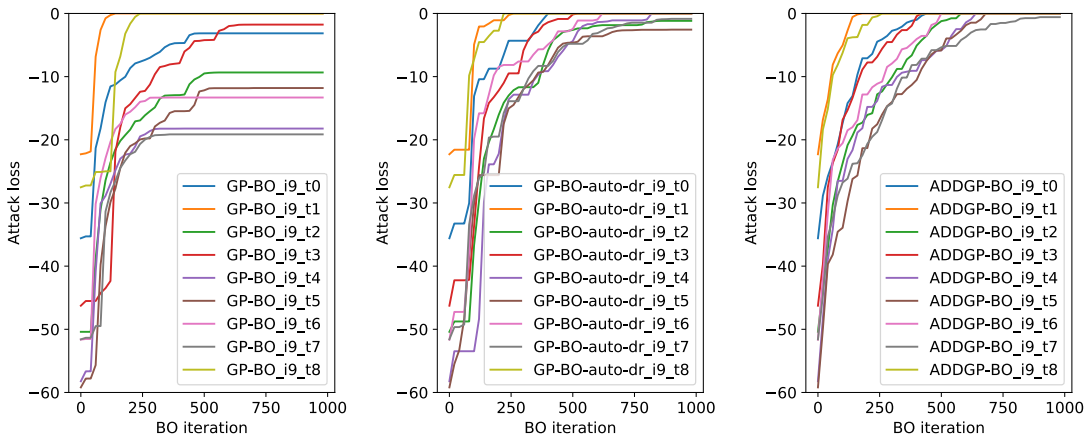
6.5.4 Ablation Studies

Objective Value over BO Iterations

We illustrate t via the case of attacking a CIFAR10 image of label 9(class truck) on the other 9 target labels with original label 9. We plot the value of objective function (Equation 6.2), which is equal to the negative of attack loss, against the BO iterations/query counts. We can see that our additive GP surrogate (ADDGP-BO) as well as the Bayesian learning of optimal d^r (GP-BO-auto-d^r) lead to faster convergence and thus higher attack success rate for this instance.

Effectiveness of Decomposition Learning

As mentioned in Section 6.4.3, to learn the decomposition for the additive-GP surrogate in ADDGP-BO attack, we follow the approach proposed in Kandasamy et al. (2015) to treat the decomposition as an additional hyperparameter and learn the optimal decomposition by maximising marginal likelihood. However, exhaustive



(a) GP-BO **(b)** GP-BO-auto-d^r **(c)** ADDGP-BO

Figure 6.4: Attack objective value against BayesOpt iterations(query count) for using various BayesOpt methods to attack one CIFAR10 image of class label 9 (denoted by i). Curves of different colours correspond to the 9 different target labels (denoted by t). Convergence to 0 objective value indicates successful attack. ADDGP-BO and GP-BO-auto-d^r enjoy faster convergence and thus higher attack success rates than simple GP-BO.

Table 6.4: Summary of attack results on CIFAR10 for different decomposition learning method. $d_r = 14 \times 14 \times 3$ which is further decomposed into 12 subspaces of $d_s = 49$. Q denotes the query count. ASR denotes attack success rate. The standard errors are in parentheses.

Attack method	*ASR*	*Q (Max, Median, Mean)*	Average L_2 perturbation (per pixel)
ADDGP-BO-LD	94%	$885, 134, 212(\pm 16)$	$5.78 \times 10^{-4}(\pm 5.32 \times 10^{-6})$
ADDGP-BO-FD	87%	$899, 143, 196(\pm 14)$	$5.05 \times 10^{-4}(\pm 4.52 \times 10^{-6})$

search over all $M!d!/(d_s!^M)$[5] possible decompositions is expensive. We adopt a computationally cheap alternative by randomly selecting 20 decompositions and choosing the one with the largest marginal likelihood. The method decomposition learning procedure is repeated every 40 BO iterations and we use $M = 12$ for CIFAR10. We denote the method as ADDGP-BO-LD in this section but as ADDGP-BO in the rest of the chapter.

We also experiment with another alternative way to learn the decomposition (ADDGP-BO-FD), which is similar to importance sampling in ZOO. We group the pixels/dimensions together if the magnitude of average change in their pixel

[5] M is the number of subspaces and $d_s = |A_j|$ is the dimension of each subspaces

values over the past 5 BO iterations are closer (i.e. pixels that are subject to the most large adversarial perturbations are grouped together). Again, we divide the dimensions into disjoint 12 groups as above to ensure fair comparison.

We compare the both types of decomposition learning methods using 20 randomly selected CIFAR10 images, each of which is attacked on 9 other classes except its original class and a query budget of 1000. The results are shown in Table 6.4. It is evident that learning decomposition by maximising marginal likelihood (ADDGP-BO-LD) can achieve higher attack success rate than pixel-value-change-based decomposition learning (ADDGP-BO-FD) given the query budget.

6.6 Extension Work

Besides attacking image classification models, we demonstrate in a recent work (Wan et al., 2021a) that BO can be used to efficiently attack *graph* neural networks in a black-box setting. We name this first BO-based attack method for graph classification models, GRABNEL, which stands for Graph Adversarial attack via Bayesian Efficient Loss-minimisation. In view of the growing interest in applying graph-based machine learning models in various important applications such as semi-supervised learning, link prediction, community detection and graph classification (Cai et al., 2018b; Zhou et al., 2020a; Hamilton, 2020), our GRABNEL can be a useful tool to efficiently assess the adversarial robustness of complex graph-based models designed by human practitioners or discovered by AutoML. We present the overall algorithm of GRABNEL in Algorithm 10 and explain some key design features in the following subsections. For readers who are interested in more details about our attack method, please refer to (Wan et al., 2021a).

6.6.1 Problem Setup

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V} = \{v_i\}_{i=1}^n$ and edges $\mathcal{E} = \{\mathbf{e}_i\}_{i=1}^m$ where each edge $\mathbf{e}_k = \{v_i, v_j\}$ connects between nodes v_i and v_j. The overall

Algorithm 10 GRaph Adversarial attack via BayesiaN Efficient Loss-minimisation (GRABNEL)

Input: Original graph $\mathcal{G}_0$, victim model f, number of random initialising points n_{init}, query budget B, perturbation budget Δ
Output: An adversarial graph $\mathcal{G}^*$
Set base graph $\mathcal{G}_{\text{base}} \leftarrow \mathcal{G}_0$; initialise stage count $\texttt{stage} \leftarrow 0$
Randomly sample n_{init} perturbed graphs $\{\mathcal{G}'\}_{i=1}^{n_{\text{init}}}$ that are 1 edit distance different from $\mathcal{G}$ and query each perturbed graph to obtain their attack losses $\{y_{\text{attack}}(\mathcal{G}'_i)\}_{i=1}^{n_{init}}$
Compute the WL feature encoding for all graphs: $[\phi(\mathcal{G}'_1), \ldots, \phi(\mathcal{G}'_{n_{\text{init}}})] = \texttt{WLFeatureExtract}(\mathcal{G}_0, (\mathcal{G}'_1, \ldots \mathcal{G}'_{n_{\text{init}}}))$
Fit the sparse Bayesian linear regression surrogate with the data $\{(\phi(\mathcal{G}'_i), y_{\text{attack}}(\mathcal{G}'_i))\}_{i=1}^{n_{\text{init}}}$
Divide total budget of B into Δ stages
while query budget is not exhausted and attack has not succeeded **do**
  if query budget of the *current stage* is exhausted **then**
    Increment the stage count $\texttt{stage} \leftarrow \texttt{stage} + 1$ and update the base graph $\mathcal{G}_{\text{base}}$ with the graph leading to largest increase in attack loss in the previous stage.
  end if
  Propose graph to be queried next $\mathcal{G}'_{\text{proposal}}$ via acquisition optimisation
  Compute $y_{attack}(\mathcal{G}'_{\text{proposal}})$ by query the victim model f
  if attack succeeded **then**
    return $\mathcal{G}^* \leftarrow \mathcal{G}'_{\text{proposal}}$
  end if
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{G}'_{\text{proposal}}, y_{attack}(\mathcal{G}'_{\text{proposal}})\}$
  Update the WL feature encodings of *all* observed graphs $[\phi(\mathcal{G}'_1), \ldots, \phi(\mathcal{G}'_{|\mathcal{D}|})] = \texttt{WLFeatureExtract}(\mathcal{G}_0, (\mathcal{G}'_1, \ldots \mathcal{G}'_{|\mathcal{D}|}))$ and the surrogate model
end while
return `None` if fail to find successful attack within the query budget

topology can be represented by the adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$ where $\mathbf{A}_{ij} = 1$[6] if the edge $\{v_i, v_j\}$ is present.

The attack objective in this case is highly similar to, if not the same as, that in Section 6.3.1: we aim to degrade the predictive performance of the pre-trained victim graph classifier f by finding a graph $\mathcal{G}'$ perturbed from the original test graph $\mathcal{G}$ (ideally with the minimum amount of perturbation) such that f produces an incorrect class label for $\mathcal{G}$. Same as the case for image classification in Section 6.3.1, we assume the black-box attack setting where we can only interact with the

[6] We discuss the unweighted graphs for simplicity.

victim model f by querying it with an input graph $\mathcal{G}'$ and observe the model output as pseudo-probabilities over all classes $f(\mathcal{G}') \in [0,1]^C$. Additionally, we assume that *query efficiency* is highly valued. Note, different from attacking images in Section 6.3.1, adversarial attacks on graphs can happen at node level (cause the victim model to misclassify the node labels) or at graph level (cause the victim model to misclassify the class label of the entire graph). Our work focus on the latter setting which is important but yet less explored.

Formally, we can modify Equation (6.2) accordingly:

$$\max_{\mathcal{G}' \in \Psi(\mathcal{G})} y_{\text{attack}}\Big(\mathcal{G}'\Big) \tag{6.10}$$

where $y_{\text{attack}}\Big(\mathcal{G}'\Big)$ is the attack loss and $\Psi(\mathcal{G})$ denotes the set of possible $\mathcal{G}'$ generated from perturbing the original graph $\mathcal{G}$. In this work, we experiment with a diverse modes of attacks to show that our attack method can be generalised to different set-ups including but not limited to creating/removing edges and rewiring/swapping edges.

Denote the output logit for the class i as $f(\mathcal{G})_i$ and the correct label of the original graph is thus $t_{origin} = \arg\max_{i \in \{1,\dots,C\}} f(\mathcal{G}')_i$. The attack loss $y_{\text{attack}}\Big(\mathcal{G}'\Big)$ can be defined as:

$$y_{\text{attack}}\Big(\mathcal{G}'\Big) = \begin{cases} \max_{i \neq t_{origin}} \log f(\mathcal{G}')_i - \log f(\mathcal{G}')_{t_{origin}} & \text{(untargeted attack)} \\ \log f(\mathcal{G}')_t - \log f(\mathcal{G}')_{t_{origin}} & \text{(targeted attack on class } t\text{).} \end{cases}$$

6.6.2 Surrogate Model in GRABNEL

In the BO-based adversarial attack, we need to build a probabilistic surrogate model to locally learn the mapping from a perturbed graph $\mathcal{G}'$ to its attack loss $y_{\text{attack}}(\mathcal{G}')$. Under this graph attack setting, the surrogate model needs to be able to handle graph inputs naturally while remaining scalable to large graphs (e.g. in the order of 10^3 nodes or more) typical of common graph classification tasks with reasonable run-time efficiency. In view of this and inspired by the GP surrogate with Weisfeiler-Lehman(WL) graph kernel adopted in Chapter 7, we propose to first use a WL feature extractor to extracts the latent feature representation of $\mathcal{G}$, $\phi(\mathcal{G})$,

followed by fitting a *sparse Bayesian linear regression* on the latent feature space to balances performance with efficiency and give an probabilistic output. This is essentially equivalent to a GP with WL graph kernel but uses a linear base kernel within the WL graph kernel computation. Please refer to Section 7.3.1 for more detailed explanation of the WL mechanism underlying the WL feature extractor. After running feature extraction with H WL iterations (i.e. a WL feature extractor with its hyper-parameter $h = H$) for each training graph and concatenating their corresponding feature vectors, we obtain a feature matrix $\boldsymbol{\Phi} = [\phi(\mathcal{G}'_1), \ldots, \phi(\mathcal{G}'_n)]^\top$ to be passed to the Bayesian linear regressor.

For large graphs, the resulting extracted feature vector will likely be very high-dimensional, which would lead to high-variance regression coefficients $\boldsymbol{\alpha}$ being estimated if the number of input samples is comparatively few. To attain a good predictive performance in such a case, we employ the automatic relevance determination (ARD) prior for Bayesian linear regression surrogate to regularise weights and encourage sparsity in $\boldsymbol{\alpha}$ (Wipf et al., 2007):

$$y_{\text{attack}}|\boldsymbol{\Phi}, \boldsymbol{\alpha}, \sigma^2_{noise} \sim \mathcal{N}(\boldsymbol{\alpha}^\top \boldsymbol{\Phi}, \sigma^2_{noise}\mathbf{I}), \tag{6.11}$$

$$\boldsymbol{\alpha}|\boldsymbol{\lambda} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}),\ \text{diag}(\boldsymbol{\Lambda}) = \boldsymbol{\lambda}^{-1} = \{\lambda_1^{-1}, ..., \lambda_{\dim(\boldsymbol{\lambda})}^{-1}\}, \tag{6.12}$$

$$\lambda_i \sim \text{Gamma}(k, \theta)\ \forall i \in [1, \dim(\boldsymbol{\lambda})], \tag{6.13}$$

where $\boldsymbol{\Lambda}$ is the diagonal covariance matrix. $\boldsymbol{\alpha}$ and the noise variance σ^2_{noise} are learnt by maximising the model marginal likelihood. Overall, the WL routines scales as $\mathcal{O}(Hm)$ (Shervashidze et al., 2011), whereas training of Bayesian linear regression has a linear scaling with respect to the number of queries to the victim model; these ensure the surrogate is scalable to both larger graphs and/or a large number of graphs, both of which are commonly encountered in graph classification attacks.

6.6.3 Sequential Perturbation Selection

In the default structural perturbation setting, given an attack budget of Δ (i.e. we are allowed to flip up to Δ edges from $\mathcal{G}$), finding exactly the set of perturbations $\delta\mathbf{A}$ that leads to the largest increase in y_{attack} entails an combinatorial optimisation

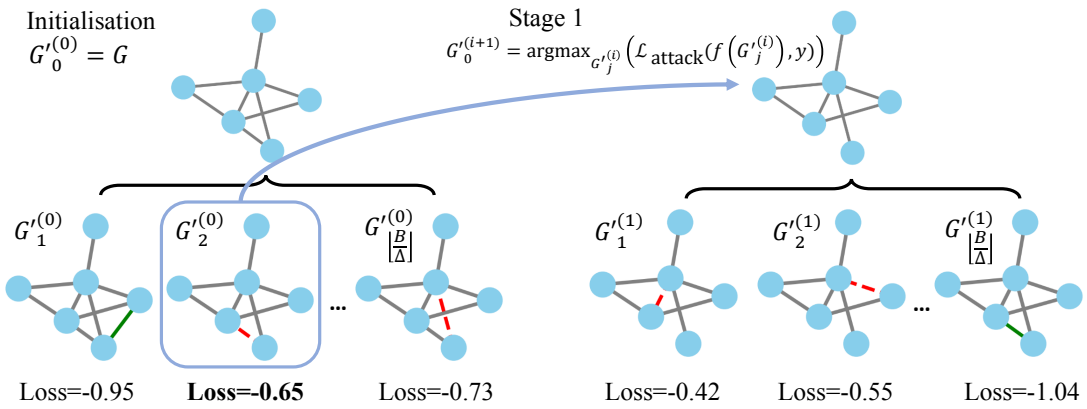


Figure 6.5: Sequential edge selection. At each stage the BO agent sequentially proposes candidate graphs with edge edit distance of 1 from the base graph $\mathcal{G}'^{(i)}_0$ (which is the original unperturbed graph $\mathcal{G}$ at initialisation, or a perturbed graph that led to the largest increase in loss from the previous stage otherwise). This procedure repeats until either the attack succeeds (i.e. we find a graph $\mathcal{G}'$ with $y_{\text{attack}}(\mathcal{G}') > 0$) or the maximum number of B queries to the victim model f is exhausted.

over $\binom{n^2}{\Delta}$ candidates. This is a huge search space that is difficult for the surrogate to learn meaningful patterns in a sample-efficient way even for modestly-sized graphs. To tackle this challenge, we adopt the strategy illustrated in Figure 6.5: given the query budget B (i.e. the total number of times we are allowed to query f for a given $\mathcal{G}$), we amortise B into Δ stages and focus on selecting *one* edge perturbation at each stage. While this strategy is greedy in the sense that it always commits the perturbation leading to the largest increase in loss at each stage, it worths noting that we *do not* treat the previously modified edges differently, and the agent can, and does occasionally as we observed, correct previous modifications by flipping edges back. This is possible due to the edge selection being permutation invariant. Another benefit of this strategy is that it can potentially make full use of the entire attack budget Δ *while* remaining parsimonious with respect to the amount of perturbation introduced, as it only progresses to the next stage and modifies the $\mathcal{G}$ further when it fails to find a successful adversarial example in the current stage.

In addition, the discrete graph nature of the input space makes it impossible to use gradient-based methods for optimising the acquisition function in BO. Thus, we optimise the acquisition function using genetic algorithm (GA) similar to what we

do for the neural architecture search task in Section 7.3.2. Specifically, we generate the initial population of 100 candidates by mutating from the top-3 perturbed graphs that previously led to the largest attack loss; if we have not yet queried any graphs, we simply mutate the base graph. Each mutation corresponds to flipping the connection [7] between one pair of randomly selected end nodes from the base graph. We then evolve the population 10 times, with each evolution cycle involving mutating the current population to generate offspring, popping the oldest members in the population and evaluating the acquisition function value of the population. Finally, we select the top 5 unique candidates seen during the evolution process that have the highest acquisition function value (EI in our case) to query the victim model.

6.6.4 Experiments

Table 6.5: Key statistics of the TU datasets used.

Dataset	N_{graphs}	N_{labels}	Avg. number of nodes	Avg. number of edges
IMDB-M	1500	3	13.0	65.9
PROTEINS	1113	2	39.1	72.8
COLLAB	5000	3	74.5	2457.8

Table 6.6: Validation accuracy of the victim models on the TU datasets before (clean) and after various attack methods. Results shown in mean ± 1 standard deviation across 3 trials.

Victim Models	GCN (Kipf and Welling, 2017)			GIN(Xu et al., 2019a)		
Datasets	IMDB-M	PROTEINS	COLLAB	IMDB-M	PROTEINS	COLLAB
Clean	$50.53_{\pm 1.4}$	$71.73_{\pm 2.6}$	$79.73_{\pm 2.1}$	$48.85_{\pm 0.4}$	$70.53_{\pm 2.3}$	$80.80_{\pm 0.9}$
Random	$47.43_{\pm 1.2}$	$19.46_{\pm 1.7}$	$76.41_{\pm 6.2}$	$40.44_{\pm 2.5}$	$42.90_{\pm 2.2}$	$71.29_{\pm 0.9}$
Genetic (Dai et al., 2018)	$47.82_{\pm 1.5}$	$14.88_{\pm 1.7}$	$58.61_{\pm 7.9}$	$39.68_{\pm 3.1}$	$23.25_{\pm 5.3}$	$61.68_{\pm 2.5}$
Gradient-based†	$\mathbf{39.31_{\pm 2.2}}$	$50.60_{\pm 4.5}$	$36.67_{\pm 1.2}$	$\mathbf{37.56_{\pm 2.2}}$	$11.90_{\pm 4.4}$	$\mathbf{54.00_{\pm 2.9}}$
GRABNEL (ours)	$45.23_{\pm 0.2}$	$\mathbf{10.82_{\pm 2.5}}$	$\mathbf{35.38_{\pm 9.3}}$	$38.22_{\pm 3.9}$	$\mathbf{10.72_{\pm 7.1}}$	$57.33_{\pm 4.7}$

†: White-box method

[7] If an edge already exists, remove it; otherwise, add an edge.

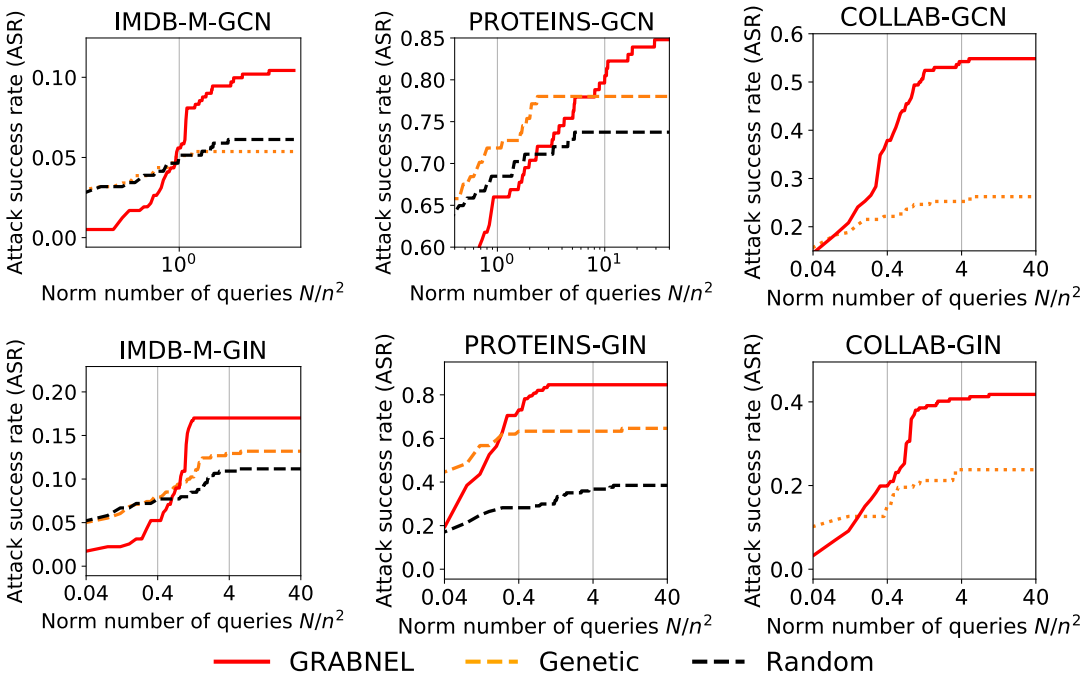


Figure 6.6: Attack success rate (ASR) against the number of queries to the victim models (normalised by the square of the number of nodes of each graph) in various datasets on GCN and GIN models. Note the x-axis is on log-scale. Lines and shades denote mean ±1 sd across 3 random initialisations. It is evident that `grabnel` outperforms other attack methods considerably. `Random` and `Genetic` appear to converge faster in some tasks as they always use exploit full perturbation budget allocated, while `grabnel` only attempts a higher perturbation budget when attack fails in a lower one.

We validate the performance of our proposed attack method, GRABNEL, on three common TU datasets (Morris et al., 2020), namely (in ascending order of average graph sizes in the dataset) IMDB-M, PROTEINS and COLLAB[8]. The key statistics of these TU datasets are summarised in Table 6.5. In all cases, we define the perturbation budget Δ in terms of the maximum *structural perturbation ratio* r defined in Chen et al. (2021b) where $\Delta \leq rN^2$. We similarly link the maximum numbers of queries B allowed for individual graphs to their sizes as $B = 50\Delta$, thereby giving larger graphs and thus potentially more difficult instances higher attack[9]. In the following experiments, unless otherwise specified, we set $r = 0.03$. For our GRABNEL, we set $H = 1$ for the WL feature extractor.

[8]All TU datasets may be downloaded at https://chrsmrrs.github.io/datasets/docs/datasets/.

[9]Due to computational constraints, we cap the maximum number of queries to be 2×10^4 on each graph.

For comparison we consider a number of competitive baselines, including random search, genetic algorithm originally introduced in Dai et al. (2018) and an additional simple gradient-based method which greedily adds or delete edges based on the magnitude computed input gradient similar to the gradient based method described in Dai et al. (2018) (note that this method is *white-box* as access to parameter weights and gradients is required). To verify whether the proposed attack method can be used for a variety of classifier architectures we also consider two widely used victim models, namely the Graph Convolutional Network (GCN) (Kipf and Welling, 2017) and the Graph Isomorphism Network (GIN) (Xu et al., 2019a).

We show the classification performance of both victim models before and after attacks using various methods in Table 6.6, and we show the attack success rate (ASR) against the (normalised) number of queries in Figure 6.6. It is worth noting that in consistency with the image attack setting in Section 6.5, we launch and consider attacks on the *graphs that were originally classified correctly*, and compute statistics, such as the ASR, in the same fashion.

The results generally show that the attack method is effective against both GCN and GIN models with GRABNEL typically leading to the largest degradation in victim predictions in all tasks, often performing on par or better than Gradient, a white-box method. Further, GRABNEL typically outperforms by a larger extent for the larger graphs (e.g. COLLAB) on which the benefit of the sequential selection of edge perturbation is more significant.

As discussed, in real life, adversarial agents might encounter additional constraints other than the number of queries to the victim model or the amount of perturbation introduced. To demonstrate that our framework can handle such constraints, we further carry out attacks on victim models using identical protocols as above but with a variety of additional constraints considered in several previous works. Specifically, the scenarios considered, in the ascending order of restrictiveness, are:

- *Base*: The base scenario is identical to the setup in Table 6.6 and Figure 6.6;

- *2-hop*: Edge addition between nodes (u, v) is only permitted if v is within 2-hop distance of u;
- *2-hop+rewire* (Ma et al., 2019b): Instead of flipping edges, the adversarial agent is only allowed to rewire from nodes (u, v) (where an edge exists) to (u, w) (where no edge currently exists). Node w must be within 2-hop distance of u;

We test on the PROTEINS dataset, and show the results in Figure 6.7, where it is evident that while additional constraints unsurprisingly lead to slightly lower attack success rates, the performance of GRABNEL remains relatively robust in all scenarios considered. In fact, as we elaborate in Section 6.6.5, we find the phenomenon of attacked edges remaining relatively clustered within a relatively small neighbourhood is a general pattern in adversarial examples of many tasks. This implies that the 2-hop condition, which constrains the spatial relations of the adversarial edges, might already hold even without explicit specification, thereby explaining the marginal difference between the base and the 2-hop constrained cases in Figure 6.7.

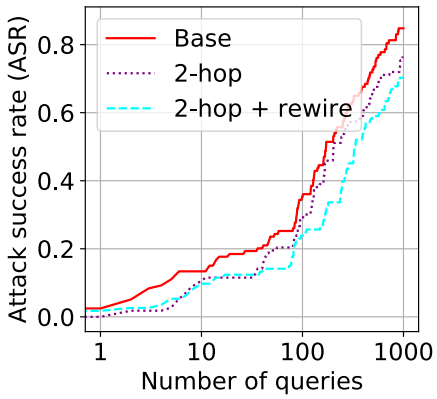
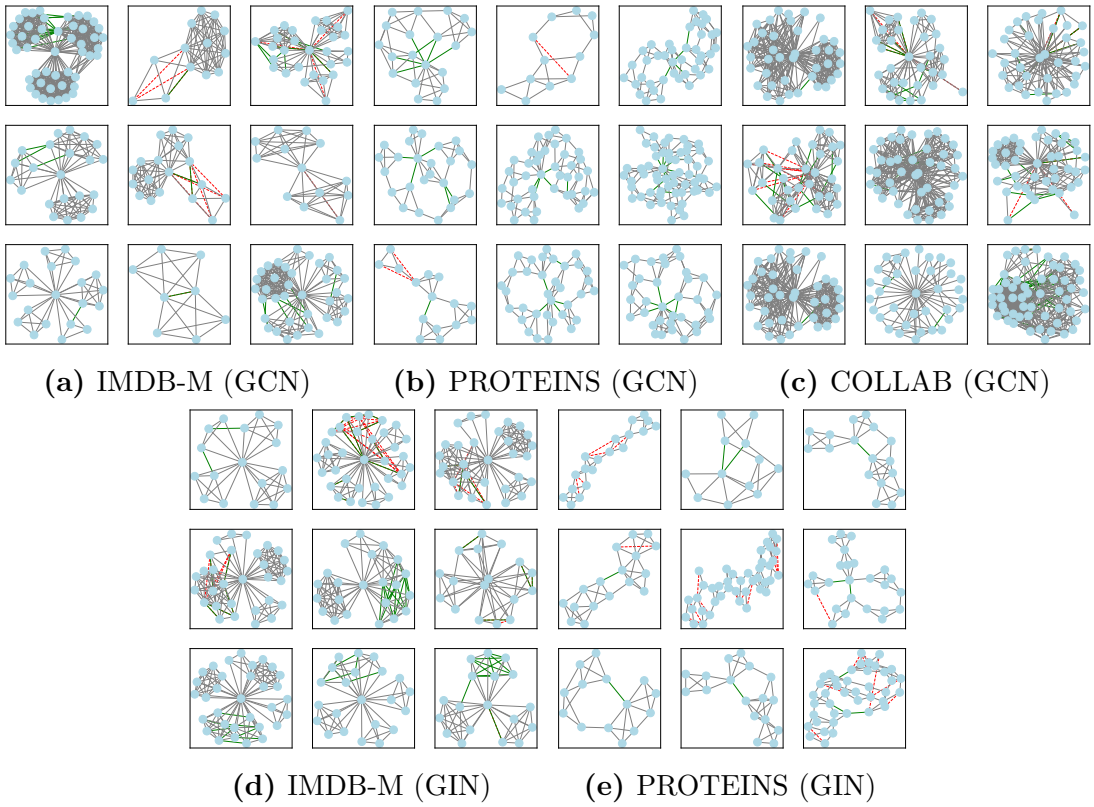


Figure 6.7: ASR of GRABNEL against number of queries on a GCN model trained on PROTEINS dataset under various attack constraints.

6.6.5 Atack Analysis

Having demonstrated the effectiveness of our method, in this section we provide a qualitative analysis on the common interpretable patterns behind the adversarial



(a) IMDB-M (GCN) **(b)** PROTEINS (GCN) **(c)** COLLAB (GCN)

(d) IMDB-M (GIN) **(e)** PROTEINS (GIN)

Figure 6.8: Adversarial examples found by the proposed method. Red edges denote deleted edges from the original samples and green edges indicate those added. In Twitter fake news detection task, green nodes/edges denote the injected nodes and their connections to the existing graphs.

samples found, which provides further insights into the robustness of graph classification models against structural attacks. We believe such analysis is especially valuable, as it may facilitate the development of even more effective attack methods, and may provide insights that could be useful for identification of real-life vulnerabilities for more effective defence. We show some examples of the adversarial samples in Figure 6.8, and we summarise some key findings below.

1. *Adversarial edges tend to cluster closely together*: A common observation across many datasets and models is that the distribution of the adversarial edges (either removal or addition) in a graph is highly uneven, with many adversarial edges often sharing common end-nodes or having small spatial distance to each other. This is empirically consistent with recent theoretical findings on the stability of spectral graph filters in Kenlay et al. (2021).

From an attacker point of view, this may provide a "prior" on the attack to constrain the search space, as the regions around existing perturbations should be exploited more.

2. *Adversarial edges often attempt to destroy or modify community structures*: for example, the original graphs in the IMDB-M dataset can be seen to have community structure. When the GCN model is attacked, the attack tends to flip the edges *between* the communities, and thereby destroying the structure by either merging communities or deleting edges within a cluster. On the other hand, the GIN examples tend to strengthen the community structures by adding edges within clusters and deleting edges between them. With similar observations also present in, for example, PROTEINS dataset, this may suggest that the models may be fragile to modification of the community structure.

6.7 Conclusion

We introduce a new black-box adversarial attack which leverages BO to find successful adversarial perturbations with high query efficiency. We also improve our attack by adopting an additive surrogate structure to ease the optimisation challenge over the typically high-dimensional task. Moreover, we take full advantage of our statistical surrogate model and the available query data to learn the optimal degree of dimension reduction for the search space via Bayesian model selection. In comparison to several existing black-box attack methods, our BayesOpt attacks can achieve high success rates with 2-5 times fewer queries while still producing adversarial examples that are closer to the original image (in terms of average L_2 distance). One limitation of our attack methods is that due to the poor scalability of GPs, the attack algorithms are computationally more expensive than most existing alternatives especially as the query number increases. Thus, our BayesOpt attacks are optimal for the setting where the cost of evaluating the victim model, being it the monetary costs, computational costs or the risk of being detected,

is much higher than the computational cost of the attack algorithm itself and thus query efficiency is highly prioritised.

Beyond image classification attacks, we also extend the use of BO for attacking graph-based machine learning models under the black-box setting. Our graph attack method, GRABNEL, leverages the WL feature extraction to handle graph inputs and uses a Bayesian linear regressor as the surrogate model to avoid the above-mentioned scalability issue. GRABNEL achieves superior query efficiency and attack success rate on a variety of graph classification tasks.

Therefore, our works demonstrate that BO-based adversarial attacks can be a highly promising class of approaches for efficiently evaluating the adversarial robustness of machine learning models.

Future directions With the initial successes achieved by our BO-based attacks, one immediate extension would be to leverage the rich literature of transfer-learning/multi-tasking BO techniques to further enhance the query efficiency in performing black-box adversarial attacks. Moreover, given the high-dimensionality of the adversarial attack problems, other useful techniques such as local trust region used in Section 5.6 or more advanced input encodings (e.g. VAEs with deep metric learning (Grosnit et al., 2021)) can be incorporated into our BO attack frameworks to reduce the search complexity for adversarial examples. Finally, Bayesian neural networks, which are highly expressive yet scalable, can be investigated as an alternative surrogate option to the GP and Bayesian linear regressor used in our works.

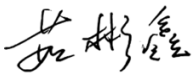
Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Attacking graph classification via Bayesian optimisation
Publication Status	□Published □ Accepted for Publication ☒Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Xingchen Wan, Henry Kenlay, Binxin Ru, Arno Blaas, Michael A. Osborne, and Xiaowen Dong. Attacking graph classification via Bayesian optimisation. Under review at *Advances in Neural Information Processing Systems (NeurIPS 2021),* 2021.

Student Confirmation

Student Name:	Binxin Ru		
Contribution to the Paper	I contributed to the discussions of the problem motivation and the surrogate model for the BO attack, which is heavily related to another joint-work of Wan and I. I join the discussion on result analyses and experiments and helped with the final write-up.		
Signature		Date	17/09/2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof. Michael A. Osborne		
Supervisor comments To the best of my knowledge, all above seems fair and correct.		
Signature 	Date	17/09/2021

This completed form should be included in the thesis, at the end of the relevant chapter.

7

Bayesian Optimisation for Neural Architecture Search

The content of this chapter is based on the following paper:

> Binxin Ru*, Xingchen Wan*, Xiaowen Dong, and Michael A. Osborne. Interpretable neural architecture search via Bayesian optimisation with Weisfeiler-Lehman kernels. In *International Conference on Learning Representations (ICLR 2021)*, 2021.

where our contributions were listed in the acknowledgements section of this thesis.

Neural architecture search (NAS) is an important AutoML direction which aims to automate the design of good neural network architectures for a given task and dataset. Similar to hyper-parameter optimisation, NAS can also be formulated as a black-box optimisation problem (Elsken et al., 2018) and each evaluation of the objective is expensive as it involves training of the architecture queried. Therefore, query efficiency is highly valued (Kandasamy et al., 2018b; Elsken et al., 2018; Shi et al., 2019) and Bayesian optimisation (BO), which has been successfully applied for hyper-parameter optimisation as shown in Chapter 3 to 5, becomes a natural choice to consider. However, conventional BO methods cannot be applied directly to NAS as the search spaces are non-continuous and graph-like (Elsken

et al., 2018; Zoph et al., 2018a; Ying et al., 2019; Dong and Yang, 2020b); each possible architecture can be represented as an acyclic directed graph with node or edge attributes denoting the operation units/layers. In this chapter, we propose a BO approach for NAS that combines the Weisfeiler-Lehman graph kernel with a Gaussian process (GP) surrogate. Our method optimises the architecture in a highly data-efficient manner: it is capable of capturing the topological structures of the architectures and is scalable to large graphs, thus making the high-dimensional and graph-like search spaces amenable to BO.

More importantly, our method affords *interpretability* by discovering useful network features and their corresponding impact on the network performance. Indeed, we demonstrate empirically that our surrogate model is capable of identifying useful motifs which can guide the generation of new architectures. This marks the first attempt towards *interpretable* NAS and is in contrast to current NAS strategies, which only return a final good architecture without offering any insight into why a specific architecture is good, or how we should modify the architecture if we want further improvements. Through extensive empirical results, we show that our GP surrogate model achieves superior prediction and is scalable to large architectures; competing methods require 3 to 20 times more queries to be on par with ours. And our BO strategy can generalise to a diverse set of search spaces and outperforms existing NAS approaches to achieve the state of the art.

7.1 Introduction

Although different NAS strategies have led to state-of-the-art neural architectures, outperforming human experts' design on a variety of tasks (Real et al., 2017b; Zoph and Le, 2017; Cai et al., 2018a; Liu et al., 2018b,a; Luo et al., 2018; Pham et al., 2018; Real et al., 2018; Zoph et al., 2018a; Xie et al., 2018), these strategies behave in a black-box fashion, which returns little design insight except for the final architecture for deployment. In this chapter, we introduce the idea of *interpretable* NAS, extending the learning scope from simply the optimal architecture to interpretable features. These features can help explain the performance of networks searched

and guide future architecture design. We make the first attempt at interpretable NAS by proposing a new NAS method, NAS-BOWL; our method combines a GP surrogate with the Weisfeiler-Lehman (WL) subtree graph kernel (we term this surrogate GPWL) and applies it within the BO framework to efficiently query the search space. During search, we harness the interpretable architecture features extracted by the WL kernel and learn their corresponding effects on the network performance based on the surrogate gradient information.

Besides offering a new perspective on interpratability, our method also improves over the existing BO-based NAS approaches. To accommodate the popular cell-based search spaces, which are non-continuous and graph-like (Zoph et al., 2018a; Ying et al., 2019; Dong and Yang, 2020b), current approaches either rely on encoding schemes (Ying et al., 2019; White et al., 2021a) or manually designed similarity metrics (Kandasamy et al., 2018b), both of which are not scalable to large architectures and ignore the important topological structure of architectures. Another line of work employs graph neural networks (GNNs) to construct the BO surrogate (Ma et al., 2019a; Zhang et al., 2019; Shi et al., 2019); however, the GNN design introduces additional hyper-parameter tuning, and the training of the GNN also requires a large amount of architecture data, which is particularly expensive to obtain in NAS. Our method, instead, uses the WL graph kernel to naturally handle the graph-like search spaces and capture the topological structure of architectures. Meanwhile, our surrogate preserves the merits of GPs in data-efficiency, uncertainty computation and automated hyper-parameter treatment. In summary, our main contributions are as follows:

- We introduce a GP-based BO strategy for NAS, NAS-BOWL, which is highly query-efficient and amenable to the graph-like NAS search spaces. Our proposed surrogate model combines a GP with the WL graph kernel (GPWL) to exploit the implicit topological structure of architectures. It is scalable to large architecture cells (e.g. 32 nodes) and can achieve better prediction performance than competing methods.

- We propose the idea of *interpretable NAS* based on the graph features extracted by the WL kernel and their corresponding surrogate derivatives. We show that interpretability helps in explaining the performance of the searched neural architectures. As a singular example of concrete application, we propose a simple yet effective motif-based transfer learning baseline to warm-start search on a new image tasks.

- We demonstrate that our surrogate model achieves superior performance with much fewer observations in search spaces of different sizes, and that our strategy both achieves state-of-the-art performances on both NAS-Bench datasets and open-domain experiments while being much more efficient than comparable methods.

7.2 Preliminaries

Graph Representation of Neural Networks

Architectures in popular NAS search spaces can be represented as an acyclic directed graph (Elsken et al., 2018; Zoph et al., 2018b; Ying et al., 2019; Dong and Yang, 2020b; Xie et al., 2019a), where each graph node represents an operation unit (e.g. a `conv3×3-bn-relu` [1] in Ying et al. (2019)) and each edge defines the information flow from one layer to another. To reduce the search complexity in NAS, a popular practice is to search for the repeated motifs/cells in a neural network instead of the whole architecture (Zoph et al., 2018b). The architecture is then formed by stacking a number of searched cells in a predefined way. In such cell-based search space, an architecture cell is an acyclic directed graph as shown in Figure 7.1.

With this representation, NAS can be formulated as an optimisation problem to find the directed graph and its corresponding node operations (i.e. the directed attributed graph G) that give the best architecture validation performance $y(G)$:

$$G^* = \arg\max_{G \in \mathcal{G}} y(G) \tag{7.1}$$

[1] A sequence of operations: convolution with 3×3 filter size, followed by batch normalisation and ReLU activation.

Bayesian Optimisation

To solve the above optimisation, we adopt BO for optimising a black-box, expensive-to-evaluate objective (Brochu et al., 2010b). Please refer to Chapter 2 for a detailed discussion on BO. We use a GP as the surrogate model in this work, as it can achieve competitive modelling performance with small amount of query data (Williams and Rasmussen, 2006) and give analytic predictive posterior mean $\mu(G_t|\mathcal{D}_{t-1})$ and variance $\sigma(G_t, G_t'|\mathcal{D}_{t-1})^2$ on the heretofore unseen graph G_t given $t-1$ observations:

$$\mu(G_t|\mathcal{D}_{t-1}) = K(G_t, G_{1:t-1})\mathbf{K}_{1:t-1}^{-1}\mathbf{y}_{1:t-1} \tag{7.2}$$

$$\sigma(G_t, G_t'|\mathcal{D}_{t-1})^2 = K(G_t, G_t') - K(G_t, G_{1:t-1})\mathbf{K}_{1:t-1}^{-1}K(G_{1:t-1}, G_t') \tag{7.3}$$

where $\mathcal{D}_{t-1} = \{(G_i, y_i)\}_i^{t-1} = \{G_{1:t-1}, \mathbf{y}_{1:t-1}\}$ are the $t-1$ observed graphs and their corresponding objective function values. $[\mathbf{K}_{1:t-1}]_{i,j} = k(G_i, G_j)$ is the (i,j)-th element of Gram matrix induced on the (i,j)-th training samples by $k(\cdot,\cdot)$, the graph kernel function. For acquisition function, we use EI (Mockus et al., 1978) in this work though our approach is compatible with alternative choices.

Graph Kernels

Graph kernels are kernel functions defined over graphs to compute their level of similarity. A generic graph kernel may be represented by the function $k(\cdot,\cdot)$ over a pair of graphs G and G' (Kriege et al., 2020):

$$k(G, G') = \langle \boldsymbol{\phi}(G), \boldsymbol{\phi}(G') \rangle_{\mathcal{H}} \tag{7.4}$$

where $\boldsymbol{\phi}(\cdot)$ is some feature representation of the graph extracted by the graph kernel and $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ denotes inner product in the associated reproducing kernel Hilbert space (RKHS) (Nikolentzos et al., 2019; Kriege et al., 2020). For example, the popular class of $\mathcal{R}$-Convolution graph kernels (Haussler, 1999) recursively decomposes the graph G into smaller constituting substructures, and $\phi(G)$ corresponds to the counts of these substructures in the graph. For more detailed reviews on graph kernels, the readers are referred to (Nikolentzos et al., 2019), (Ghosh et al., 2018) and (Kriege et al., 2020).

Algorithm 11 NAS-BOWL Algorithm. Optional steps of the exemplary use of motif-based warm starting (Section 7.3.3) are marked in *blue italics*.

Input: Maximum BO iterations T, BO batch size b, acquisition function $\alpha(\cdot)$, initial observed data on the target task $\mathcal{D}_0$, Optional: past-task query data $\mathcal{D}_{\text{past}}$ and surrogate $\mathcal{S}_{\text{past}}$
Output: The best architecture G_T^*
Initialise the GPWL surrogate $\mathcal{S}$ with $\mathcal{D}_0$
for $t = 1, \ldots, T$ **do**
  if *Pruning based on the past-task motifs* **then**
    Compute the motif importance scores based on Equation (7.8) *with* $\mathcal{S}_{\text{past}}/\mathcal{S}$ *on* $\mathcal{D}_{\text{past}}/\mathcal{D}_t$
    while $|\mathcal{G}_t| < B$ **do**
      Generate a batch of candidate architectures and reject those which contain none of the top 25% good motifs (similar procedure as Figure 7.2(a))
    end while
  else
    Generate B candidate architectures $\mathcal{G}_t$
  end if
  $\{G_{t,i}\}_{i=1}^{b} = \arg\max_{G \in \mathcal{G}_t} \alpha_t(G|\mathcal{D}_{t-1})$
  Evaluate their validation accuracy $\{y_{t,i}\}_{i=1}^{b}$
  $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup (\{G_{t,i}\}_{i=1}^{B}, \{y_{t,i}\}_{i=1}^{b})$
  Update the surrogate $\mathcal{S}$ with $\mathcal{D}_t$
end for
Return the best architecture seen so far G_T^*

7.3 Bayesian Optimisation with Weisfeiler-Lehman Graph Kernel

We begin by presenting our proposed approach, NAS-BOWL in Algorithm 11 and will discuss its three key design components in the following subsections. Specifically, we explain the GP surrogate design for handling graph inputs in architecture search (we term the surrogate GPWL) in Section 7.3.1 and go through the methods for optimising the acquisition function over architectures in Section 7.3.2. Finally, we elaborate the technique for learning the impact of interpretable motifs on architecture performance in Section 7.3.3 and demonstrate the new possibilities opened by our work followed by an exemplary practical use of interpretable motifs for transfer learning.

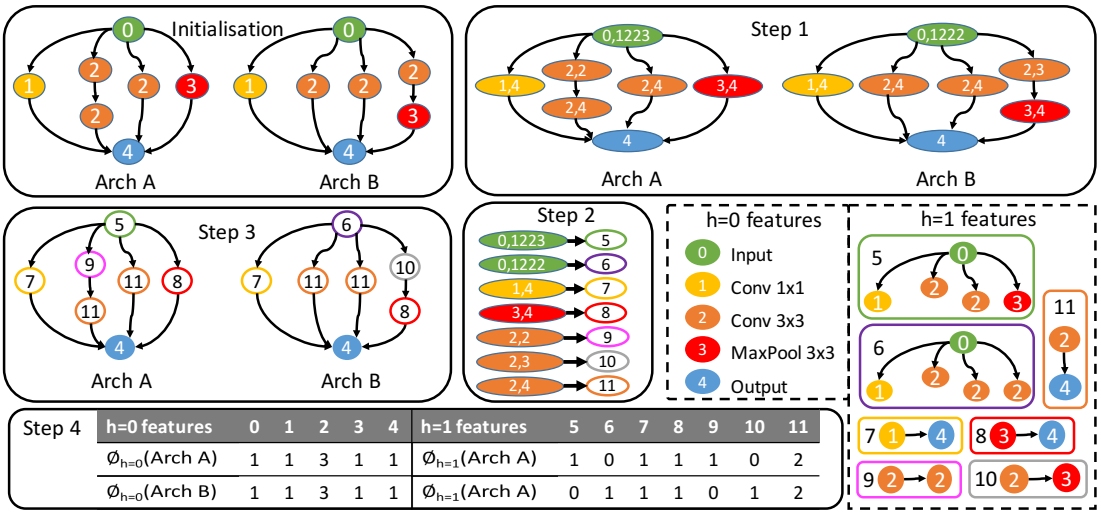


Figure 7.1: Illustration of one WL iteration. Given two architecture cells at initialisation, WL kernel first collects the neighbourhood labels of each node (Step 1) and compress the collected $h = 0$ labels into $h = 1$ features (Step 2). Each node is then relabelled with $h = 1$ features (Step 3) and the two graphs are compared based on the histogram on both $h = 0$ and $h = 1$ features (Step 4). This WL iteration will be repeated until $h = H$. h is both the index the WL iteration and the depth of the subtree features extracted. Substructures at $h = 0$ and $h = 1$ in Arch A are shown in the bottom right of the plot.

7.3.1 Surrogate Design and Graph Kernel

To enable the GP to work effectively on the graph-like architecture search space, we need to decide on a suitable kernel function. Here we propose to use the Weisfeiler-Lehman (WL) graph kernel (Shervashidze et al., 2011) to enable the direct definition of a GP surrogate on the graph-like search space. The WL kernel compares two directed graphs based on both local and global structures. It starts by comparing the node labels of both graphs via a base kernel $k_{\text{base}}\big(\boldsymbol{\phi}_0(G), \boldsymbol{\phi}_0(G')\big)$ where $\boldsymbol{\phi}_0(G)$ denotes the histogram of features at level $h = 0$ (i.e. node features) in the graph, where h is both the index of WL iterations and the depth of the subtree features extracted. For the WL kernel with $h > 0$, as shown in Figure 7.1, it then proceeds to collect features at $h = 1$ by aggregating neighbourhood labels, and compare the two graphs with $k_{\text{base}}\big(\boldsymbol{\phi}_1(G), \boldsymbol{\phi}_1(G')\big)$ based on the subtree structures of depth 1 (Shervashidze et al., 2011; Höppner and Jahnke, 2020). The procedure then repeats until the highest iteration level $h = H$ specified and the

resulting WL kernel is given by:

$$k_{\text{WL}}^{H}(G, G') = \sum_{h=0}^{H} k_{\text{base}}\Big(\boldsymbol{\phi}_h(G), \boldsymbol{\phi}_h(G')\Big). \tag{7.5}$$

In the above equation, k_{base} is a base kernel over the vector feature embedding (e.g. a dot product $\boldsymbol{\phi}(G) \cdot \boldsymbol{\phi}(G')$). As h increases, the WL kernel captures higher-order features which correspond to increasingly larger neighbourhoods. Features at each h are concatenated to form the the final feature vector ($\boldsymbol{\phi}(G) = [\boldsymbol{\phi}_0(G), ..., \boldsymbol{\phi}_H(G)]$). The algorithmic description of the WL procedure is shown in Algorithm 18 in Appendix app10sec:sumkernelgpwl.

The WL kernel is a desirable choice for the NAS application for the following reasons:

1. **Applicable to labeled and directed graphs.** As discussed in Section 7.2, architectures in almost all popular NAS search spaces (Ying et al., 2019; Dong and Yang, 2020b; Zoph et al., 2018b; Xie et al., 2019a) can be represented as directed graphs with node/edge attributes. Thus, WL kernel can be directly applied on them. On the other hand, many graph kernels either do not handle node labels such as graphlet kernel (Shervashidze et al., 2009), or are incompatible with directed graphs such as those based on eigenvalues of the Laplacian (Kondor and Pan, 2016; de Lara and Pineau, 2018); the eigenvalues are required to be real, which implies a symmetric adjacency matrix hence an undirected graph. Converting architectures into undirected graphs can result in loss of valuable information such as the direction of data flow in the architecture (Shown in Section 7.5.1).

2. **Expressive yet highly interpretable.** WL kernel is able to capture substructures that go from local to global scale with increasing h values. Such multi-scale comparison is similar to that enabled by a Multiscale Laplacian Kernel (Kondor and Pan, 2016) and is desirable for architecture comparison. This is in contrast to graph kernels such as (Kashima et al., 2003; Shervashidze et al., 2009), which only focus on local substructures, or those based on graph spectra (de Lara and Pineau, 2018), which only look at global connectivities. Furthermore, the

WL kernel is derived directly from the Weisfeiler-Lehman graph isomorphism test (Weisfeiler and Lehman, 1968), which is shown to be as powerful as a GNN in distinguishing non-isomorphic graphs (Morris et al., 2019; Xu et al., 2018). However, the higher-order graph features extracted by GNNs are hard to interpret by humans. On the other hand, the subtree features learnt by WL kernel (e.g. the $h = 0$ and $h = 1$ features in Figure 7.1) are easily interpretable. As we will discuss in Section 7.3.3 later, we can harness the surrogate gradient information on low-h substructures to identify the effect of particular node labels on the architecture performance and thus learn useful information to guide new architecture generation.

3. **Relatively efficient and scalable.** Other expressive graph kernels are often prohibitive to compute: for example, defining $\{n, m\}$ to be the number of nodes and edges in a graph, random walk (Gärtner et al., 2003) and shortest path (Borgwardt and Kriegel, 2005) incur a complexity of $\mathcal{O}(n^3)$ and $\mathcal{O}(n^4)$. Another approach based on computing the architecture edit-distance (Jin et al., 2019) is also expensive: its exact solution is NP-complete (Zeng et al., 2009) and is provably difficult to approximate (Lin, 1994). On the other hand, the WL kernel only entails a complexity[2] of $\mathcal{O}(hm)$ (Shervashidze et al., 2011). Empirically, we find that in typical NAS search spaces (such as NAS-Bench datasets) featuring rather small cells, $h \leq 3$ usually suffices; this implies the kernel computing cost is likely eclipsed by the $\mathcal{O}(N^3)$ complexity of GPs, not to mention the main bottleneck of NAS is the actual training of the architectures. The scalability of WL is also to be contrasted to other approaches such as path encoding (White et al., 2021a), which without truncation scales exponentially with n.

4. **WL kernel gives positive semidefinite covariance matrix.** Given any positive semidefinite base kernel, the resultant WL kernel is also positive semidefinite, which is a prerequisite to be a valid GP covariance function. This is not enjoyed by graph kernels in general: Some graph kernels, including graph edit distance and the general optimal assignment (OA) kernel itself, are *not* generally positive

[2]Naively computing the Gram matrix consisting of pairwise kernel between all pairs in N graphs is of $\mathcal{O}(N^2hm)$, but this can be further improved to $\mathcal{O}(Nhm + N^2hn)$ (Morris et al., 2019).

semidefinite (Vert, 2008) (although the OA kernel applied on WL is indeed positive semidefinite (Kriege et al., 2016)).

With the above-mentioned merits, the incorporation of the WL kernel permits the usage of GP-based BO on various NAS search spaces. This enables the practitioners to harness the rich literatures of GP-based BO methods on hyper-parameter optimisation and redeploy them on NAS problems. Most prominently, the use of GP surrogate frees us from hand-picking the WL hyper-parameter H as we can automatically learn the optimal values by maximising the Bayesian marginal likelihood. As we will justify in Section 7.5.1, this process is extremely effective. This renders a further major advantage of our method as it has no inherent hyper-parameters that require manual tuning. This reaffirms with our belief that a practical NAS method itself should require minimum tuning, as it is almost impossible to run traditional hyper-parameter search given the vast resources required. We empirically justify the superior prediction performance of our GP surrogate with a WL kernel in Section 7.5.1. Other enhancements, such as improving the expressiveness of the surrogate by combining multiple types of kernels, are briefly investigated in Appendix B.2. We find the amount of performance gain depends on the NAS search space and a WL kernel alone suffices for common cell-based spaces.

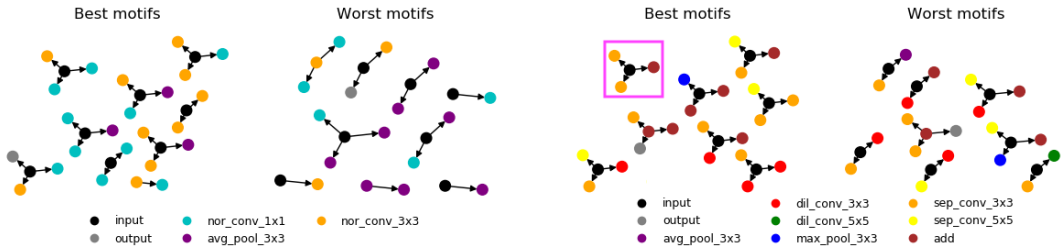
7.3.2 Acquisition Function Optimisation over Graph Inputs

Under the NAS setting, optimising the acquisition function over the search space can be challenging (Kandasamy et al., 2018b; Ma et al., 2019a; White et al., 2021a), because the non-continuous search space makes it ineffective to use the analytic gradients, and exhaustively evaluating on all possible architectures is computationally unviable. A way to generate a population of candidate architectures for acquisition function optimisation at each BO iteration is necessary for all BO-based NAS strategies.

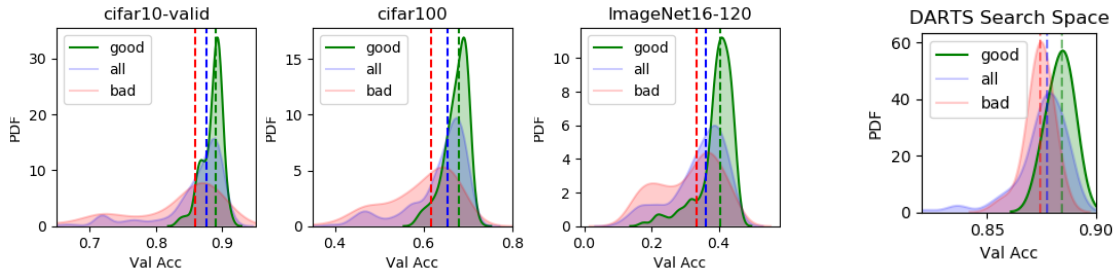
The naïve way to do so is to randomly sample architectures from the search space (Ying et al., 2019; Yang et al., 2020). Specifically, we randomly generate a valid adjacency matrix and a list of operation types assigned to each node. This is

simple to implement but ignores any information contained in past query data, thus inefficient in exploring the huge search space. A more popular approach is based on genetic mutation which generates the candidate architectures by mutating a small pool of parent architectures (Kandasamy et al., 2018b; Ma et al., 2019a; White et al., 2021a; Shi et al., 2019). The parent architectures are usually chosen among queried architectures which give the top validation performance or acquisition function values. Generating candidate architectures in this way enables us to exploit the prior information on the best architectures observed so far to explore the large search space more efficiently. We report NAS-BOWL with both random sampling (NASBOWLr) and mutation algorithm (NASBOWm) in our experiments in Section 7.5.2.

7.3.3 Interpretable Motifs



(a) Best and worst motifs identified on NB201-CIFAR10 dataset using 300 training samples (left) and on DARTS search space after 3 GPU days of search by NAS-BOWL (right). For DARTS space, the motif boxed in pink is featured in all optimal cells found by various NAS methods in Figure 7.3.



(b) Validation accuracy distributions of the validation architectures on *different tasks* of NB201 (left 3) and DARTS (right). all denotes the entire validation set, while good/bad denote the distributions of the architectures with at least 1 best/worst motif, respectively; dashed lines denote the distribution medians. Note that in all cases, the good subset includes the population max and in NB201, the patterns solely trained on the CIFAR10 task also transfer well to CIFAR100/ImageNet120.

Figure 7.2: Motif discovery on NB201-CIFAR10 and DARTS spaces.

In the preceding section, we have elaborated the advantages of using WL graph kernels to make the NAS search space amenable to GP-based BO. A key advantage of WL that we identify in Section 7.3.1 is that it extracts *interpretable* features, i.e. network motifs from the original graphs. This in combination with our GP surrogate enables us to predict the effect of the extracted features on the architecture performance directly by examining the derivatives of the GP predictive mean w.r.t. the features. Derivatives as tools to interpret ML models have been used previously (Engelbrecht et al., 1995; Koh and Liang, 2017; Ribeiro et al., 2016) but, given the GP, we can compute these derivatives *analytically*. Following the notations in Section 7.2, the derivative with respect to $\phi^j(G_t)$, the j-th element of $\boldsymbol{\phi}(G_t)$ (the feature vector of a graph G_t) is Gaussian with an expected value:

$$\mathbb{E}_{p(y|G_t,\mathcal{D}_{t-1})}\left[\frac{\partial y}{\partial \phi^j(G_t)}\right] = \frac{\partial \mu}{\partial \phi^j(G_t)} = \frac{\partial \langle \boldsymbol{\phi}(G_t), \Phi_{1:t-1}\rangle}{\partial \phi^j(G_t)} \mathbf{K}_{1:t-1}^{-1} \mathbf{y}_{1:t-1} \tag{7.6}$$

where $\Phi_{1:t-1} = [\boldsymbol{\phi}(G_1), \ldots, \boldsymbol{\phi}(G_{t-1})]^{\mathrm{T}}$ is the feature matrix stacked from the feature vectors of the previous observations. Intuitively, since each $\phi^j(G_t)$ denotes the count of a WL feature in G_t, its derivative naturally encodes the direction and sensitivity of the objective (in this case the predicted validation accuracy) about that feature. Computationally, since the costly term, $\mathbf{K}_{1:t-1}^{-1}\mathbf{y}_{1:t-1}$, is already computed in the posterior mean, the derivatives can be obtained at minimal additional cost.

By evaluating the aforementioned derivative at some graph G, we obtain the *local* sensitivities of the objective function around $\boldsymbol{\phi}(G)$. To achieve *global* attribution of network performance w.r.t interpretable features which we are ultimately interested in, we take inspirations from the principled averaging approach featured in many gradient-based attribution methods (Sundararajan et al., 2017; Ancona et al., 2017), by computing and integrating over the aforementioned derivatives at all training samples to obtain the *averaged gradient* (AG). AG of the j-th feature ϕ_j is given by:

$$\mathrm{AG}(\phi^j) = \mathbb{E}_G\left[\frac{\partial \mu}{\partial \phi^j(G)}\right] = \int_{\phi^j(G)>0} \frac{\partial \mu}{\partial \phi^j(G)} p(\phi^j(G)) d\phi^j(G). \tag{7.7}$$

Fortunately, in WL kernel, $\phi^j(\cdot) \in \mathbb{Z}^{\geq 0}\ \forall j$ and thus $p(\phi^j(\cdot))$ is discrete, the expectation integral reduces to a weighted summation over the "prior" distribution

$p(\phi^j(\cdot))\forall j$. To approximate $p(\phi^j(\cdot))$, we count the number of occurrences of each feature $\phi^j(G_n)$ in all the training graphs $\{G_1, ..., G_{t-1}\}$ where $\phi^j(\cdot)$ is present and assign weights according to its frequency of occurrence. For example, suppose in 100 training graphs, feature ϕ^j occurs in 50 of them, and out of the 50 graphs, in 30 graphs it appears once and appears twice in the rest. Its AG is given by $\frac{30}{50}\frac{\partial \mu}{\partial \phi^j(G_t)}|_{\phi^j(G_t)=1} + \frac{20}{50}\frac{\partial \mu}{\partial \phi^j(G_t)}|_{\phi^j(G_t)=2})$.

Formally, denoting $\mathcal{G}$ as the subset of the training graphs where for each of its element $\phi^j > 0$, we have

$$\text{AG}(\phi^j) \approx \frac{\sum_{n=1}^{|\mathcal{G}|} w_n(\phi_j)\frac{\partial \mu}{\partial \phi^j(G_n)}}{\sum_{n=1}^{|\mathcal{G}|} w_n(\phi_j)} \text{ where } w_n(\phi_j) = \frac{1}{|\mathcal{G}|}\sum_{n'=1}^{|\mathcal{G}|}\delta(\phi_j(G_n), \phi_j(G_{n'})) \quad (7.8)$$

where $\delta(\cdot,\cdot)$ is the Kronecker delta function and $|\cdot|$ the cardinality of a set. Finally, we additionally incorporate the uncertainty of the derivative estimation by also normalising AG with the square root of the *empirical variance* (EV) to penalise high-variance (hence less trustworthy as a whole) gradient estimates closer to 0. EV may be straightforwardly computed:

$$\text{EV}(\phi^j) = \mathbb{V}_G\Big[\frac{\partial \mu}{\partial \phi^j(G)}\Big] = \mathbb{E}_G\Big[\Big(\frac{\partial \mu}{\partial \phi^j(G)}\Big)^2\Big] - \Big(\mathbb{E}_G\Big[\frac{\partial \mu}{\partial \phi^j(G)}\Big]\Big)^2. \quad (7.9)$$

The resultant derivatives w.r.t. interpretable features $\text{AG}(\phi^j)/\sqrt{\text{EV}(\phi^j)}$ allow us to directly identify the most influential motifs on network performance. By considering the presence or absence of such motifs, we may explain the competitiveness of an architecture or the lack of it, provided the surrogate is accurate which we show is the case in Section 7.5. More importantly, beyond passive *explaining*, we can also actively use these features as building blocks to facilitate manual *construction* of promising networks, or as priors to *prune* the massive NAS search space, which we believe would be of interest to both human designers and NAS practitioners.

To validate this, we train our GPWL on architectures drawn from respective search spaces, rank all the features based on their computed derivatives and show the motifs with most positive and negative derivatives (hence the most and least desirable features) Specifically, for NASBench-201(NB201) and NASBench-101(NB101) datasets, we train the GPWL surrogate on 300 random architectures to

compute the motif derivatives, and label the top and bottom quantiles of the motifs as ”best motifs” and ”worst motifs” respectively. As for the open-domain DARTS search space, we train the GPWL surrogate on 120 architectures queried by NAS-BOWL during one round of search and select the top and bottom 15% of the motifs as the "best motifs” and "worst motifs" respectively. As a regularity constraint, we only consider the motifs which appear for more than 10 times in the NASBench training set or appear at least twice in the DARTS training set to filter out rare outliers.

We present the extracted network motifs on the CIFAR10 task of NB201 (Dong and Yang, 2020b) and DARTS search space in Figure 7.2(a). The motifs extracted on other NB201 image tasks (CIFAR100/ImageNet120) and on NB101 (Ying et al., 2019) are shown in Appendix A.6. The results reveal some interesting insights on network performance: for example, almost every good motif in NB201 contains `conv_3×3` and all-but-one good motifs in the DARTS results contain at least one *separable* `conv_3×3`. In fact, this preference over (separable) convolutions is almost universally observed in many popular NAS methods (Liu et al., 2018b; Shi et al., 2019; Wang et al., 2019; Pham et al., 2018) and ours: besides skip links, the operations in their best cells are dominated by separable convolutions (Figure 7.3). Moving from node operation label to higher-order topological features, in both search spaces, our GPWL consistently finds a series of high-performing motifs that entail the parallel connection from input to multiple `convs` often of different filter sizes – this corresponds to the *grouped convolution* unit critical to the success of, e.g. ResNeXt (Xie et al., 2017). A specific example is the boxed motif in Figure 7.2(a), which combines parallel `convs` with a skip link. This motif or other highly similar ones are consistently present in the optimal cells found by many NAS methods including ours (as shown in Figure 7.3) despite the disparity in their search strategies. This suggests a correlation between these motifs and good architecture performance. Another observation is that a majority of the important motifs for both search spaces in Figure 7.2(a) involve the input. From this and our previous remarks on the consensus amongst NAS methods in favouring certain operations and connections, we hypothesise that at least for a cell-based search

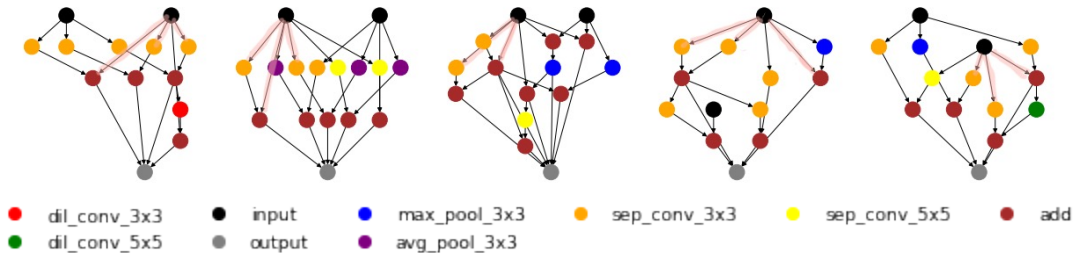


Figure 7.3: Best cells discovered by (left to right) DARTS, ENAS, LaNet, BOGCN and NAS-BOWL (ours) in the DARTS search space. Note the dominance of separable convolutions (especially 3x3) in operation nodes (i.e. nodes excluding `input`, `output` and `add`) and the presence of highlighted structures encompassing the boxed motif in Figure 7.2 in all cells.

space, the network performance might be determined more by the connections in the vicinity to the inputs (on which the optimal cells produced by different NAS methods are surprisingly consistent) than other parts of the network (on which they differ). This phenomenon is partly observed in Shu et al. (2019), where the authors found that NAS algorithms tend to favour architecture cells with most intermediate nodes having direct connection with the input nodes. The verification of this hypothesis is beyond the scope of this work, but this shows the potential of our GPWL in discovering novel yet interpretable network features, with potential implications for both NAS and manual network design.

Going beyond the qualitative arguments above, we now quantitatively validate the informativeness of the motifs discovered. After identifying the motifs, in NB201, we randomly draw another 1,000 validation architectures unseen by the surrogate. Given the motifs identified in Figure 7.2(a), an architecture is labelled either "good" (≥ 1 good motif), "bad" (≥ 1 bad motif) or neither. Note if an architecture contains both good and bad motifs, it is both "good" and "bad". As demonstrated in Figure 7.2(b), we indeed find that the presence of important motifs is predictive of network performance. A similar conclusion holds for DARTS space. However, due to the extreme cost in sampling the open-domain space, we make two modifications to our strategy. Firstly, the training samples are taken from a BO run, instead of randomly sampled. Thus we are more biased towards the better performing networks, explaining the smaller gap between "good" and "all" in Figure 7.2(b). Secondly,

we reuse the training samples for Figure 7.2(b) instead of sampling and evaluating the hold-out sets. The key takeaway here is that motifs are effective in identifying promising and unpromising candidates, and thus can be used to aid NAS agents to partition the vast combinatorial search space, which is often considered a key challenge of NAS, and to focus on the most promising sub-regions. More importantly, the motifs are also *transferable*: while the patterns in Figure 7.2(a) are solely trained on the CIFAR10, they generalise well to CIFAR100/ImageNet120 tasks – this is unsurprising, as one key motivation of cell-based search space is *exactly* to improve transferability of the learnt structure across related tasks (Zoph et al., 2018a). Given that motifs are the building blocks of the cells, we expect them to transfer well, too.

With this, we propose a simple transfer learning baseline as a singular demonstration of how motifs could be practically useful for NAS. Specifically, we can exploit the motifs identified on one task to warm-start the search on a related new task. With reference to Algorithm 11, under the transfer learning setup, we use a GPWL surrogate trained on the query data of a past related task $\mathcal{S}_{\text{past}}$ as well as the surrogate on the new target task $\mathcal{S}$ to compute the AG of motifs present in queried architectures (equation 7.8) and identify the most positively influential motifs similar to Figure 7.2(a). We then use these motifs to generate a set of candidate architectures $\mathcal{G}_t$ for optimising the acquisition function at every BO iteration on the new task; Specifically, we only accept a candidate if it contains at least one of the top 25% good motifs (i.e. *pruning rule*). Finally, with more query data obtained on the target task, we will dynamically update the surrogate $\mathcal{S}$ and the motif scores to mitigate the risk of discarding motifs purely based on the past task data. Through this, we force the BO agent to select from a smaller subset of architectures deemed more promising from a previous task, thereby "warm starting" the new task. We briefly validate this proposal in the NB201 experiments of Section 7.5.2.

7.4 Related Work

In terms of NAS strategies, there have been several recent attempts in using BO (Kandasamy et al., 2018b; Ying et al., 2019; Ma et al., 2019a; Shi et al., 2019; White

et al., 2021a). To overcome the limitations of conventional BO for discrete and graph-like NAS search spaces, (Kandasamy et al., 2018b) use optimal transport to design a similarity measure among neural architectures while (Ying et al., 2019) and (White et al., 2021a) suggest encoding schemes to characterise neural architectures with discrete and categorical variables. Yet, these methods are either computationally inefficient or not scalable to large architectures/cells (Shi et al., 2019; White et al., 2021a). Alternatively, several works use graph neural networks (GNNs) as the surrogate model (Ma et al., 2019a; Zhang et al., 2019; Shi et al., 2019) to capture the graph structure of neural networks. However, the design of the GNN introduces many additional hyper-parameters to be tuned and GNN requires a relatively large number of training data to achieve decent prediction performance as shown in Section 7.5.1. Another related work (Ramachandram et al., 2018) apply GP-based BO with diffusion kernels to design multimodal fusion networks; however, it assigns each possible architecture as a node in an *undirected* super-graph and the need for construction of and computation on such super-graphs limits the method to relatively small search spaces. In terms of interpretability, (Shu et al., 2019) study the connection pattern of network cells found by popular NAS methods and find a shared tendency for choosing wide and shallow cells which enjoy faster convergence. (You et al., 2020a), by representing neural networks as relational graphs, observe that the network performance depends on the clustering coefficient and average path length of its graph representation. (Radosavovic et al., 2020) propose a series of manual design principles derived from extensive empirical comparison to refine a ResNet-based search space. Nevertheless, all these works do not offer a NAS strategy, and purely rely on human experts to derive insights on NAS architectures from extensive empirical studies. In contrast, our method learns the interpretable feature information without human inputs *while* searching for the optimal architecture.

7.5 Experiments

7.5.1 Surrogate Regression Performance

We examine the regression performance of GPWL on several NAS datasets: NASBench-101(NB101) on CIFAR10 (Ying et al., 2019), and NASBench-201(NB201) on CIFAR10, CIFAR100 and ImageNet120. As both datasets only contain CIFAR-sized images and relatively small architecture cells[3], to further demonstrate the scalability of our proposed methods to much larger architectures, we also construct a dataset with 547 architectures sampled from the randomly wired graph generator described in Xie et al. (2019a); each architecture cell has 32 operation nodes and all the architectures are trained on the Flower102 dataset (Nilsback and Zisserman, 2008) (we denote this dataset as RWNN-Flower102 hereafter). Please refer to Appendix B.5 for more details on the datasets. Similar to (Ying et al., 2019; Dong and Yang, 2020b; Shi et al., 2019), we use Spearman's rank correlation between predicted validation accuracy and the true validation accuracy as the performance metric, as what matters for comparing architectures is their relative performance ranking.

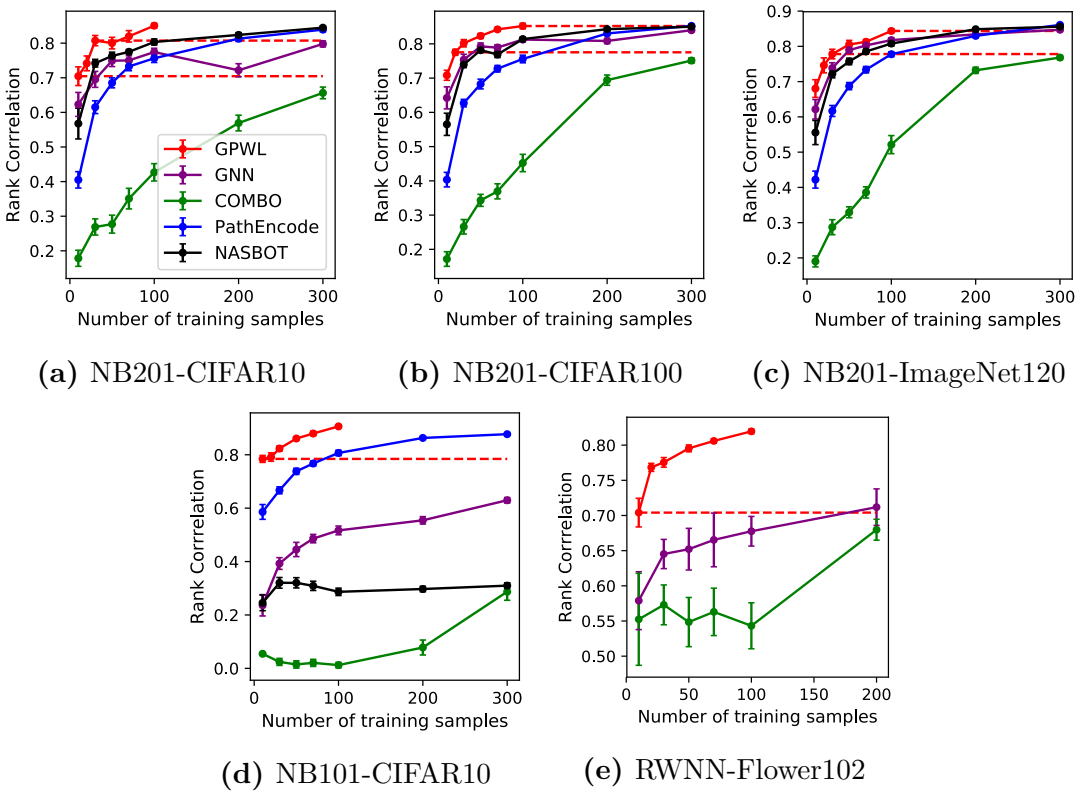
Comparison Against Other Surrogates

We first compare the regression performance against various competitive baselines, including NASBOT (Kandasamy et al., 2018b), GPs with path encodings (PathEncode) (White et al., 2021a), GNN (Shi et al., 2019) which uses a combination of graph convolutional network and a final Bayesian linear regression layer as the surrogate, and COMBO (Oh et al., 2019)[4], which use a GP with a diffusion kernel on a graph representation of the combinatorial search spaces. We implemented the GNN surrogate by ourself and use the released code for COMBO.

We repeat each trial 20 times on a varying number of training data to evaluate the data-efficiency of different surrogates, and report the results in Figure 7.4: our GPWL surrogate clearly outperforms all competing methods on all the NAS datasets

[3] In NB101 and NB201, each cell is a graph of 7 and 4 nodes, respectively.

[4] We choose COMBO as methodologically it is very close to the most related work (Ramachandram et al., 2018) whose implementation is not publicly available.



(a) NB201-CIFAR10 (b) NB201-CIFAR100 (c) NB201-ImageNet120

(d) NB101-CIFAR10 (e) RWNN-Flower102

Figure 7.4: Mean Spearman correlation achieved by various surrogates across 20 trials on different datasets. Error bars denote ±1 standard error. The red dashed lines are there to help with visual comparison between the performance of GPWL and other baselines. GPWL can achieve superior regression performance measured in terms of high rank correlation with at least 3 times fewer training data than GCN on NB201 (CIFAR10, CIFAR100, ImageNet120) datasets and over 20 times fewer training data on datasets with larger search spaces (RWNN-Flower102 and NB101-CIFAR10).

with much less training data: specifically, GPWL requires at least 3 times less data than GNN and PathEncode and 10 times less than COMBO on NB201 datasets. It is also able to achieve high rank correlation on datasets with larger search spaces such as NB101-CIFAR10 and RWNN-Flower102 while requiring 20 times less data than GNN on RWNN-Flower102 and 30 times less data on NB101-CIFAR10.

Comparison Against Other Graph Kernels

In addition to these surrogates previously used in NAS, we further compare the performance of the WL kernel against other popular graph kernels when combined with GPs. The graph kernels compared include random walk (RW) kernel (Kashima et al., 2003; Gärtner et al., 2003), shortest-path (SP) kernel (Borgwardt and Kriegel,

Table 7.1: Regression performance in terms of Spearman's rank correlation for different graph kernels. WL graph kernel adopted by us consistently outperforms other kernels across search spaces while retaining modest computational costs.

Kernel	Complexity	NB101	NB201			RWNN
		CIFAR10	CIFAR10	CIFAR100	ImageNet120	Flower102
WL	$\mathcal{O}(Hm)$	**0.862**±0.03	**0.812**±0.06	**0.823**±0.03	**0.796**±0.04	**0.804**±0.018
RW	$\mathcal{O}(n^3)$	0.801±0.04	0.809±0.04	0.782±0.06	0.795±0.03	0.759±0.04
SP	$\mathcal{O}(n^4)$	0.801±0.05	0.792±0.06	0.761±0.06	0.762±0.08	0.694±0.08
MLP	$\mathcal{O}(Ln^5)^\dagger$	0.458±0.07	0.412±0.15	0.519±0.14	0.538±0.07	0.492±0.12

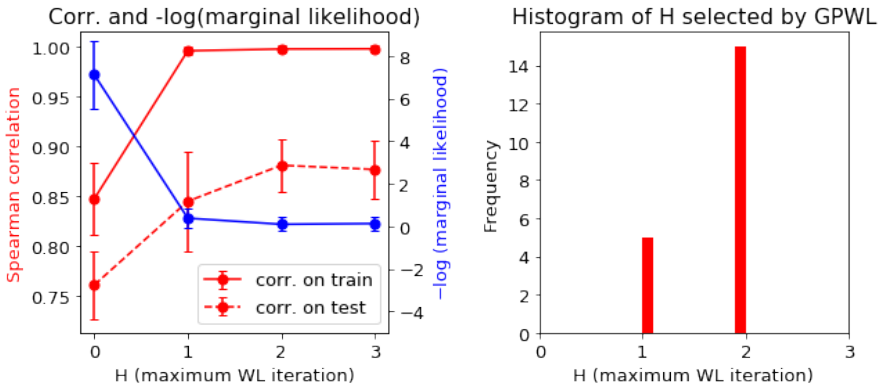
†: L is the number of neighbours, a hyper-parameter of MLP kernel.

2005) and multiscale Laplacian (MLP) kernels(Kondor and Pan, 2016). These competing graph kernels are chosen because they represent distinct graph kernel classes and are suitable for NAS search space with small or no modifications. In each NAS dataset, we randomly sample 50 architecture data to train the GP surrogate and use another 400 architectures as the validation set to evaluate the rank correlation between the predicted and the ground-truth validation accuracy.

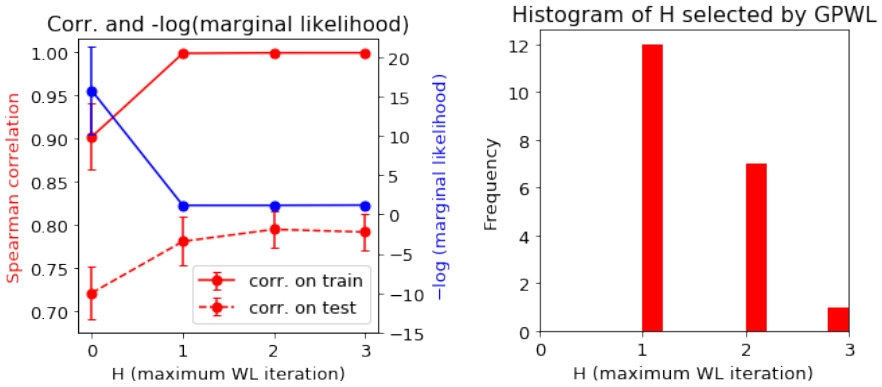
We repeat each trial 20 times, and report the mean and standard error of all the kernel choices on all NAS datasets in Table 7.1. We also include the worst-case complexity of the kernel computation between a pair of graphs in the table. The results justify our reasoning in Section 7.3.1; combined with the interpretability benefits we discussed, WL consistently outperforms other kernels across search spaces while retaining the modest computational costs. RW often comes a close competitor, but its computational complexity is worse and does not always converge. MLP, which requires us to convert directed graphs to undirected graphs, performs poorly, thereby validating that directional information is highly important.

Maximum Number of the WL Iterations, H

As discussed, the Weisfeiler-Lehman kernel is only parameterised by H, the maximum number of WL iterations. The expressive power of the kernel generally increases with H, as the kernel is capable of covering increasingly global features,



(a) NB101-CIFAR10



(b) RWNN-Flower102

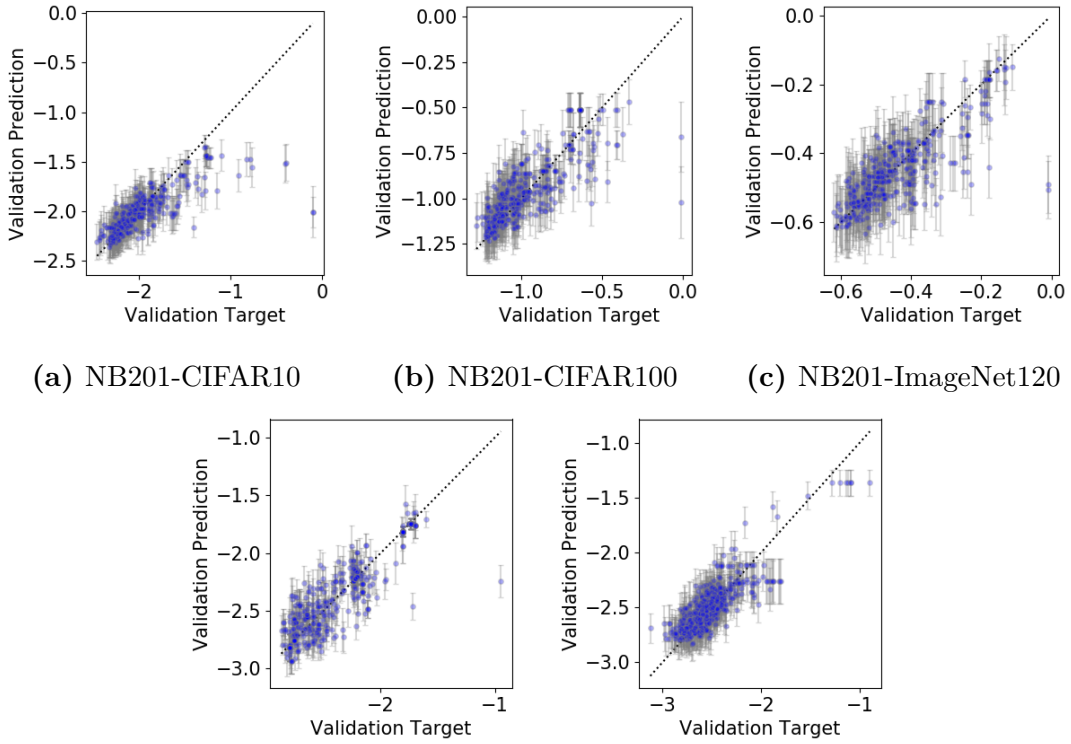
Figure 7.5: Spearman correlation on train/validation sets and the negative log-marginal likelihood of GP against H (the maximum WL iteration) and the histograms of selected H by GPWL over 20 trials on (a) NB101-CIFAR10 and (b) RWNN-Flower102.

but at the same time we might overfit into the training set, posing a classical problem of variance-bias trade-off. In this work, by combining WL with GP, we optimise H against the negative log-marginal likelihood of the GP. In this section, on different data-sets we show that this approach satisfactorily balances data-fitting with model complexity.

To verify on both NB101-CIFAR10 and RWNN-Flower102 datasets, we train GPWL surrogates on 50 random training samples. On NB101, we draw another 400 testing samples and on RWNN-Flower102, we use the rest of the data-set as the validation set. We use the Spearman correlation between prediction and the ground truths of the validation set as the performance metric. We summarise our result in Figure 7.5: in both data-sets, we observe a large jump in performance

from $H = 0$ to 1 (measured by the improvements in both validation and training Spearman correlation), and a slight dip in validation correlation from $H = 2$ to 3, suggesting an increasing amount of overfitting if we increase H further. In both cases, the automatic selection described above succeeded in finding the "sweet spot" of $H = 1$ or 2, demonstrating the effectiveness of the approach.

Good Uncertainty Estimates of GPWL



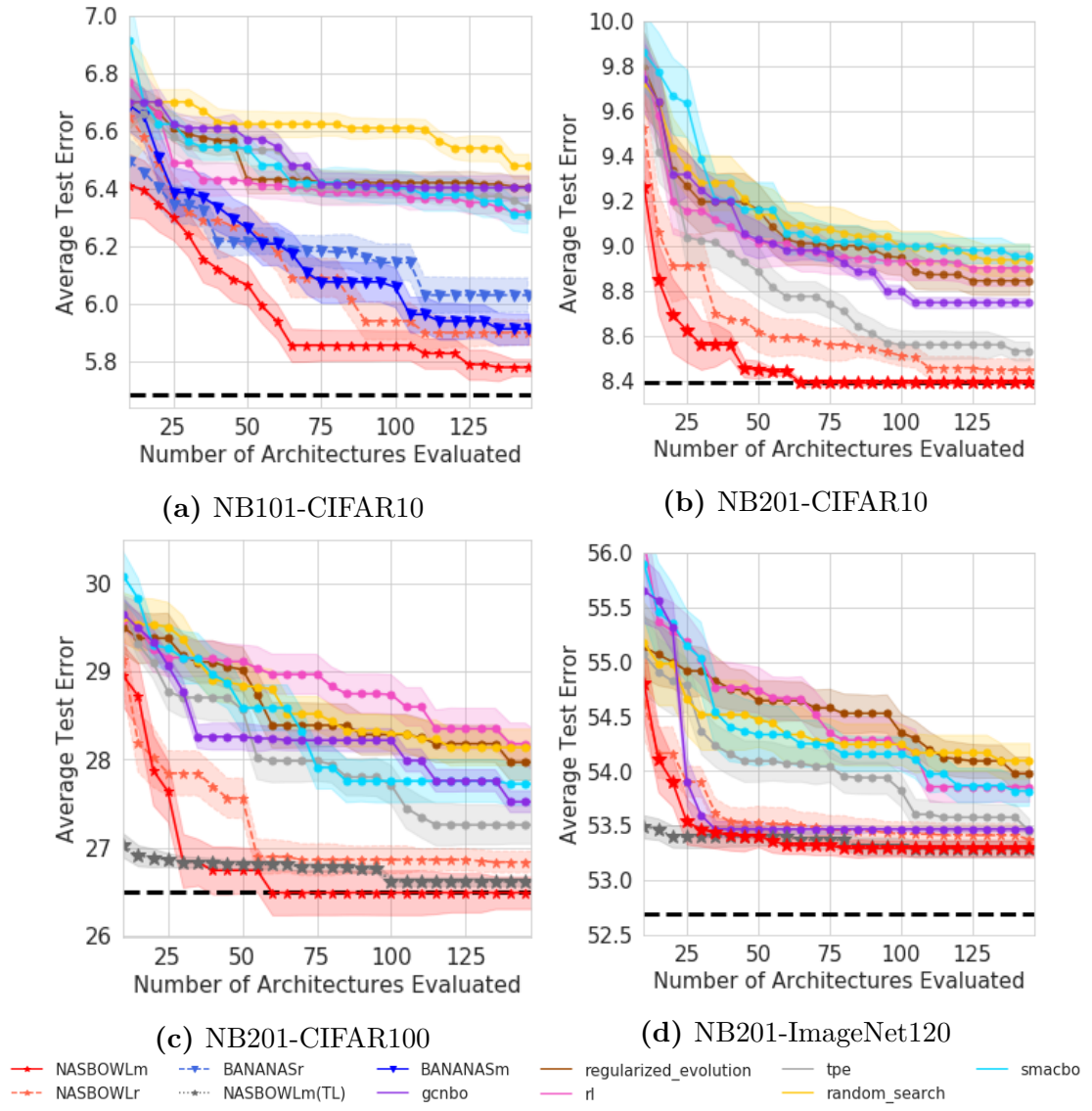
(a) NB201-CIFAR10 **(b)** NB201-CIFAR100 **(c)** NB201-ImageNet120

(d) NB101-CIFAR10 **(e)** RWNN-Flower102

Figure 7.6: Predicted vs ground-truth validation error of GPWL in various NAS-Bench tasks in log-log scale. Error bar denotes ±1 SD from the GP posterior predictive distribution.

We further show the GPWL predictions on the various NAS datasets when trained with 50 samples each in Figure 7.6. It can be shown that not only a satisfactory predictive mean is produced by GPWL in terms of the rank correlation and the agreement with the ground truth, there is also sound uncertainty estimates, as we can see that in most cases the ground truths are within the error bar representing one standard deviation of the predictive posterior distributions of GPWL.

7.5.2 Architecture Search on NAS-Bench Datasets

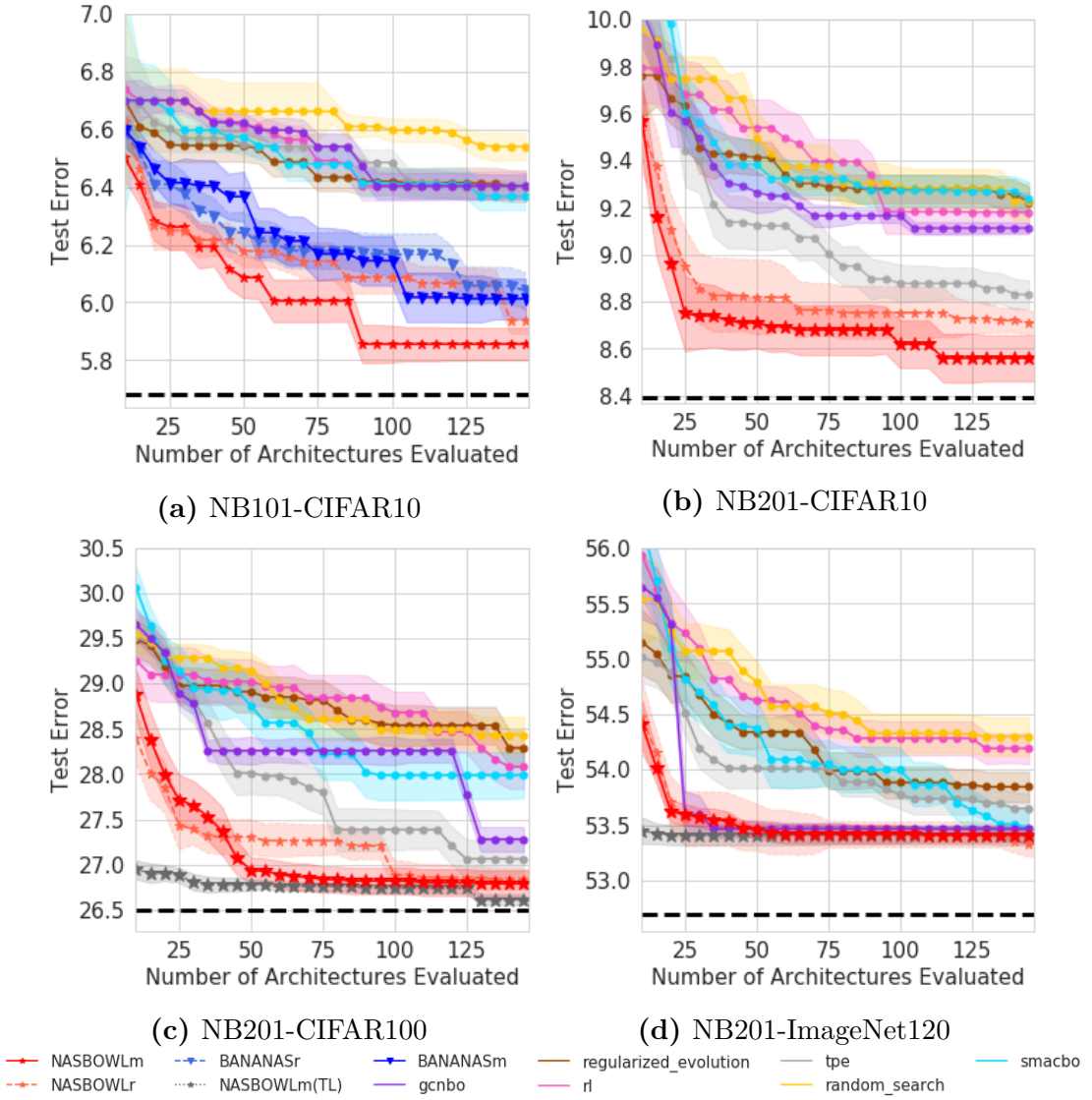


(a) NB101-CIFAR10 **(b)** NB201-CIFAR10

(c) NB201-CIFAR100 **(d)** NB201-ImageNet120

Figure 7.7: Median test error on NAS-Bench datasets with *deterministic* observations from 20 trials. Shades denote ±1 standard error and black dotted lines are ground-truth optima. Note the seemingly large regret in NB201-ImageNet120 is due to that there are only 5 out of 15.6K architectures with test error in the interval of [52.69 (optimum), 53.25].

We benchmark our proposed method, NAS-BOWL, against a range of existing methods, including random search, TPE (Bergstra et al., 2011b), Reinforcement Learning (rl) (Zoph and Le, 2016), BO with SMAC (smacbo) (Hutter et al., 2011b), regularised evolution (Real et al., 2019) and BO with GNN surrogate (gcnbo) (Shi et al., 2019). On NB101, we also include BANANAS (White et al., 2021a)



(a) NB101-CIFAR10 **(b)** NB201-CIFAR10

(c) NB201-CIFAR100 **(d)** NB201-ImageNet120

Figure 7.8: Median test error on NAS-Bench datasets with *noisy* observations from 20 trials. Shades denote ±1 standard error and black dotted lines are ground-truth optima.

which claims the state-of-the-art performance. More implementation details of NAS-BOWL and these baseline methods can be found in Appendix B.3. In both NAS-Bench datasets, validation errors of different random seeds are provided, thereby creating noisy objective functions. We perform experiments using the deterministic setup described in White et al. (2021a), where the validation errors over multiple seeds are averaged to eliminate stochasticity, and also report results with noisy objective functions. We show the test results in both setups in Figure 7.7 and the validation results in Appendix A.7. In these figures, we

use NASBOWLm and NASBOWLr to denote NAS-BOWL with architectures generated from mutating good observed candidates and from random sampling, respectively. Similarly, BANANASm/BANANASr represent the BANANAS with mutation/random sampling (White et al., 2021a). On CIFAR100/ImageNet120 tasks of NB201, we also include NASBOWLm(TL) which is NASBOWLm with additional knowledge on motifs transferred from a previous run on the CIFAR10 task of NB201 to prune the candidate architectures as described in Section 7.3.3. The readers are referred to Appendix B.3 for detailed setups.

It is evident that NAS-BOWL outperforms all baselines on all NAS-Bench tasks in achieving both lowest validation and test errors. The recent neural-network-based methods such as BANANAS and GCNBO are often the strongest competitors, but we would like to highlight that unlike our approach, these methods inevitably introduce a number of extra hyper-parameters whose tuning is non-trivial and have poorly calibrated uncertainty estimates (Springenberg et al., 2016b). The experiments with noisy observations in Figure 7.8 further show that even in a more realistic setup with noisy objective function observations, NAS-BOWL still performs very well as it inherits the robustness against noise from the GP. The preliminary experiments on transfer learning also show that motifs contain extremely useful prior knowledge that may be transferred to warm-start a related task: notice that even the architectures at the very start without any search already perform well – this is particularly appealing, as in a realistic setting, searching directly on large-scale datasets like ImageNet from scratch is extremely expensive. While further experimental validation on a wider range of search spaces or tasks of varying degrees of similarity are required to fully verify the effectiveness of this particular method, we feel as an exemplary use of motifs, the promising preliminary results here already demonstrates the usefulness.

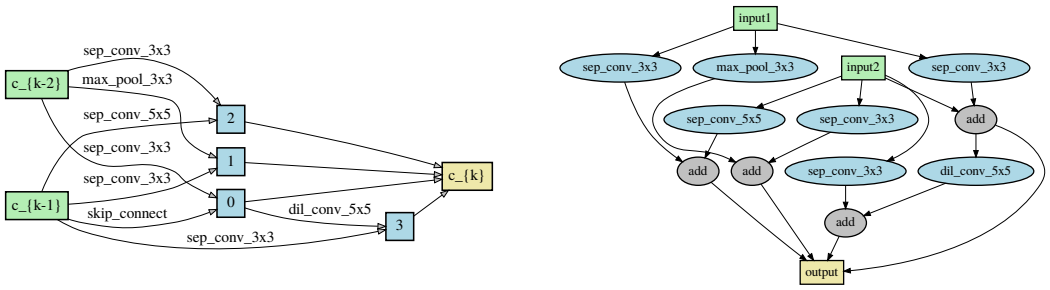
7.5.3 Architecture Search on Open-domain Space

We finally test NAS-BOWL on the open-domain search space from DARTS (Liu et al., 2018b). We allow a maximum budget of 150 queries, and we follow the

Table 7.2: Performances on CIFAR10. GPU days do *not* include the evaluation cost of the final architecture; NAS-BOWL results from 4 random seeds on a single NVIDIA GeForce RTX 2080 Ti.

Algorithm	Avg. Error	Best Error	#Params(M)	GPU Days
GP-NAS (Li et al., 2020b)	-	3.79	3.9	1
DARTS(v2) (Liu et al., 2018b)	$2.76_{\pm 0.09}$	-	3.3	4
ENAS† (Pham et al., 2018)	-	2.89	4.6	6
ASHA(Li and Talwalkar, 2019)	$3.03_{\pm 0.13}$	2.85	2.2	9
Random-WS (Xie et al., 2019a)	$2.85_{\pm 0.08}$	2.71	4.3	10
BANANAS (White et al., 2021a)	2.64	2.57	-	12
BOGCN (Shi et al., 2019)	-	2.61	3.5	93*
LaNet† (Wang et al., 2019)	$2.53_{\pm 0.05}$	-	3.2	150
NAS-BOWL	$\mathbf{2.61_{\pm 0.08}}$	**2.50**	**3.7**	**3**

†: expanded search space from DARTS. *: estimated by us. -: not reported.



(a) Edge-attributed DAG

(b) Node-attributed DAG

Figure 7.9: Equivalent representations of the best cell identified by NAS-BOWL in the DARTS search space. Our method uses the node-attributed version during search, and this cell is used for both the normal and reduction cells.

DARTS setup (See Appendix B.4 for details): during the search phase, instead of training the final 20-cell architectures, we train a small 8-cell architectures for 50 epochs. Whereas this results in significant computational savings, it also leads to the degraded rank correlation of performance during search and evaluation stages. This leads to a more challenging setup than most other sample-based methods which train for longer epochs and/or search on the final 20-cell architectures directly. Beyond this, we also search a single cell structure and use it for the two cell types

(normal and reduction) defined in DARTS search space. We also use a default maximum number of 4 operation blocks to fit the training on a single GPU; this is contrasted to, e.g. ENAS and LaNet that allow for up to 5 and 7 blocks, respectively.

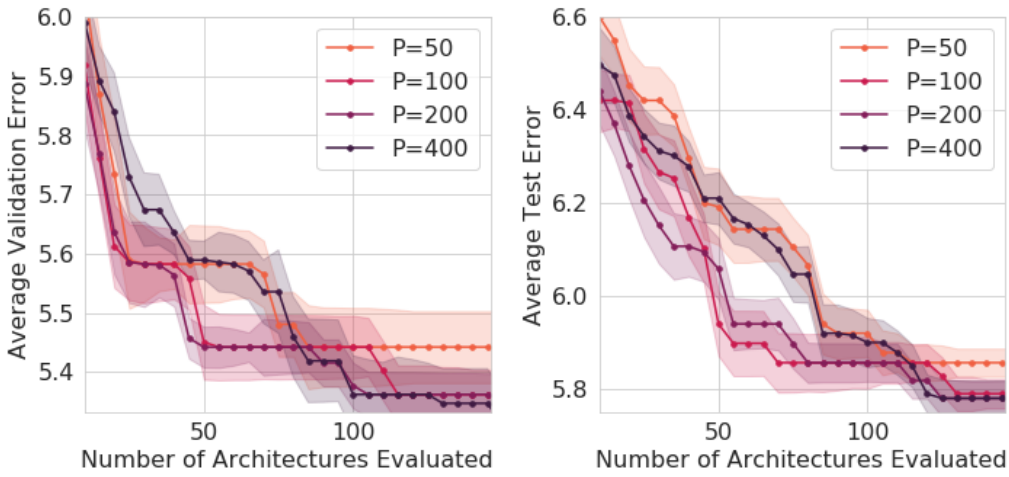
We compare NAS-BOWL with other methods in Table 7.2, and the best cell found by NAS-BOWL is shown in Figure 7.9. To ensure fairness of comparison, we only include previous methods with comparable search spaces and training techniques, and exclude methods that train much longer and/or use additional tricks (Liang et al., 2019; Cai et al., 2019). Furthermore, since the GPU time for BOGCN (Shi et al., 2019) is not reported, we estimate it based on the methodology of the paper: during search stage they train each final architecture for 100 epochs and sample 400 architectures; we assume training each architecture takes 200s per epoch, a runtime typical on a single NVIDIA GeForce RTX 2080 Ti GPU. It is evident that NAS-BOWL finds very promising architectures despite operating in a more restricted setup. Consuming 3 GPU-days, NAS-BOWL is of comparable computing cost to the one-shot methods but performs on par with or better than methods that consume orders of magnitude more resources, such as LaNet which is 50× more costly. Furthermore, it is worth noting that, if desired, NAS-BOWL may benefit from any higher computing budgets by relaxing the aforementioned restrictions (e.g. train longer on larger architectures during search). Finally, while we use a single GPU, NAS-BOWL can be easily deployed to run on parallel computing resources to further reduce wall-clock time.

7.5.4 Ablation Studies

Finally, we perform ablation studies on the candidate pool size P as well as several other design choices in our proposed approach, NAS-BOWL.

Effect of Varying Pool Size

As discussed in the main text, NAS-BOWL introduces no inherent hyper-parameters that require manual tuning as it relies on a non-parametric surrogate. Nonetheless, besides the surrogate, the choice on how to generate the candidate architectures



(a) Validation Error **(b)** Test Error

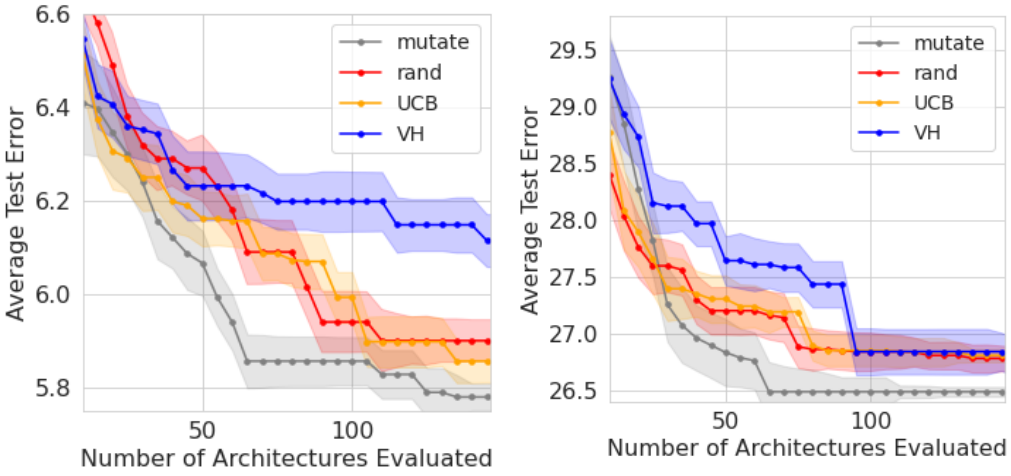
Figure 7.10: Effect of varying P on NAS-BOWL in NB101.

requires us to specify the pool size (P, the number of candidate architectures to generate at each BO iteration). In all our NAS experiments above, we have set $P = 200$ following (White et al., 2021a); here we show that the performance of NAS-BOWL is robust to the value of P.

We vary $P \in \{50, 100, 200, 400\}$, and keep all other settings to be consistent. We report our results on NB101 with deterministic observations in Figure 7.10 where the median result is computed from 20 experiment repeats. It can be shown that while the convergence speed varies slightly between the different P choices, for all choices of P apart from 50 which performs slightly worse, NAS-BOWL converges to similar test errors at the end of 150 architecture evaluations – this suggests that the performance of NAS-BOWL is rather robust to the value of P and that our recommendation of $P = 200$ does perform well both in terms of both the final solution returned and the convergence speed.

Other Design Choices

We further conduct ablation studies on both NB101 and NB201 with deterministic test errors. We repeat each experiment 20 times and present the median and standard error in terms of test performances in Figure 7.11. We now explain each legend as follow:



(a) Test Error on NB101

(b) Test Error on NB201

Figure 7.11: Ablation studies of various design choices for NAS-BOWL

1. *mutate*: Full NAS-BOWL with the *mutation* described in Section 7.3.2 (identical to NASBOWLm in Figures 7.7 and 7.8);

2. *rand*: NAS-BOWL with *random* candidate generation. This is identical to NASBOWLr in Figures 7.7 and 7.8;

3. *UCB*: NAS-BOWL with *random* candidate generation, but with the acquisition function changed from Expected Improvement (EI) to Upper Confidence Bound (UCB) (Srinivas et al., 2009).

4. *VH*: NAS-BOWL with *random* candidate generation, but instead of leaving the value of h (number of WL iterations) to be automatically determined by the optimisation of the GP log marginal likelihood, we set $h = 0$, i.e. no WL iteration takes place and the only features we use are the counts of each type of original node operation features (e.g. `conv3×3-bn-relu`). This essentially reduces the WL kernel to a Vertex Histogram (VH) kernel.

We find that topological information and using an appropriate h are highly crucial: in both NB101 and NB201, `VH` significantly underperforms the other variants, although the extent of underperformance is smaller in NB201 likely due to its smaller search space. This suggests that how the nodes are connected, which

are extracted as higher-order WL features, are very important, and the multi-scale feature extraction in the WL kernel is crucial to the success of NAS-BOWL. On the other hand, the choice of the acquisition function seems not to matter as much, as there is little difference between `UCB` and `WL` runs in both NB101 and NB201. Finally, using mutation algorithm leads to a significant improvement in the performance of NAS-BOWL, as we have already seen in the main text.

7.6 Conclusion

In this chapter, we propose a novel BO-based NAS strategy, NAS-BOWL, which uses a GP surrogate with the WL graph kernel to naturally handle the graph input space and thus achieves competitive performance with remarkable query efficiency. Building on our proposed framework, the rich literatures of GP-based BO techniques in hyper-parameter optimisation can be re-applied to NAS, tackling more challenging problems like searching for *multi-objectives* or warm-starting the search with *transfer learning*. More importantly, our method represents a first step towards *interpretable NAS*, where we propose to learn interpretable network features to help explain the architectures found as well as guide the search on new tasks.

Future directions We believe interpretable NAS is an exciting direction that warrants future investigations. Possible extensions include finding approaches other than the use of the surrogate gradients proposed in this work to assess the contribution of interpretable motifs to the architecture performance, developing alternatives to identify architecture motifs which are different from the WL subtree features and even macro substructures which span cross multiple architecture cells, as well as suggesting other novel scenarios to make use of the interpretable insights learnt during NAS. Hopefully, interpretable NAS can lead to new design paradigms or guidelines that are valuable to human practitioners.

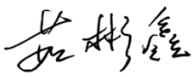
Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Interpretable Neural Architecture Search via Bayesian Optimisation with Weisfeiler-Lehman Kernels
Publication Status	☒Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Binxin Ru*, Xingchen Wan*, Xiaowen Dong, and Michael A. Osborne. Interpretable neural architecture search via Bayesian optimisation with Weisfeiler-Lehman kernels. In *International Conference on Learning Representations (ICLR 2021),* 2021.

Student Confirmation

Student Name:	Binxin Ru		
Contribution to the Paper	Wan and I contributed equally to the research idea discussions, experiments and code reviews and result analyses as well as the write-up of the paper. I was responsible for processing NAS-Bench datasets and running competing baseline methods surrogate regression and NAS experiments. Wan was responsible for implementing the our method for experiments and generating plots in the interpretability section.		
Signature		Date	17/09/2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof. Michael A. Osborne		
Supervisor comments To the best of my knowledge, all above seems fair and correct.		
Signature 	Date	17/09/2021

This completed form should be included in the thesis, at the end of the relevant chapter.

8

Speedy Performance Estimation for Neural Architecture Search

The content of this chapter is based on the following paper:

> Binxin Ru*, Clare Lyle*, Lisa Schut, Mark van der Wilk, and Yarin Gal. Speedy Performance Estimation for Neural Architecture Search. In *arXiv preprint arXiv:2006.04492 (Under review of NeurIPS 2021)*, 2021.

where our contributions were listed in the acknowledgements section of this thesis.

Evaluating the generalisation performance of a new configuration is the most cost-intensive procedure in AutoML processes. Thus, efficient performance estimation is crucial to the successful deployment of AutoML algorithms in practical applications, and necessary for making AutoML affordable and accessible to a larger pool of potential users. Following Chapter 7, in this chapter, we studies the problem of performance estimation in the context of neural architecture search (NAS).

Traditional approaches to evaluate test performance of an new architecture face a variety of limitations: training each architecture to completion is prohibitively expensive, early stopped validation accuracy may correlate poorly with fully trained performance, and model-based estimators require large training sets. We instead

propose to estimate the final test performance based on a simple measure of training speed. Our estimator is theoretically motivated by the connection between generalisation and training speed, and is also inspired by the marginal likelihood, which is used for Bayesian model selection. Our model-free estimator is simple, efficient, and cheap to implement, and does not require hyper-parameter-tuning or surrogate training before deployment. We demonstrate on various NAS search spaces that our estimator consistently outperforms other alternatives in achieving better correlation with the true test performance rankings. We further show that our estimator can be easily incorporated into both query-based and one-shot NAS methods to improve the speed or quality of the search.

8.1 Introduction

Reliably estimating the generalisation performance of a proposed architecture is crucial to the success of Neural Architecture Search (NAS) but has always been a major bottleneck in NAS algorithms (Elsken et al., 2018). The traditional approach of training each architecture for a large number of epochs and evaluating it on validation data (*full training*) provides a reliable performance measure, but requires prohibitively large computational resources on the order of thousands of GPU days (Zoph and Le, 2017; Real et al., 2017b; Zoph et al., 2018b; Real et al., 2019; Elsken et al., 2018). This motivates the development of methods for speeding up performance estimation to make NAS practical for limited computing budgets.

A popular simple approach is *early-stopping*, which offers a low-fidelity approximation of generalisation performance by training for fewer epochs (Li et al., 2016; Falkner et al., 2018; Li and Talwalkar, 2019). However, if we stop training early and evaluate the model on validation data, the relative performance ranking may not correlate well with the performance ranking of the full training (Zela et al., 2018), i.e. the test performance of the model after the entire trainin budget has been used. Another line of work focuses on *learning curve extrapolation* (Domhan et al.; Klein et al., 2016b; Baker et al., 2017), which trains a surrogate model to predict the final generalisation performance based on the initial learning curve and/or meta-features

of the architecture. However, the training of the surrogate often requires hundreds of fully evaluated architectures to achieve satisfactory extrapolation performance and the hyper-parameters of the surrogate also need to be optimised. Alternatively, the idea of *weight sharing* is adopted in one-shot NAS methods to speed up evaluation (Pham et al., 2018; Liu et al., 2019; Xie et al., 2019b). Despite leading to significant cost-saving, weight sharing heavily underestimates the true performance of good architectures and is unreliable in predicting the relative ranking among architectures (Yang et al., 2020; Yu et al., 2020). A more recent group of estimators claim to be zero-cost (Mellor et al., 2020; Abdelfattah et al., 2021). Yet, their performance is often not competitive with the state of the art, inconsistent across tasks and cannot be further improved with additional training budgets.

In view of the above limitations, we propose a simple model-free method, Training Speed Estimation (TSE), which provides a reliable yet computationally cheap estimate of the generalisation performance ranking of architectures. Our method is inspired by recent empirical and theoretical results linking training speed and generalisation (Hardt et al., 2016; Lyle et al., 2020) and measures the training speed of an architecture by summing the training losses of the commonly-used SGD optimiser during training. We empirically show that our estimator can outperform strong existing approaches to predict the relative performance ranking among architectures, and can remain effective for a variety of search spaces and datasets. Moreover, we verify its usefulness under different NAS settings and find it can speed up query-based NAS approaches significantly as well improve the performance of one-shot and differentiable NAS.

8.2 Training Speed Estimation

8.2.1 Theoretical Motivations

Training Speed The theoretical relationship between training speed and generalisation is described in a number of existing works. Stability-based generalisation bounds for SGD (Hardt et al., 2016; Liu et al., 2017) bound the generalisation gap of a model based on the number of optimisation steps used to train it. These bounds

predict that models which train faster obtain a lower worst-case generalisation error. In networks of sufficient width, a neural-tangent-kernel-inspired complexity measure can bound both the worst-case generalisation gap and the rate of convergence of (full-batch) gradient descent (Arora et al., 2019; Cao and Gu, 2019). However, these bounds cannot distinguish between models that are trained for the same number of steps but attain near-zero loss at different rates as they ignore the training trajectory.

We instead draw inspiration from another approach which incorporates properties of the trajectory taken during SGD, seen in the information-theoretic generalisation bounds of (Negrea et al., 2019) and (Neu, 2021). These bounds depend on the variance of gradients computed during training, a quantity which provides a first-order approximation of training speed by quantifying how well gradient updates computed for one minibatch generalize to other points in the training set. In view of this link, the empirical and theoretical findings in recent work (Fort et al., 2019; Smith et al., 2021), which show a notion of the variance of gradients over the training data is correlated with generalisation, become another piece of supporting evidence for training speed as a measure of generalisation.

Bayesian Marginal Likelihood ,

Besides training speed, our estimator is also theoretically inspired by the marginal likelihood, which is the basis for Bayesian model selection and has been briefly discussed in Section 2.1.2 of Chapter 2. The marginal likelihood quantifies how likely a dataset $\mathcal{D}_n$ is to have been generated by a model, and so can be used to update a prior belief distribution over which model from a given set is most likely to have generated $\mathcal{D}_n$. Given a model with parameters $\mathbf{w}$, prior $p(\mathbf{w})$, and likelihood $p(\mathcal{D}_n|\mathbf{w})$ for a training data set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n = \{\mathbf{X}_n, \mathbf{y}_n\}$, the (log) marginal likelihood can be expressed as follows.

$$\log p(\mathcal{D}_n) = \log \mathbb{E}_{p(\mathbf{w})}\left[p(\mathcal{D}_n|\mathbf{w})\right] \Leftrightarrow \log p(\mathcal{D}_n) = \sum_{i=1}^{n} \log p(\mathbf{x}_i, y_i|\mathcal{D}_{<i}) = \sum_{i=1}^{n} \log \left[\mathbb{E}_{p(\mathbf{w}|\mathcal{D}_{<i})}\left[p(\mathbf{x}_i, y_i|\mathbf{w})\right]\right]$$

Interpreting the negative log posterior predictive probability $-\log p(\mathbf{x}_i, y_i|\mathcal{D}_{<i})$ of each data point as a 'loss' function, the log marginal likelihood then corresponds to the area under a training loss curve, where each training step would be computed by sampling a data point $(\mathbf{x}_i, y_i)$, taking the log expected likelihood under the current posterior $p(\mathbf{w}|\mathcal{D}_{<i})$ as the current loss, and then updating the posterior by incorporating the new sampled data point: $\mathcal{D}_{<i+1} := \mathcal{D}_{<i} \cup \{(\mathbf{x}_i, y_i)\}$. One can therefore interpret the marginal likelihood as a measure of training speed in a Bayesian updating procedure.

In the setting where we cannot compute the posterior analytically and only samples $\hat{\mathbf{w}}$ from the posterior over parameters are available, we obtain an unbiased estimator of a lower bound $\mathcal{L}(\mathcal{D}_n)$ on the marginal likelihood by Jensen's inequality,

$$\begin{aligned}
\log p(\mathcal{D}_n) &= \sum_{i=1}^{n} \log \left[\mathbb{E}_{p(\mathbf{w}|\mathcal{D}_{<i})} \left[p(\mathbf{x}_i, y_i|\mathbf{w})\right]\right] \\
&\geq \sum_{i=1}^{n} \mathbb{E}_{p(\mathbf{w}|\mathcal{D}_{<i})} \left[\log p(\mathbf{x}_i, y_i|\mathbf{w})\right] \\
&\approx \sum_{i=1}^{n} \log p(\mathbf{x}_i, y_i|\hat{\mathbf{w}}) = \mathcal{L}(\mathcal{D}_n)
\end{aligned}$$

with $\approx$ denoting equality in expectation. The unbiased estimator of a lower bound $\mathcal{L}(\mathcal{D}_n)$ again corresponds to minimising a sum over training losses

Although not included in this thesis, we conduct a full analysis of the Bayesian setting in Lyle et al. (2020) where we prove for linear models and infinitely wide deep models performing Bayesian updates, the marginal likelihood can be bounded by a notion of training speed (i.e. sum of training losses). In the NAS setting, the deep neural networks does not follow Bayesian updates but we argue that the training speed estimation metric is still *plausibly* useful for model selection. Just as the marginal likelihood measures the utility of updates based on early data points in predicting later data points, the TSE of a model trained with SGD will be lower for models whose mini-batch gradient descent updates improve the loss of later mini-batches seen during optimisation. We would like to emphasise that the Bayesian connection thus justifies the use of a *sum* over training losses, instead of the training loss from a single mini-batch update, as a tool for model selection.

8.2.2 Method Description

The above theoretical motivations suggest that leveraging a notion of training speed may benefit model selection procedures in NAS. Many such notions exist in the generalisation literature: (Jiang et al., 2020) count the number of optimisation steps needed to attain a loss below a specified threshold, while (Hardt et al., 2016) consider the total number of optimisation steps taken. Both measures are strong predictors of generalisation after training, yet neither is suitable for NAS, where we seek to stop training as early as possible if the model is not promising.

We draw inspiration from the Bayesian marginal likelihood discussed above and present an alternate estimator of training speed that amounts to the area under the model's training curve. Models that train quickly attain a low loss after few training steps, and so will have a lower area-under-curve than those which train slowly. This addresses the shortcomings of the previous two methods as it is able to distinguish models both early and late in training.

We now define our proposed performance estimator. Let ℓ denote a loss function, $f_{\mathbf{w}}(\mathbf{x})$ the output of a neural network f with input $\mathbf{x}$ and parameters $\mathbf{w}$, and let $\mathbf{w}_{t,i}$ denote the parameters of the network after t epochs and i minibatches of SGD. After training the network for T epochs[1], we sum the training losses collected so far to get the following *Training Speed Estimate* (TSE):

$$\text{TSE} = \sum_{t=1}^{T} \left[\frac{1}{B} \sum_{i=1}^{B} \ell \left(f_{\mathbf{w}_{t,i}}(\mathbf{X}_i), \mathbf{y}_i \right) \right] \tag{8.1}$$

where l is the training loss of a mini-batch $(\mathbf{X}_i, \mathbf{y}_i)$ at epoch t and B is the number of training steps within an epoch.

This estimator weights the losses accumulated during every epoch equally. However, recent work suggests that training dynamics of neural networks in the very early epochs are often unstable and not always informative of properties of the converged networks (Lewkowycz et al., 2020). Therefore, we hypothesise that an estimator of network training speed that assigns higher weights to later epochs may exhibit a better correlation with the true generalisation performance of the final

[1] T can be far from the total training epochs T_{end} used in complete training

trained network. On the other hand, it is common for neural networks to overfit on their training data and reach near-zero loss after sufficient optimisation steps, so attempting to measure training speed *solely* based on the epochs near the end of training will be difficult and likely suffer degraded performance on model selection.

To verify whether it is beneficial to ignore or downplay the information from early epochs of training, we propose two variants of our estimator. The first, **TSE-E**, treats the first few epochs as a burn-in phase for $\mathbf{w}_{t,i}$ to converge to a stable distribution $P(\mathbf{w})$ and starts the sum from epoch $t = T - E + 1$ instead of $t = 1$. In the case where $E = 1$, we start the sum at $t = T$ and our estimator corresponds to the sum over training losses within the most recent epoch $t = T$.

$$\text{TSE-E} = \sum_{t=T-E+1}^{T} \left[\frac{1}{B} \sum_{i=1}^{B} \ell \left(f_{\mathbf{w}_{t,i}}(\mathbf{X}_i), \mathbf{y}_i \right) \right],$$

$$\text{TSE-EMA} = \sum_{t=1}^{T} \gamma^{T-t} \left[\frac{1}{B} \sum_{i=1}^{B} \ell \left(f_{\mathbf{w}_{t,i}}(\mathbf{X}_i), \mathbf{y}_i \right) \right]$$

The second, **TSE-EMA**, does not completely discard the information from the early training trajectory but takes an exponential moving average of the sum of training losses with $\gamma = 0.9$, thus assigning higher weight to the sum of losses obtained in later training epochs.

We empirically show in Section 8.4.2 that our proposed TSE and its variants (TSE-E and TSE-EMA), despite their simple form, can reliably estimate the generalisation performance of neural architectures with a very small training budget, can remain effective for a large range of training epochs, and are robust to the choice of hyper-parameters such as the summation window E and the decay rate γ. However, our estimator is *not* meant to replace the validation accuracy at the end of training or when the user can afford large training budget to sufficient train the model. In those settings, validation accuracy remains as the gold standard for evaluating the true test performance of architectures. Ours is just a speedy performance estimator for NAS, aimed at giving an indication early in training about an architecture's generalisation potential under a fixed training set-up.

Our choice of using the training loss, instead of the validation loss, to measure training speed is an important component of the proposed method. While it is

possible to formulate an alternative estimator, which sums the validation losses of a model early in training, this estimator would no longer be measuring *training speed*. In particular, such an estimator would not capture the generalisation of gradient updates from one minibatch to later minibatches in the data to the same extent as TSE does. Indeed, we hypothesise that once the optimisation process has reached a local minimum, the sum over validation losses more closely resembles a variance-reduction technique that estimates the expected loss over parameters sampled via noisy SGD steps around this minimum. We show in Section 8.4.2 and 8.4.6 that our proposed sum over training losses (TSE) outperforms the sum over validation losses (SoVL) in ranking models in agreement with their true test performance.

8.3 Related Work

Various approaches have been developed to speed up architecture performance estimation, thus improving the efficiency of NAS. Low-fidelity estimation methods accelerate NAS by using the validation accuracy obtained after training architectures for fewer epochs (namely early-stopping) (Li et al., 2016; Falkner et al., 2018; Zoph et al., 2018b; Zela et al., 2018), training a down-scaled model with fewer cells during the search phase (Zoph et al., 2018b; Real et al., 2019), or training on a subset of the data (Klein et al., 2016a). However, low-fidelity estimates underestimate the true performance of the architecture and can change the relative ranking among architectures (Elsken et al., 2018). This undesirable effect on relative ranking is more prominent when the cheap approximation set-up is too dissimilar to the full training (Zela et al., 2018). As shown in Figure 8.3 below, the validation accuracy at early epochs of training suffers low rank correlation with the final test performance. Another class of performance estimation methods trains a regression model to extrapolate the learning curve from what is observed in the initial phase of training. Regression model choices that have been explored include Gaussian processes with a tailored kernel function (Domhan et al.), an ensemble of parametric functions (Domhan et al.), a Bayesian neural network (Klein et al., 2016b) and more recently a

ν-support vector machine regressor (ν-SVR)(Baker et al., 2017) which achieves state-of-the-art prediction performance (White et al., 2021b). Although these model-based methods can often predict the performance ranking better than their model-free early-stopping counterparts, they require a relatively large amount of fully evaluated architecture data (e.g. 100 fully evaluated architectures in Baker et al. (2017)) to train the regression surrogate properly and optimise the model hyper-parameters in order to achieve a good prediction performance. The high computational cost of collecting the training set makes such model-based methods less favourable for NAS unless the practitioner has already evaluated hundreds of architectures on the target task. Moreover, both low-fidelity estimates and learning curve extrapolation estimators are empirically developed and lack theoretical motivation.

Weight sharing is employed in one-shot or gradient-based NAS methods to reduce computational costs (Pham et al., 2018; Liu et al., 2019; Xie et al., 2019b). Under the weight-sharing setting, all architectures are considered as subnetworks of a supernetwork. Only the weights of the supernetwork are trained while the architectures (subnetworks) inherit the corresponding weights from the supernetwork. This removes the need for retraining each architecture during the search and thus achieves a significant speed-up. However, the weight sharing ranking among architectures often correlates poorly with the true performance ranking (Yang et al., 2020; Yu et al., 2020; Zela et al., 2020), meaning architectures chosen by one-shot NAS are likely to be sub-optimal when evaluated independently (Zela et al., 2020). In Section 8.4.5, we demonstrate that we improve the performance of weight sharing in correctly ranking architectures by combining our estimator with it. Moreover, one-shot methods are often outperformed by sample-based NAS methods (Dong and Yang, 2020a; Zela et al., 2020).

Recently, several works propose to estimate network performance without training by using methods from the pruning literature (Abdelfattah et al., 2021) or examining the covariance of input gradients across different input images (Mellor et al., 2020). Such methods incur near-zero computational costs but their performances are often not competitive and do not generalise well to larger search

spaces, as shown in Section 8.4.2 below. Moreover, these methods can not be improved with additional training budget.

Apart from the above mentioned performance estimators used in NAS, many complexity measures have been proposed to analyse the generalisation performance of deep neural networks. (Jiang et al., 2020) provides a rigorous empirical analysis of over 40 such measures. This investigation finds that sharpness-based measures (McAllester, 1999; Keskar et al., 2016; Neyshabur et al., 2017; Dziugaite and Roy, 2017) (including PAC-Bayesian bounds) obtain a good correlation with test set performance, but their estimation require adding randomly generated perturbations to the network parameters and the magnitude of the perturbations needs to be carefully optimised with additional training, making them unsuitable performance estimators for NAS. Optimisation-based complexity measures, which counts the number of steps required to reach a certain loss value, also perform well in predicting generalisation. However, as discussed in Section 8.2, it is closely related to our approach but not as easy to deploy as our estimators under the NAS setting.

8.4 Experiments

In this section, we first evaluate the quality of our proposed estimators in predicting the generalisation performance of architectures against a number of baselines (Section 8.4.2), and then demonstrate that simple incorporation of our estimators can significantly improve the search speed and/or quality of both query-based and weight-sharing NAS (Sections 8.4.4 and 8.4.5).

We measure the true generalisation performance of architectures with their final test accuracy after being completely trained for T_{end} epochs. To ensure fair assessment of the architecture performance only, we adopt the common NAS protocol where all architectures searched/compared are trained and evaluated under the *same* set of hyper-parameters. Also following (Ying et al., 2019) and (Dong and Yang, 2020a), we compare different estimators based on their Spearman's rank correlation which measures how well their predicted ranking correlates with the true test ranking among architectures.

We compare the following performance estimation methods: our proposed estimators **TSE**, **TSE-EMA** and **TSE-E** described in Section 8.2 and simply **the training losses at each mini batch (TLmini)**. **Sum of validation losses over all preceding epochs (SoVL)**[2] is similar to TSE but uses the *validation* losses. **Validation accuracy at an early epoch (VAccES)** corresponds to the early-stopping practice whereby the user estimates the final test performance of a network using its validation accuracy at an early epoch $T < T_{\text{end}}$. **Learning curve extrapolation (LcSVR)** method is the state-of-the-art extrapolation method proposed in Baker et al. (2017) which trains a ν-SVR on previously evaluated architecture data to predict the final test accuracy of new architectures. The inputs for the SVR regression model comprise architecture meta-features (e.g. number of parameters), training hyper-parameters (e.g. initial learning rate), and learning curve features up to epoch T (e.g. the validation curve and its 1st-oder and 2nd-order differences up to epoch $t = T$). In our experiments, we optimise the SVR hyper-parameters via cross-validation following (Baker et al., 2017). Three recently proposed zero-cost baselines are also included: an estimator based on **input Jacobian covariance (JavCov)** (Mellor et al., 2020) and two adapted from pruning techniques **SNIP** and **SynFlow** (Abdelfattah et al., 2021).

We run experiments on architectures generated from a diverse set of NAS search spaces listed in Table 8.1 to show that our estimators generalise well (more details on the datasets are provided in Appendix B.5). Note $N_{samples} = 6466$ for NASBench-201 (NB201) as it's the number of unique architectures in the space. We use the architecture information released in NAS-Bench-301 (Siems et al., 2020) for DARTS and in (Radosavovic et al., 2019) for ResNet and ResNeXt. As for RandWiredNN (RWNN) search space (Xie et al., 2019a; Ru et al., 2020b), although the number of possible randomly wired architectures are immense, they are generated via a random graph generator which is defined by 3 hyper-parameters. We thus uniformly sampled 69 sets of hyper-parameter values for the generator and generated 8 randomly wired

[2]Note, we flip the sign of TSE/TSE-EMA/TSE-E/SoVL/TLmini (which we want to minimise) to compare to the Spearman's rank correlation of the other methods (which we want to maximise).

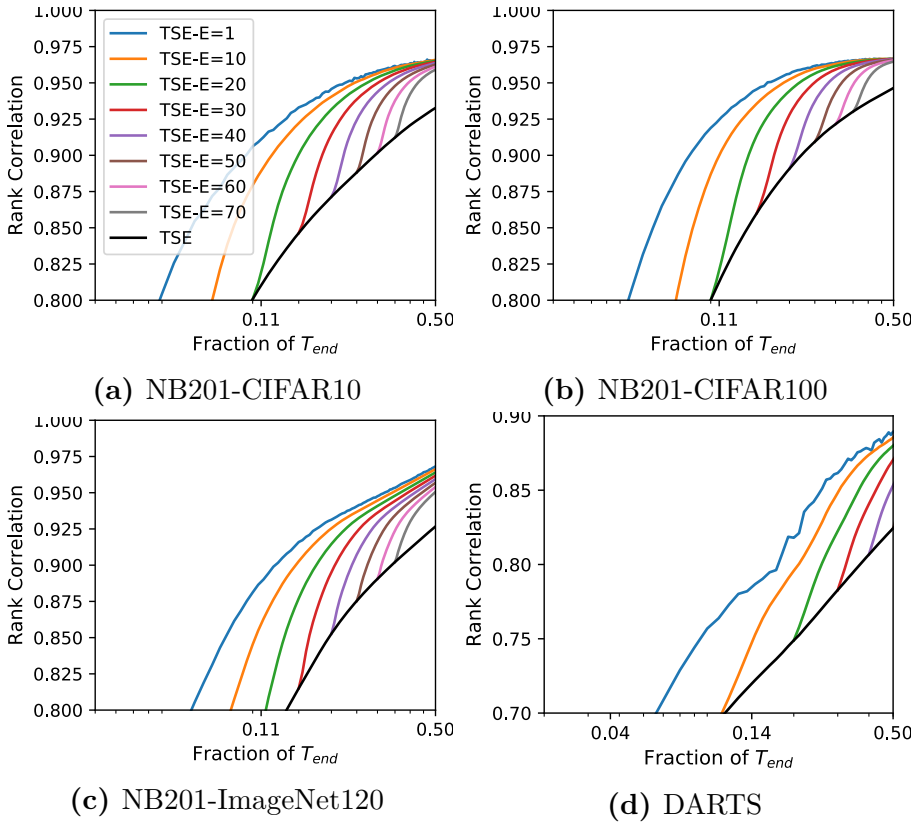
Table 8.1: NAS search spaces used. The true test accuracy of architectures from each search space is obtained after training with SGD on the corresponding image datasets for T_{end} epochs. N_{total} denotes the total possible architectures exist in the search space and $N_{samples}$ denotes the number of architectures we sample/generate for our experiments.

Search space	T_{end}	$N_{samples}$	N_T	Image datasets
NASBench-201 (NB201) (Dong and Yang, 2020a)	200	6466	15625	CIFAR10, CIFAR100, ImageNet120
DARTS (Liu et al., 2019; Siems et al., 2020)	100	5000	$\mathcal{O}(2^{42})$	CIFAR10
ResNet/ResNeXt (Radosavovic et al., 2019)	100	50000	$\mathcal{O}(2^{26})$	CIFAR10
RandWiredNN (RWNN) (Xie et al., 2019a)	250	69×8	$\mathcal{O}(2^{378})$	Flower102

neural networks from each hyper-parameter value, leading to $N_{samples} = 69 \times 8 = 552$. All the experiments were conducted on an internal cluster of 16 RTX2080 GPUs.

8.4.1 Hyper-parameters of TSE Estimators

Our proposed TSE estimators require very few hyper-parameters: the summation window size E for TSE-E and the decay rate γ for TSE-EMA, and we show empirically that our estimators are robust to these hyper-parameters. For the former, we test different summation window sizes on various search spaces and image datasets in Figure 8.1 and find that $E = 1$ consistently gives the best results across all cases. This, together with the almost monotonic improvement of our estimator's rank correlation score over the training budgets, supports our hypothesis discussed in Section 8.2 that training information in the more recent epochs is more valuable for performance estimation. Note that TSE-E with $E = 1$ corresponds to the sum of training losses over all the batches in one single epoch. As for γ, we show in Figure 8.2 that TSE-EMA is robust to a range of popular choices $\gamma \in [0.9, 0.95, 0.99, 0.999]$ across various datasets and search spaces. Specifically, the performance difference among these γ values are almost indistinguishable compared to the difference



(a) NB201-CIFAR10 **(b)** NB201-CIFAR100

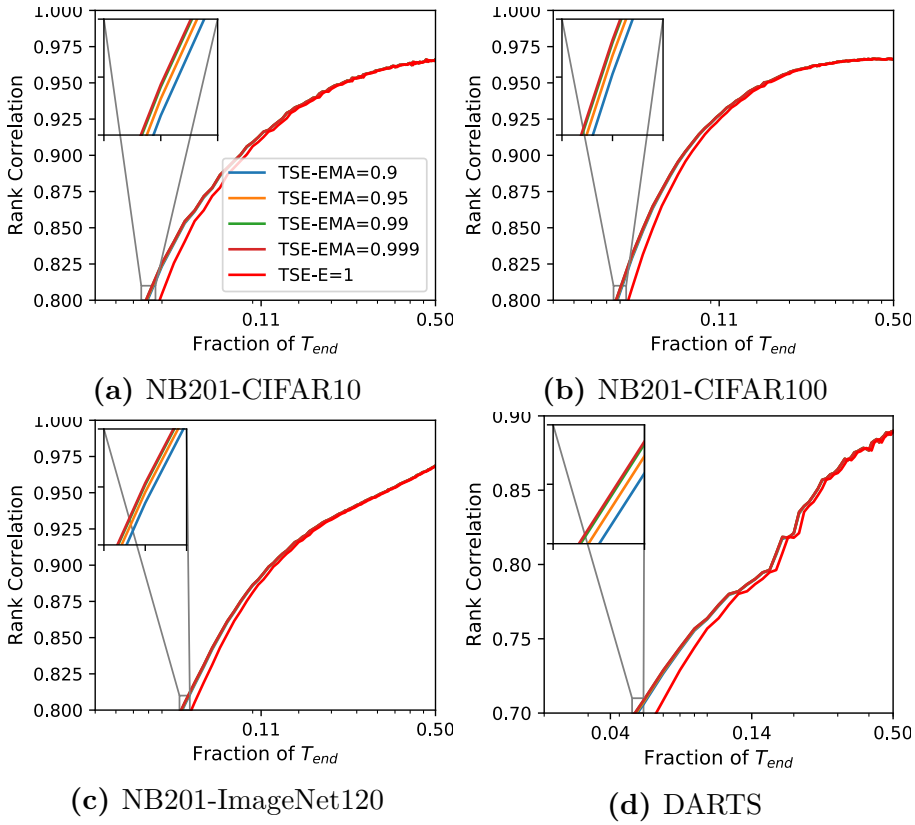
(c) NB201-ImageNet120 **(d)** DARTS

Figure 8.1: Rank correlation performance of TSE-E computed over E most recent epochs. Different E values are investigated for architectures in NASBench-201 (NB201) on three image datasets and 5000 architectures from NASBench-301 (DARTS) on CIFAR10. In all cases, smaller E consistently achieves better rank correlation performance with $E = 1$ being the best choice.

between TSE-EMA and TSE-E. Thus, we set $E = 1$ and $\gamma = 0.999$ in all the following experiments and recommend them as the default choice for potential users who want to apply TSE-E and TSE-EMA on a new task without additional tuning.

8.4.2 Comparison of Performance Estimation Quality

Robustness across different NAS search spaces We now compare our TSE estimators against a variety of other baselines. To mimic the realistic NAS setting (Elsken et al., 2018), we assume that all the estimators can only use the information from early training epochs and limit the maximum budget to $T \leq 0.5T_{\text{end}}$ in this set of experiments. This is because NAS methods often need to evaluate hundreds of architectures or more during the search (Ru et al., 2020b) and thus rarely use evaluation budget beyond $0.5T_{\text{end}}$ so as to keep the search cost practical/affordable.



(a) NB201-CIFAR10 **(b)** NB201-CIFAR100

(c) NB201-ImageNet120 **(d)** DARTS

Figure 8.2: Rank correlation performance of TSE-EMA with γ on architectures in NASBench-201 (NB201) on three image datasets and 5000 architectures from NASBench-301 (DARTS) on CIFAR10. In all cases, TSE-EMA is very robust to different γ values; the difference among TSE-EMA with different γ is indistinguishable compared to that with TSE-E.

The results on a variety of the search spaces are shown in Figure 8.3. Our proposed estimator TSE-EMA and TSE-E, despite their simple form and cheap computation, outperform all other methods under limited evaluation budget $T < 0.5T_{\text{end}}$ for all search spaces and image datasets. They also remain very competitive on ranking among the top 1% ResNet/ResNeXt architectures as shown in Figure 8.3(f) and (g). TSE-EMA achieves superior performance over TSE-E especially when T is small. This suggests that although the training dynamics at early epochs might be noisy, they still carry some useful information for explaining the generalisation performance of the network. The learning curve extrapolation method, LcSVR, is competitive. However, the method requires 100 fully trained architecture data to fit the regression surrogate and optimise its hyperparamters via cross validation; a large amount of computational resources are needed to collect these training

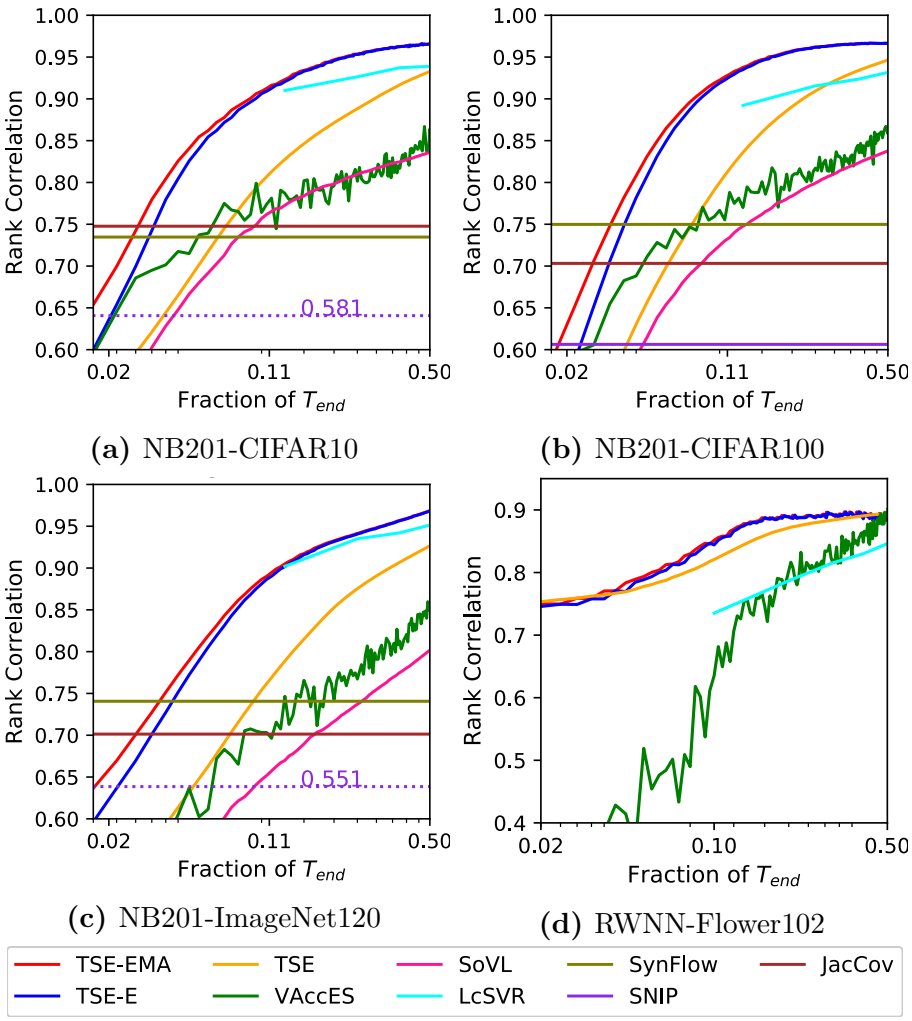


Figure 8.3: Rank correlation performance of various baselines for (a) to (c) NB201 architectures on three image datasets and for (d) RWNNs on Flowers102. In all cases, our TSE-EMA and TSE-E achieve superior rank correlation with the true test performance in much fewer epochs than other baselines. In (a) and (c), we mark SNIP in a violet dotted line labelled with its rank correlation value as it falls out of the plotted range.

data in practice. The zero-cost measures JacCov and SynFlow achieve good rank correlation at initialisation but is quickly overtaken by TSE-EMA and TSE-E once the training budget exceeds 6-7 epochs. SNIP performs poorly and falls out of the plot range in Figure 8.3 (a) and 8.3 (c).

We further validate on the architectures from the more popular search space used in DARTS. One potential concern is that if models are trained using different hyper-parameters that influence the learning curve (e.g. learning rate), the prediction performance of our proposed estimators will be affected. However, this is not a

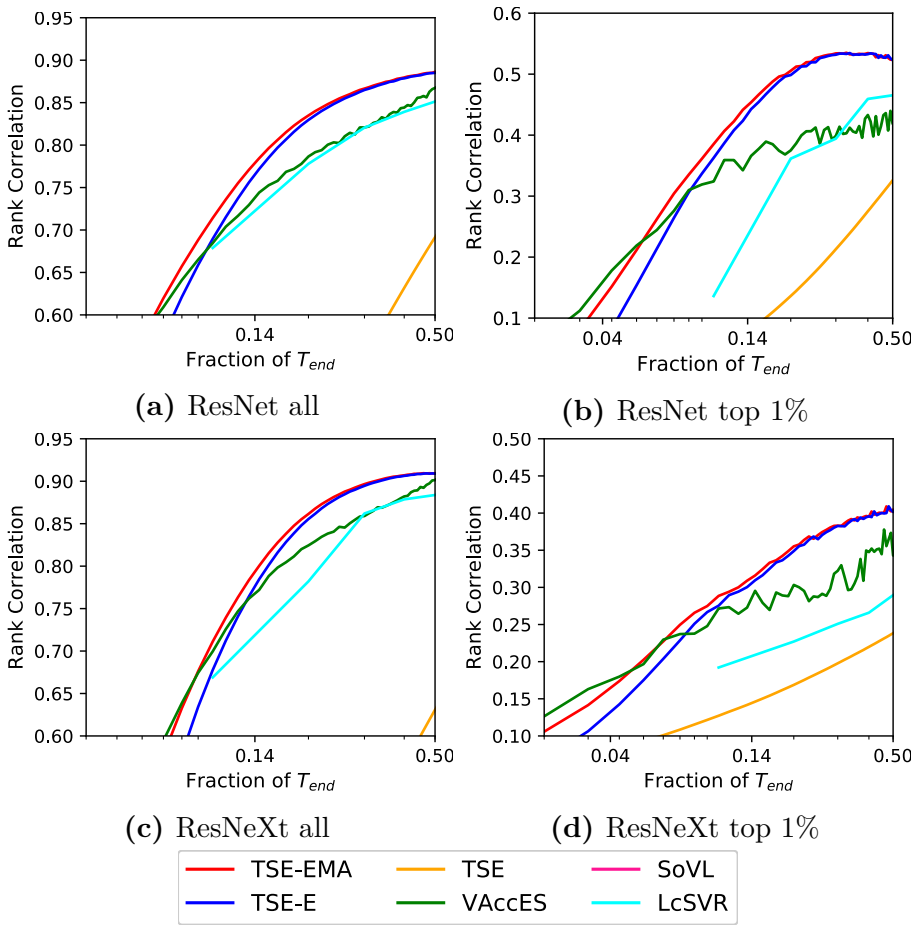
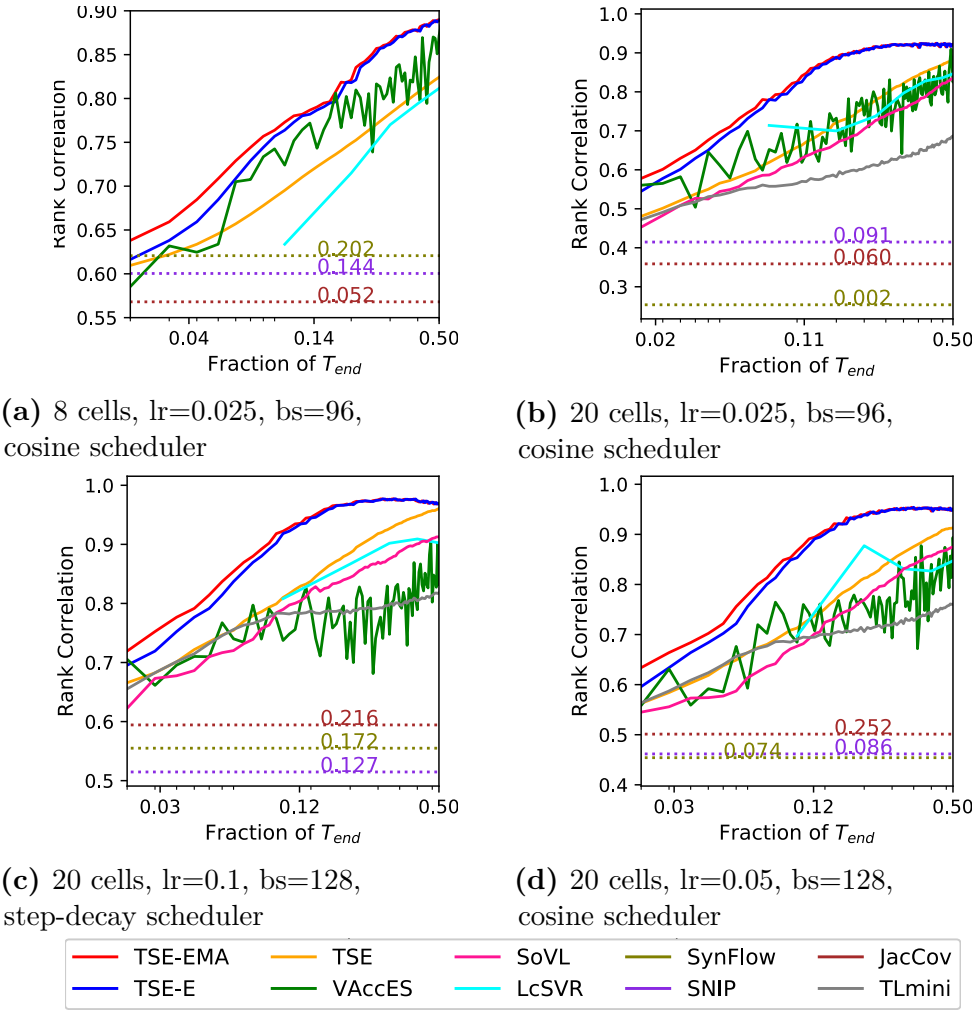


Figure 8.4: Rank correlation performance of various baselines for ResNet and ResNeXt architectures on CIFAR10. In (b) and (d), we evaluate estimators on the top 1% of the ResNet/ResNeXt architectures and show that our TSE-EMA and TSE-E can remain competitive on ranking among top architectures, which are particularly desirable for NAS.

problem in NAS because almost all existing NAS methods (Dong and Yang, 2020a; Ying et al., 2019; Xie et al., 2019a; Liu et al., 2019; Siems et al., 2020; White et al., 2021b) search for the optimal architecture under a fixed set of training hyper-parameters. We also follow this *fixed-hyperparamter* set-up in our work. Verifying the quality of various estimators for predicting the generalisation performance across *different hyper-parameters* lies outside the scope of our work but would be interesting for future work.

Robustness across different NAS set-ups Here, we conduct experiments to verify the robustness of our estimators across different NAS set-ups. On top of the architecture data from NAS-Bench-301 (Siems et al., 2020), we also generate



(a) 8 cells, lr=0.025, bs=96, cosine scheduler

(b) 20 cells, lr=0.025, bs=96, cosine scheduler

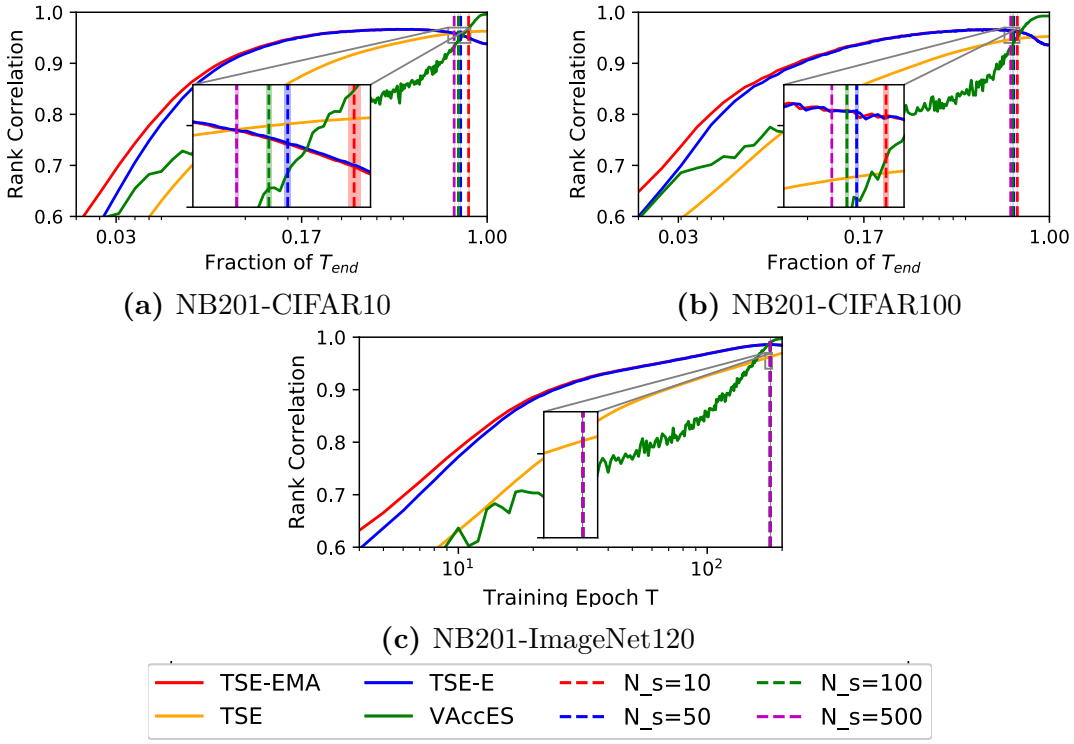
(c) 20 cells, lr=0.1, bs=128, step-decay scheduler

(d) 20 cells, lr=0.05, bs=128, cosine scheduler

Figure 8.5: Rank correlation performance of various baselines for 5000 small 8-cell architectures (a) and 150 large 20-cell architectures (b) to (d) from DARTS search space on CIFAR10. We use NAS-Bench-301 dataset(NAS301) for computing (a) and for large architectures, we test three training hyper-parameter set-ups with different initial learning rates, learning rate schedulers and batch sizes as denoted in the subcaptions. On all four settings, our TSE-E again consistently achieves superior rank correlation in fewer epochs than other baselines. Note all three zero-cost estimators perform poorly (below the plotted range) on DARTS search space across all settings. We denote them in dotted lines with their rank correlation value labelled.

several additional architecture datasets; each dataset correspond to a different set-up (e.g. different architecture depth, initial learning rate, learning rate scheduler and batch size) and contains 150 large 20-cell architectures which are randomly sampled from the DARTS space and evaluated on CIFAR10. The results in Figure 8.5 show that our estimator consistently outperforms all the competing methods in comparing architectures under different NAS set-ups. Note here the curve of

TLmini corresponds to the *average* rank correlation between final test accuracy and the mini-batch training loss over the epoch. The clear performance gain of our TSE estimators over TLmini supports our claim that it is the sum of training losses, which measures the training speed and thus carries the theoretical motivations explained in Section 8.2, instead of simply the training loss at a single minibatch, that gives a good estimation of generalisation performance. Note the rank correlation of all zero-cost measures drop significantly (e.g. SynFlow drops from 0.74 on NB201 to below 0.2) on the DARTS search space, and even do worse than the training losses at the first few minibatches (TLmini at T=1). Such inconsistent prediction performance, especially doing weakly on the more practical search space, is undesirable for real-world NAS applications.



(a) NB201-CIFAR10 **(b)** NB201-CIFAR100

(c) NB201-ImageNet120

Figure 8.6: Rank correlation performance up to $T = T_{\text{end}}$. If the users want to apply our estimators for large training budget, they can estimate the effective range of our estimators based on the minimum epoch T_o when overfitting happens among the N_s observed architectures. They can then stop our estimators early at $0.9T_o$(marked by vertical lines) or switch back to validation accuracy beyond that.

Algorithm 12 Find Effective Training Budget for TSE Estimators

Input: A subset of N_s fully trained architectures whose training losses are $\{\{\ell_{i,t}\}_{t=1}^{T_{\text{end}}}\}_{i=1}^{N_s}$, Overfitting criterion $is\_overfit()$
Output: The effective training budget $T_{effective}$ to use TSE-EMA or TSE-E
$\mathcal{S}_{T_o} = \emptyset$
for $i = 1, \ldots, N_s$ **do**
  for $t = 1, \ldots, T_{\text{end}}$ **do**
    if `is_overfit`$(\ell_{i,t})$ **then**
      $T_{i,o} = T$
      break
    else
      $T_{i,o} = T_{\text{end}}$
    end if
  end for
  $\mathcal{S}_{T_o} = \mathcal{S}_{T_o} \cup T_{i,o}$
end for
$T_o = \min \mathcal{S}_{T_o}$
$T_{effective} = 0.9T_o$

Procedure to decide the effective training budget We include three examples showing the performance of our estimator for training budgets beyond $0.5T_{\text{end}}$ in Figure 8.6. Our estimators can remain superior for a relatively wide range of training budgets. Although they will eventually be overtaken by validation accuracy as the training budget approaches T_{end} as discussed in Section 8.2, the region requiring large training budget is less interesting for NAS where we want to maximise the cost-saving by using performance estimators. However, if the user wants to apply our estimators with a relatively large training budget, we propose a simple method here to estimate when our estimators would be less effective than validation accuracy. We notice that that our estimators, TSE-EMA and TSE-E, become less effective when the architectures compared start to overfit because both of them rely heavily on the lastest-epoch training losses to measure training speed, which is difficult to estimate when the training losses become too small.

Thus, we can adopt the simple heuristic described in Algorithm 12 to decide when to stop the computation of TSE-E and TSE-EMA early by reverting to a previous checkpoint at $T_{\text{effective}} = 0.9T_o$ [3]. Similar to Appendix A.8.1, we use a

[3]We use $T_{\text{effective}} = 0.9T_o$ rather than T_o to be conservative.

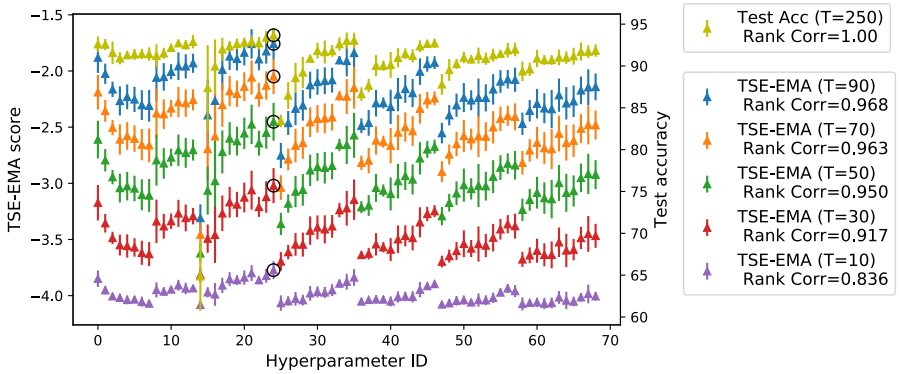
simple criterion: training loss decreasing below 0.1 (i.e. $\ell_{i,t=T_o} < 0.1$), to identify when an architecture start to overfit on NB201 datasets. We randomly sample $N_s = 10, 50, 100, 500$ architectures and assume that we have access to their full learning curves. We then decide the threshold training budget $T_{\text{effective}}$ (vertical lines) as the minimum training epoch that overfitting happens among these N_s architectures. We repeat this for 100 random seeds and plot the mean and standard error of the threshold for each N_s in Figure 8.6. It's evident that we can find a fairly reliable threshold with a sample size as small as $N_s = 10$.

8.4.3 Architecture Generator Selection

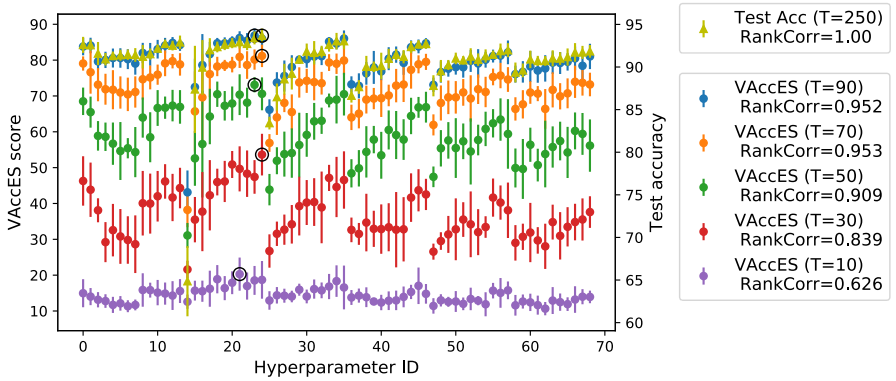
For the RWNN-Flower102 dataset, we use 69 different hyper-parameter values for the random graph generator which generates the randomly wired neural architecture. Here we would like to investigate whether our estimator can be used in place of the true test accuracy to select among different hyper-parameter values. For each graph generator hyper-parameter value, we sample 8 neural architectures with different wiring. The mean and standard error of the true test accuracies, TSE-EMA scores and early stopped validation accuracy (VAccES) over the 8 samples are presented in Figure 8.7. Our estimator can accurately predict the relative performance ranking among different hyper-parameters (Rank correlation ≥ 0.84) and accurately identify the optimal hyper-parameter (circled in black) based on as few as 10 epochs of training ($T = 10$). The prediction by VAccES is less consistent and accurate and the rank correlation between VAccES and the final test accuracy is always lower than that of our TSE-EMA across different training budgets.

8.4.4 Speed up Query-based NAS

In this section, we demonstrate the usefulness of our estimator for NAS by incorporating TSE-EMA, at $T = 10$ into several query-based NAS search strategies: Regularised Evolution (Real et al., 2019) in Figure 8.8(a), Bayesian Optimisation (Bergstra et al., 2011b) in Figure 8.8(b) and Random Search (Bergstra and Bengio, 2012) Figure 8.8(c). We perform architecture search on NB201 search space. We



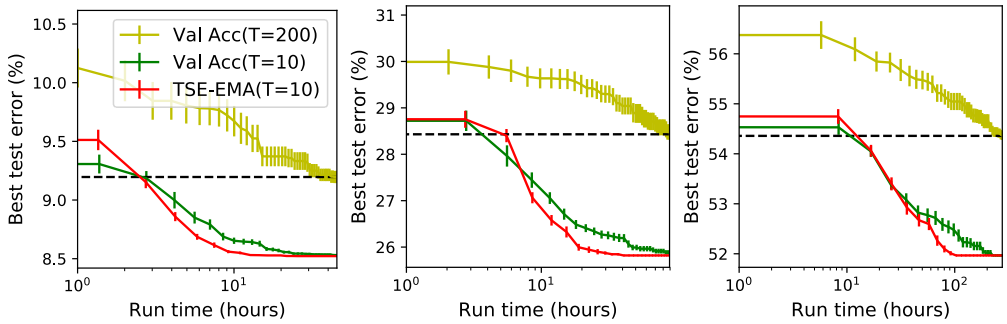
(a) Training speed estimator: TSE-EMA



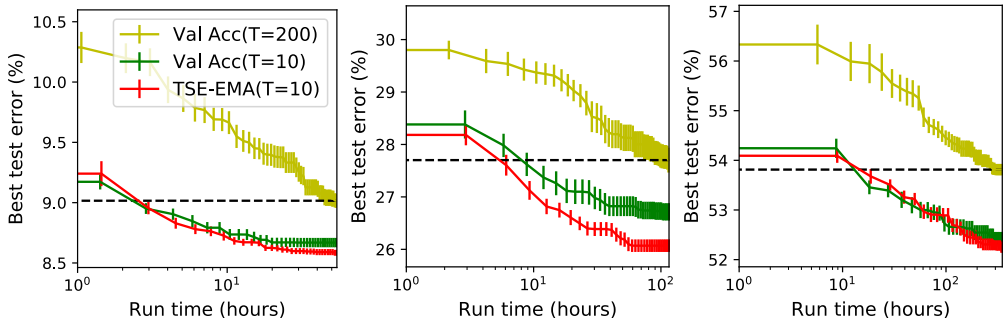
(b) Early-stopped validation accuracy: VAccES

Figure 8.7: Model selection among 69 random graph generator hyper-parameters on RWNN-Flower102 dataset using our TSE-EMA (a) and VAccES (b). We use each hyper-parameter value to generate 8 architectures and evaluate their true test accuracies after complete training. The mean and standard error of the test performance across 8 architectures for each hyper-parameter value are presented as Test Acc (yellow) and treated as ground truth (Right y-axis). We then compute our TSE-EMA estimator for all the architectures by training them for $T < T_{\text{end}} = 250$ epochs. The mean and standard error of TSE-EMA scores for $T = 10, \ldots, 90$ are presented in different colours (Left y-axis of (a)). The rank correlation between the mean Test Acc and that of TSE-EMA for various T is shown in the corresponding legends in (a). The same experiment is conducted by using early-stopped validation accuracy (VAccES) for performance estimation (b). With only 10 epochs of training, our TSE-EMA estimator can already capture the trend of the true test performance of different hyper-parameters relatively well (Rank correlation= 0.851) and can successfully identify 24-th hyperparamter setting as the optimal choice. The prediction of best hyper-parameter by VAccES is less consistent and the rank correlation scores of VAccES at all epochs are lower than those of TSE-EMA.

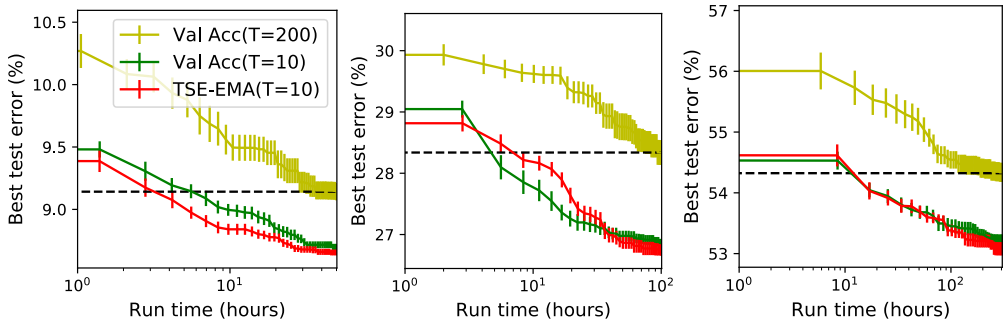
compare this against the other two benchmarks which use the final validation accuracy at $T = T_{\text{end}} = 200$, denoted as Val Acc (T=200) and the early-stop validation accuracy at $T = 10$, denoted as Val Acc (T=10), respectively to evaluate the architecture's generalisation performance. All the NAS search strategies start



(a) Regularised Evolution (RE)



(b) Bayesian Optimisation (BO)



(c) Random Search (RS)

Figure 8.8: NAS performance of Regularised Evolution (RE), Bayesian Optimisation (BO) and Random Search (RS) in combined with final validation accuracy Val Acc (T=200), early-stopping validation accuracy Val Acc (T=10) and our estimator TSE-EMA(T=10) on NB201. For each subplot, we experiment on the three image datasets: on CIFAR10 (left), CIFAR100 (middle) and ImageNet120 (right). TSE-EMA leads to the fastest convergence to the top performing architectures in all cases. The black dashed line is to facilitate the comparison of runtime taken to reach a certain test error among different variants.

their search from 10 random initial data and are repeated for 20 seeds. Note the x-axis is in log scale. The mean and standard error results over the search time are shown in Figure 8.8. By using our estimator, the NAS search strategies can find architectures with lower test error given the same time budget or identify the top performing architectures using much less runtime as compared to using

final or early-stopping validation accuracy. The performance gain of using our estimator or the early-stopped validation accuracy is more significant in the case of RE and BO as compared to the case of RS. For example, given a budget of 50 hours on CIFAR100, RE and BO with our estimator can find an architecture with a test error around or below 26% but RS only finds architecture with test error of around 26.8%. This is due to the fact that RS is purely explorative while RE and BO both trade off exploration and exploitation during their search; our estimator by efficiently estimating the final generalisation performance of the architectures will enable better exploitation. Therefore, we recommend the users to deploy our proposed estimator onto search strategies which involve some degree of exploitation to maximise the potential gain.

8.4.5 Improving One-shot and Gradient-based NAS

Table 8.2: Results of performance estimators in one-shot NAS setting over 3 supernetwork training initialisations. For each supernetwork, we randomly sample 500 random subnetworks for DARTS and 200 for NB201, and compute their TSE, Val Acc, SoVL, Tlmini after inheriting the supernetwork weights and training for B additional minibatches. *Rank correlation* measures the estimators' correlation with the rankings of the true test accuracies of subnetworks when *trained from scratch independently*, and we compute the *average test accuracy of the top 10 architectures* identified by different estimators from all the randomly sampled subnetworks.

B	Estimator	*Rank Correlation*				*Average Accuracy of Top 10 Architectures*			
		NB201-CIFAR10			DARTS	NB201-CIFAR10			DARTS
		RandNAS	FairNAS	MultiPaths	RandNAS	RandNAS	FairNAS	MultiPaths	RandNAS
100	TSE	**0.70 (0.02)**	**0.84 (0.01)**	**0.83 (0.01)**	**0.30(0.04)**	**92.67 (0.12**	**92.70 (0.10)**	92.63 (0.12)	**93.64(0.04)**
	Val Acc	0.44 (0.15)	0.56 (0.17)	0.67 (0.05)	0.11(0.04)	91.47 (0.31)	91.73 (0.21)	91.77 (0.78)	93.20(0.04)
	SoVL	0.54 (0.13)	**0.84 (0.06)**	**0.83 (0.01)**	0.10(0.04)	92.57 (0.15)	92.67 (0.06)	**92.73 (0.06)**	93.21(0.04)
	Tlmini	0.62 (0.03)	0.72 (0.09)	0.74 (0.02)	0.05(0.03)	91.80 (0.40)	92.33 (0.40)	92.43 (0.06)	93.38(0.03)
200	TSE	**0.70 (0.03)**	**0.850 (0.01)**	**0.83 (0.01)**	**0.32(0.04)**	**92.70 (0.00)**	**92.77 (0.06)**	**92.73 (0.06)**	**93.55(0.04)**
	Val Acc	0.41 (0.10)	0.56 (0.17)	0.53 (0.11)	0.09(0.02)	91.53 (0.55)	92.40 (0.10)	92.23 (0.23)	93.34(0.02)
	SoVL	0.52 (0.17)	0.84 (0.06)	0.80 (0.02)	0.08(0.02)	90.70 (1.35)	92.53 (0.15)	92.50 (0.10)	93.36(0.02)
	Tlmini	0.46 (0.14)	0.72 (0.10)	0.69 (0.06)	0.02(0.01)	92.00 (0.35)	92.53 (0.25)	92.40 (0.27)	93.15(0.01)
300	TSE	**0.71 (0.03)**	**0.85 (0.00)**	**0.82 (0.01)**	**0.34(0.04)**	**92.70 (0.00)**	**92.77 (0.06)**	**92.70 (0.00)**	**93.65(0.04)**
	Val Acc	0.44 (0.04)	0.62 (0.08)	0.59 (0.71)	0.06(0.02)	91.20 (0.35)	92.10 (0.50)	91.43 (0.72)	93.31(0.02)
	SoVL	0.45 (0.21)	0.81 (0.05)	0.81 (0.03)	0.05(0.02)	91.00 (1.60)	92.53 (0.15)	92.53 (0.06)	93.26(0.02)
	Tlmini	0.47 (0.12)	0.74 (0.02)	0.70 (0.04)	0.09(0.01)	91.60 (0.44)	92.43 (0.21)	92.27 (0.12)	92.95(0.01)

Different from query-based NAS strategies, which evaluate the architectures queried by training them independently from scratch, another popular class of

NAS methods use weight sharing to accelerate the evaluation of the validation performance of architectures (subnetworks) and use this validation information to select architectures or update architecture parameters. Here we demonstrate that our TSE estimator can also be a plug-in replacement for validation accuracy or loss used in such NAS methods to improve their search performance.

One-shot NAS

We first experiment on a classic one-shot method, RandNAS (Li and Talwalkar, 2020), which trains a supernetwork by uniform sampling, then performs architecture search by randomly sampling subnetworks from the trained supernetwork and comparing them based on their validation accuracy. We follow the RandNAS procedure for the supernetwork training but modify the search phase: for each randomly sampled subnetwork, we train it for B additional mini-batches after inheriting weights from the trained supernetwork to compute our TSE estimator. Note although this introduces some costs, our estimator saves all the costs from evaluation on validation set as it doesn't require validation data. To ensure fair comparison, we also recompute the validation accuracy, SoVL, Tlmini of each subnetwork after the additional training. We also experiment with more advanced supernetwork training techniques such as FairNAS (Chu et al., 2019) and MultiPaths (Yu and Huang, 2019), and show that our estimators can be applied on top of them to further improve the rank correlation performance.

We evaluate the rank correlation performance and average test accuracy of the top-10 architectures recommended by different performance estimators among 500 random subnetworks sampled from the DARTS supernet[4] and 200 random subnetworks from the NB201 supernet. We repeat the experiments on $B = 100, 200, 300$ and over 3 different supernetworks training seeds. The mean and standard deviation results on rank correlation and test accuracy of top architectures are shown in Table 8.2. It is evident that our TSE leads to 170% to 300% increase in rank correlation performance compared to validation accuracy. Moreover, TSE achieves

[4]We use NAS-Bench-301 to compute the true test accuracy of each subnetwork when trained independently from scratch.

higher average test accuracy of the top 10 architectures across all supernetwork training techniques and search spaces. This implies that in comparison with other performance estimators, our estimator can effectively help find architectures with better generalisation performance under the popular weight-sharing setting. Note although our TSE requires additional training of B mini-batches, it does not require the computation of validation statistics. Thus, only a bit or even no additional costs would be induced compared to conventional use of validation accuracy. For example, on DARTS search space, if we follow the RandNAS protocol (batch size=64, train/valid split=0.5), the average cost to compute validation accuracy on entire validation set for one architecture is 6.6 seconds and the average cost of training for $B = 100$ additional mini-batches takes 7.5 seconds. On NB201-CIFAR10, if we follow its default protocol, the validation cost is 6.4 seconds while the additional train cost for $B = 100$ is 4.4 seconds.

Differentiable NAS

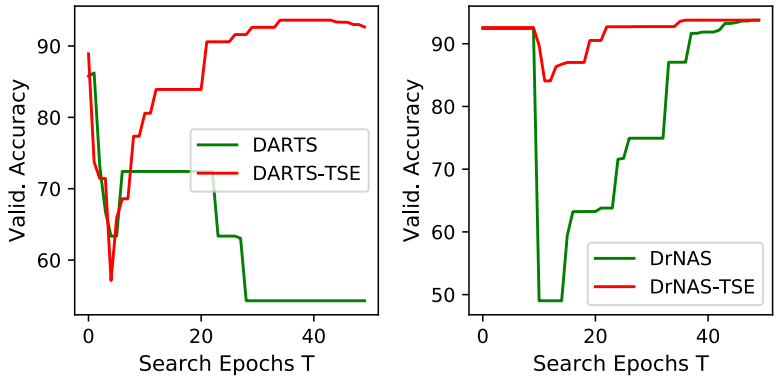
Finally, we demonstrate the use of our estimators on differentiable NAS. We modify two differentiable approach, DARTS (Liu et al., 2019) and DrNAS (Chen et al., 2021c) by directly replacing the derivative of the validation loss with that of our TSE estimator computed over 100 mini-batches (B=100 as in one-shot NAS setting above) to update the architecture parameters. In DARTS (Liu et al., 2019), each intermediate node $\phi^{(j)}$is computed based on all of its predecessors:

$$\phi^{(j)} = \sum_{i<j} \bar{o}^{(i,j)} \left(\phi^{(i)}\right) \tag{8.2}$$

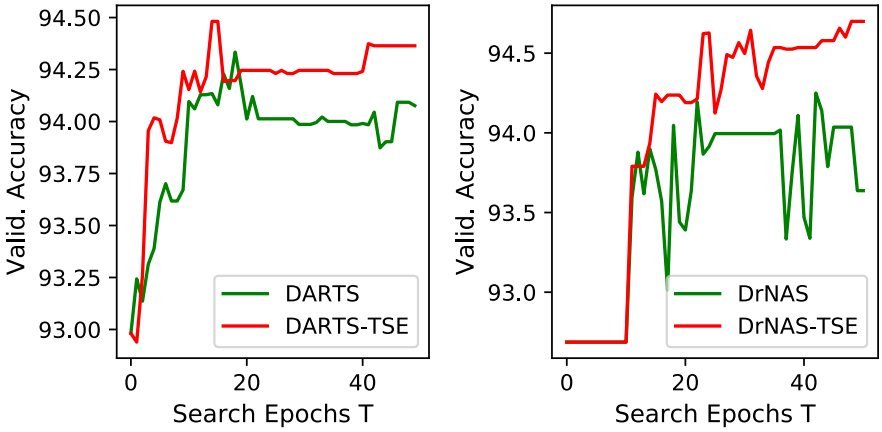
where $\bar{o}^{(i,j)}(\phi)$ is a mix of all possible operations $o(\phi)$:

$$\bar{o}^{(i,j)}\left(\phi\right) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(\phi) \tag{8.3}$$

The architecture parameters to be searched in DARTS is thus a set of continuous vectors $\boldsymbol{\alpha} = \{\alpha^{(i,j)}\}$. Assume we run the search for T epochs and each epoch comprises B mini-batches, the algorithms of DARTS and DARTS-TSE is summarised in Algorithm 13 and 14. Note in DARTS, the architecture parameters are updated



(a) NB201-CIFAR10



(b) DARTS-NB301

Figure 8.9: Test accuracy of the subnetwork recommended by differentiable NAS methods over the search epochs. Original DARTS, DrNAS (in green) use the gradient of validation loss to update the architecture parameters but their variants (DARTS-TSE and DrNAS-TSE) (in red) uses that of our estimator computed over 100 mini-batches. TSE help mitigate the overfitting of DARTS on NB201.

with the derivative of validation loss $\nabla_{\boldsymbol{\alpha}} \ell_{val}(\mathbf{w}, \boldsymbol{\alpha})$ at each mini-batch, leading to a total of BT updates. In DARTS-TSE, we compute the TSE estimator and its derivative using $K = 100$ mini-batches, leading to a less frequent update of $\boldsymbol{\alpha}$. To compensate that, we set $\nabla_{\boldsymbol{\alpha}} \ell_{TSE}(\mathbf{w}, \boldsymbol{\alpha}) = \sum_{k=1}^{K} \nabla_{\mathbf{w}} \ell_{train}^{(k)}(\mathbf{w}, \boldsymbol{\alpha})$ instead of $\nabla_{\boldsymbol{\alpha}} \ell_{TSE}(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{K} \sum_{k=1}^{K} \nabla_{\mathbf{w}} \ell_{train}^{(k)}(\mathbf{w}, \boldsymbol{\alpha})$ for updating $\boldsymbol{\alpha}$.

DrNAS (Chen et al., 2021c) is very similar to DARTS but instead of updating the architecture parameters $\boldsymbol{\alpha}$ directly, DrNAS assumes $\boldsymbol{\alpha}$ is drawn from a Dirichlet distribution $q(\boldsymbol{\alpha}|\boldsymbol{\beta})$ and optimise the distribution parameters $\boldsymbol{\beta}$. $\boldsymbol{\beta}$ is updated

at each mini-batch by descending:

$$\mathbb{E}_{q(\boldsymbol{\alpha}|\boldsymbol{\beta})}\left[\nabla_{\boldsymbol{\beta}}\ell_{val}\left(\mathbf{w},\boldsymbol{\alpha}\right)\right] \tag{8.4}$$

In DrNAS-TSE, we instead use the derivative of TSE to update $\boldsymbol{\beta}$:

$$\sum_{k=1}^{B}\mathbb{E}_{q(\boldsymbol{\alpha}|\boldsymbol{\beta})}\left[\nabla_{\boldsymbol{\beta}}\ell_{train}^{k}\left(\mathbf{w},\boldsymbol{\alpha}\right)\right] \tag{8.5}$$

For both DARTS and DrNAS, we set $K = 100$ for computing our TSE and follow the default setting in Dong and Yang (2020a); Liu et al. (2019); Chen et al. (2021c) for all the other hyper-parameters including B and T.

Algorithm 13 DARTS

Create a mixed operation $\bar{o}^{i,j}$ parametrised by $\alpha^{i,j}$ for each edge (i,j)
for $t = 1,\ldots,BT$ **do**
  Update architecture parameter $\boldsymbol{\alpha}$ by descending $\nabla_{\boldsymbol{\alpha}}\ell_{val}\left(\mathbf{w},\boldsymbol{\alpha}\right)$
  Update weights $\mathbf{w}$ by descending $\nabla_{\mathbf{w}}\ell_{train}\left(\mathbf{w},\boldsymbol{\alpha}\right)$
end for
Derive the final architecture based on the learned $\boldsymbol{\alpha}$

Algorithm 14 DARTS-TSE

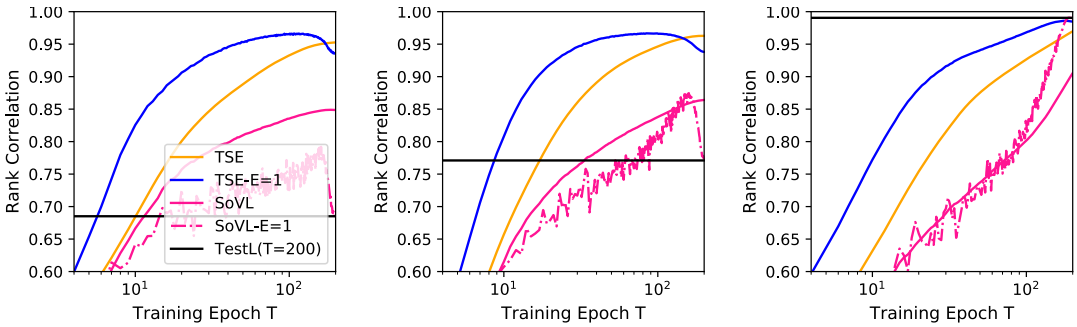
Create a mixed operation $\bar{o}^{i,j}$ parametrised by $\boldsymbol{\alpha}^{i,j}$ for each edge (i,j)
for $t = 1,\ldots,\lfloor BT/K \rfloor$ **do**
  Update architecture parameter $\boldsymbol{\alpha}$ by descending $\nabla_{\boldsymbol{\alpha}}\ell_{TSE}\left(\mathbf{w},\boldsymbol{\alpha}\right)$
  $\nabla_{\boldsymbol{\alpha}}\ell_{TSE}\left(\mathbf{w},\boldsymbol{\alpha}\right) = 0$
  for $k = 1,\ldots,K$ **do**
    Update weights $\mathbf{w}$ by descending $\nabla_{\mathbf{w}}\ell_{train}\left(\mathbf{w},\boldsymbol{\alpha}\right)$
    $\nabla_{\boldsymbol{\alpha}}\ell_{TSE}\left(\mathbf{w},\boldsymbol{\alpha}\right) = \nabla_{\boldsymbol{\alpha}}\ell_{TSE}\left(\mathbf{w},\boldsymbol{\alpha}\right) + \nabla_{\mathbf{w}}\ell_{train}\left(\mathbf{w},\boldsymbol{\alpha}\right)$
  end for
end for
Derive the final architecture based on the learned $\boldsymbol{\alpha}$

We test both DARTS and DrNAS, as well as their TSE variants on the NB201-CIFAR10 as well as DARTS search space. Again we use NAS-Bench-301 to obtain the true test performance of searched DARTS architectures on CIFAR10 (DARTS-NB301). The test accuracy of the subnetwork recommended over the search epochs are shown in Figure 8.9: we average over 3 seeds for NB201-CIFAR10 and use the default seed for DARTS-NB301. The results show that a simple integration of our

estimator into the differentiable NAS frame can lead to clear search performance improvement and even mitigate the overfitting (to skip-connection) problem suffered by DARTS (Zela et al., 2020) on NB201 search space (Figure 8.9(a)).

8.4.6 Ablation Studies

Compute TSE with training losses or validation losses



(a) NB201-CIFAR10 **(b)** NB201-CIFAR100 **(c)** NB201-ImageNet120

Figure 8.10: Rank correlation (with final test *accuracy*) performance of TSE (yellow) and TSE-E (blue), and those of validation losses (pink), SoVL (solid) and SoVL-E (dash dot), as well as that of final test loss (black) for architectures in NB201 on three image datasets. Note the correlation performances of final test loss and SoVL-E near the end of training get surprisingly poor for CIFAR10/100.

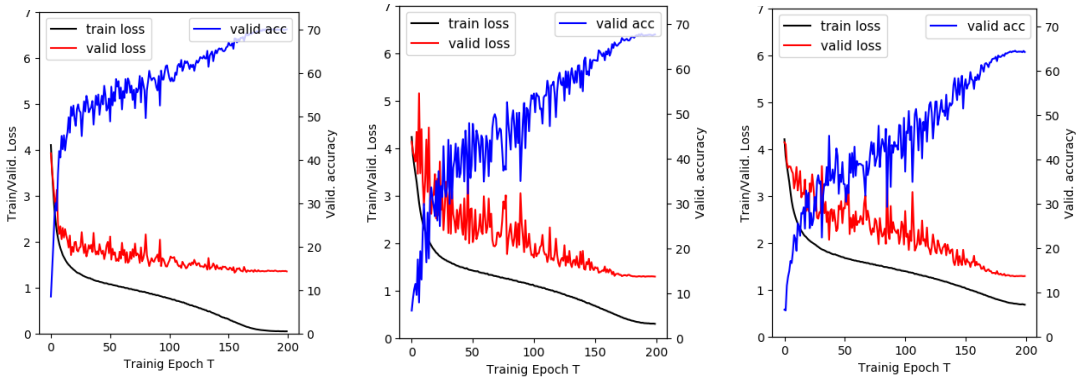
We perform a simple sanity check against the validation loss on NASBench-201 datasets. Specifically, we compare our proposed estimators, TSE and TSE-E, computed with training losses against two equivalent variants of validation loss-based estimators: Sum of validation losses (SoVL) and that over the most recent epoch (SoVL-E with $E = 1$). For each image dataset, we randomly sample 5000 different neural network architectures from the search space and compute the rank correlation between the true test accuracies (at $T = 200$) of these architectures and their corresponding TSE/TSE-E as well as SoVL/SoVL-E up to epoch T. The results in Figure 8.10 show that our proposed estimators TSE and TSE-E using training losses clearly outperform their validation counterparts.

Another intriguing observation is that the rank correlation performance of SoVL-E drops significantly in the later phase of the training (after around 100 epochs for CIFAR10 and 150 epochs for CIFAR100) and the final test loss, TestL (T=200),

also correlates poorly with final test *accuracy*. This implies that the validation/test losses can become unreliable indicator for the validation/test accuracy on certain datasets; as training proceeds, the validation accuracy keeps improving but the validation losses could stagnate at a relatively high level or even start to rise (Mukhoti et al., 2020; Soudry et al., 2018). This is because while the neural network can make more correct classifications on validation points (which depend on the maximum argument of the logits) over the training epochs, it also gets more and more confident on the correctly classified training data and thus the weight norm and maximum of the logits keeps increasing. This can make the network overconfident on the misclassified *validation* data and cause the corresponding validation loss to rise, thus offsetting or even outweighing the gain due to improved prediction performance (Soudry et al., 2018). Training loss will not suffer from this problem. While TSE-E struggles to distinguish architectures once their training losses have converged to approximately zero, this contributes to a much smaller drop in estimation performance of TSE-E compared to that of SoVL-E and only happens near a very late phase of training (after 150 epochs) which will hardly be reached if we want efficient NAS using as *few* training epochs as possible. Therefore, the possibility of network overconfidence under misclassification is another reason for our use of training losses instead of the validation losses.

Example showing training loss is better correlated with validation accuracy than validation loss

We sample three example architectures from the NASBench-201 dataset and plot their losses and validation accuracies on CIFAR100 over the training epochs T. The relative ranking for the validation accuracy is: Arch A (0.70) > Arch B (0.67) > Arch C (0.64), which corresponds perfectly (negatively) with the relatively ranking for the training loss: Arch A (0.05) < Arch B (0.31) < Arch C (0.69). Namely, the best performing architecture also has the lowest final training epoch loss. However, the ranking among their validation losses is poorly/wrongly correlated with that of validation accuracy; the worst-performing architecture has the lowest final validation losses but the best-performing architecture has the highest validation



(a) Arch A:
Train loss=0.05,
Val. loss=1.36,
Val. acc = 0.70

(b) Arch B:
Train loss=0.31,
Val. loss=1.30,
Val. acc = 0.67

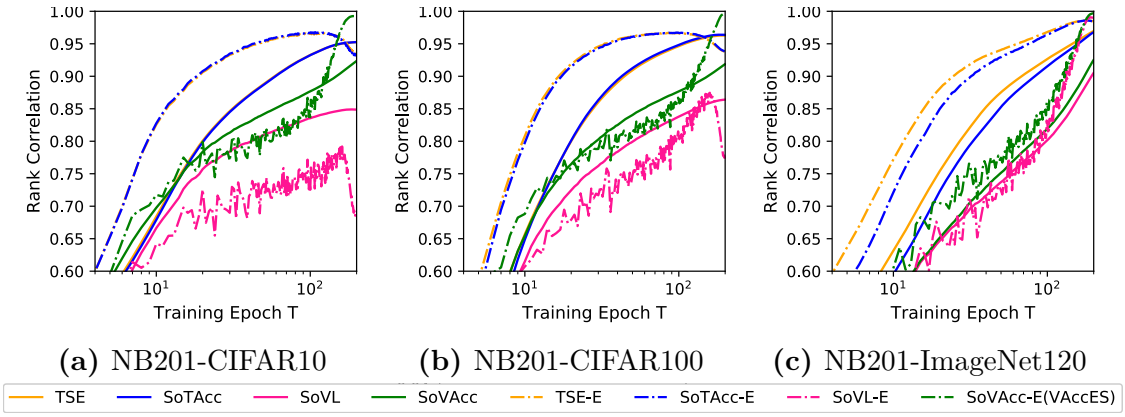
(c) Arch C:
Train loss=0.69,
Val. loss=1.29,
Val acc = 0.64

Figure 8.11: Training losses, validation losses and validation accuracies of three example architectures on CIFAR100. The average of the training losses, validation losses and validation accuracies over the final 10 epochs is presented in the subcaption of each architecture.

losses. Moreover, in all three examples, especially the better-performing ones, the validation loss stagnates at a relatively high value while the validation accuracy continues to rise. The training loss does not have this problem and it decreases while the validation accuracy increases. This confirms the observation we made above that the validation loss will become an unreliable predictor for the final validation accuracy as well as the generalisation performance of the architecture as the training proceeds due to overconfident misclassification.

Comparison with sum over accuracy

We compute another two variants of our estimator, SoTAcc and SoTAcc-E, by summing over training accuracies rather than training losses. Another two baselines to check against are the sum over validation accuracy, SoVAcc and SoVAcc-E. The results on CIFAR10 and CIFAR100 in Figure 8.10 confirm the discussion above: as the training proceeds, the validation loss can become poorly correlated with the validation/test accuracy while the training loss is still perfectly correlated with the training accuracy. It is expected that SoVAcc-E should converge to a perfect rank correlation (=1) with the true test performance at the end of the training.

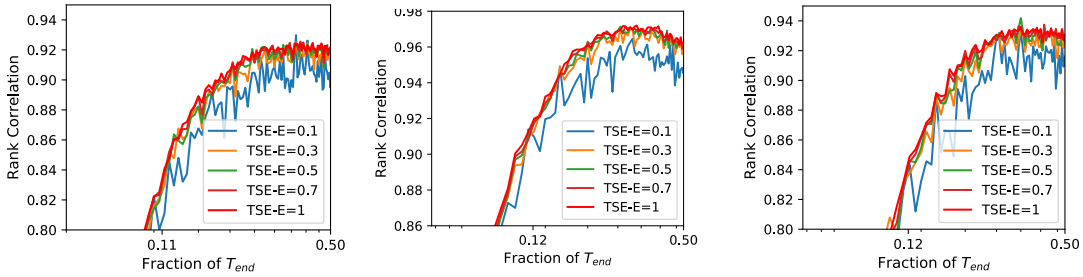


(a) NB201-CIFAR10 **(b)** NB201-CIFAR100 **(c)** NB201-ImageNet120

Figure 8.12: Rank correlation performance of our TSE (yellow), the sum over training accuracy, SoTAcc (blue), the sum over validation losses, SoVL (pink), the sum of validation accuracy, SoVAcc (green) as well as their summing over the recent E epoch counterparts (dash dot) for 5000 random architectures in NASBench-201 on three image datasets.

However, the results in (a), (b) and (c) show that our proposed estimator *TSE-E can consistently outperform SoVAcc-E* in the early and middle phase of the training (roughly $T \leq 150$ epochs). This reconfirms the usefulness of our estimator.

Compute TSE within one epoch



(a) 20 cells, lr=0.025, bs=96, cosine scheduler **(b)** 20 cells, lr=0.1, bs=128, step-decay scheduler **(c)** 20 cells, lr=0.05, bs=128, cosine scheduler

Figure 8.13: Rank correlation performance of the sum of training losses over E most recent epochs (TSE-E) on the 20-cell DARTS dataset. Different E values include those < 1 are investigated for 100 random architectures in DARTS search space under three different evaluation set-ups. In all three settings, $E = 1$ again is the best choice.

We also compute the TSE-E statistics with $E < 1$ (i.e. within an epoch) on the 20-cell DARTS architecture data (used for Figure 8.5), for which we have saved each mini-batch training losses. For example, $E = 0.1$ corresponds to the sum of training losses over the last 10% of the mini-batches/optimisation steps in an epoch.

The results for different E are shown in Figure 8.13. We consistently observe that summing over the entire epoch ($E = 1$) gives the best performance, which might be because $E = 1$ covers the entire training set as the Bayesian marginal likelihood does, which is the theoretical inspiration for our method.

8.5 Conclusion

We propose a simple yet reliable method, TSE, for estimating the generalisation performance of neural architectures based on their training speed measured by the sum of early training losses. Our estimator is theoretically motivated by the connection between generalisation and training speed, as well as the link between sum of training losses and Bayesian marginal likelihood. TSE consistently outperforms many efficient estimators in terms of rank correlation with the true test performance under a variety of search spaces as well as training set-ups. Moreover, it can lead to significant speed-ups and/or performance gains when being directly applied into different types of NAS strategies including even one-shot and differentiable ones. Therefore, we believe TSE can be a valuable tool contributing to more cost-efficient NAS, making NAS and potentially AutoML more affordable by users with limited computation budgets.

Future directions One potential direction to extend our work is to modify the neural network training protocol (e.g. using SGLD (Welling and Teh, 2011) rather than SGD) to simulate a Bayesian update setting so that our proposed TSEs or their variants can be theoretical justified or guaranteed. However, changing the well-optimised neural network training protocol may compromise the network performance and/or lead to significant overheads, thus less attractive from the practical view. The trade-off between the theoretical assurance and practical performance/simplicity needs to be analysed. In addition, our proposed estimators are only empirically verified on comparing architectures for image classification tasks. An exciting future work would be to further evaluate the robustness of our TSE estimators on tasks beyond classification, such as natural language

processing and image segmentation, and/or on other AutoML problems such as hyper-parameter optimisation. One caveat for hyper-parameter optimisation is that hyper-parameters like dropout or learning rate can directly affect the training loss curve. Thus, direct application of our estimator might not work effectively and smart modifications would be needed. Lastly, different performance estimation methods can be complementary and thus potentially combined to achieve better estimation. For example, our TSEs only focus on the network training trajectory while surrogate-based predictors like GPWL or GCN introduced in Chapter 7 only consider the architecture topology and operation types. These two groups of estimators can be integrated to account for both sources of information in predicting the final generalisation performance.

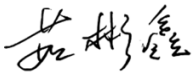
Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Speedy performance estimation for neural architecture search
Publication Status	□Published □ Accepted for Publication ☒Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Binxin Ru*, Clare Lyle*, Lisa Schut, Mark van der Wilk, and Yarin Gal. Speedy performance estimation for neural architecture search. Under review at *Advances in Neural Information Processing Systems (NeurIPS 2021),* 2021.

Student Confirmation

Student Name:	Binxin Ru		
Contribution to the Paper	Lyle and I contributed equally to the discussion of the research idea and the development of the proposed performance estimator. Specifically, I was responsible for all the empirical investigations and result analyses, and contributed to the majority of the write-up. Lyle contributed the theoretical motivations for the proposed estimator.		
Signature 		Date	17/09/2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Prof. Michael A. Osborne		
Supervisor comments To the best of my knowledge, all above seems fair and correct.		
Signature 	Date	17/09/2021

This completed form should be included in the thesis, at the end of the relevant chapter.

9 Conclusions and Future Directions

9.1 Conclusion

In this thesis, we have investigated and addressed a number of research challenges for applying BO methods to AutoML problems like hyper-parameter tuning and neural architecture search (NAS). All our proposed methods can be categorised into two main themes as follows.

1. Improve the efficiency of BO In Chapter 3, we significantly reduced the computation costs and complexity of information-theoretic acquisition functions, leading to a fast alternative, FITBO. This speed-up is achieved via a parabolic transformation of the objective function which allows us to circumvent the laborious process of separately sampling for the global optimum/optimiser. While being much faster to compute, FITBO still maintains the desirable properties and competitive performance of information-theoretic approaches. In Chapter 4, we advocated for the asynchronous batch BO setting to maximise the utility of parallel computing resources. This is particularly relevant to AutoML tasks given that the evaluation runtimes of different configurations often vary significantly. We also demonstrated our penalisation-based approach, PLAyBOOK, is particularly effective for asynchronous execution, clearly outperforming its synchronous counterpart in not

only time efficiency but also query efficiency. Finally in Chapter 8, we studied the important problem of generalisation performance estimation in the specific task of NAS. Reliable performance estimation has always been the computation bottleneck of all search methods, including BO. We developed a cheap yet effective solution, TSE, which is theoretically motivated, mode-free and very simple to deploy. Through extensive experiments on a variety of NAS datasets and search spaces, we showed that our TSE can outperform many existing alternatives including early-stopped validation accuracy, and can be easily plugged into different search methods to achieve remarkable cost saving and/or performance gain. In summary, we improved the efficiency of acquisition function optimisation for information-theoretic BO, the efficiency of resource utilisation for parallel BO and the efficiency of performance estimation for BO-based and in fact, all popular types of NAS strategies.

2. Expand the application scope of BO Besides making BO more efficient, we also explored several new or under-explored problem scenarios for BO, especially GP-based BO, in the thesis. In Chapter 5, we targeted optimisation problems involving *multiple* categorical and *multiple* continuous variables. Such problems is highly common in practical applications, including but not limited to AutoML processes, but receives little attention in BO literatures and thus remains challenging. We introduced a new framework, CoCaBO, which overcomes the drawbacks of the default one-hotting encoding approach by harnessing multi-armed bandits to handle categorical variables. Moreover, CoCaBO makes efficient use of the query data through the design of combined kernel design to capture the coupling across categorical variables. Built on the kernel design in CoCaBO and the concept of local trust regions, we further proposed a new GP-based BO solution, CASMOPOLITAN, which is scalable to the more challenging setting of high-dimensional problems with mixed input types. In Chapter 6, we applied BO on a new application: black-box adversarial attacks on machine learning models. For attacking image classifiers, we overcame the high dimensionality of image input with additive GPs or latent embedding, and developed a unified BO framework, BayesOpt Attack, to

optimise the latent space dimension together with the adversarial perturbations. For attacking graph classifiers, we drew inspiration from our prior work on NAS, and adapted a graph kernel into a sparse Bayesian linear regression to deal with graph structured inputs. The resultant attack method, GRABNEL, is scalable to large graphs and robust under different attacking constraints. Both our BO-based attacks achieve the state-of-the-art query efficiency on a diverse set of tasks, verifying the great potential of BO for efficient assessment of model robustness. Lastly, in Chapter 7, we re-examined the application of BO on NAS. Instead of using encoding schemes or user-defined similarity metrics to compare architecture graphs, we suggested the use of the Weisfeiler-Lehman (WL) graph kernel, which empowers any GP-based BO strategy to naturally handle the graph-like search space. Based on the WL graph kernel, our proposed BO strategy, NAS-BOWL, is applicable to various search spaces and can find the top-performing architectures with much fewer queries than existing baselines. More importantly, during the search, it is also able to learn interpretable sub-features which contribute significantly to the architecture performance, thus making the first attempt towards the interesting direction of interpretable NAS. In summary, we explored new BO solutions for practical problems with mixed input types and high dimensionality, identified a new application domain of black-box adversarial attacks where BO gains clear advantage in query efficiency and found a new kernel approach to enable efficient use of BO on graph input space of NAS with added benefits of interpretability.

We would like to highlight that while we focused our research motivation on the domain of AutoML, the impact of our research contributions goes beyond AutoML because problem settings discussed in this thesis are also important and commonly encountered in many other domains. For example, the graph-structured search space, appeared in NAS and graph classification attacks, is highly relevant to the molecule generation/optimisation in chemistry and drug discovery (Chen et al., 2021a; Grosnit et al., 2021). Tasks involving a mixture of multiple categorical and continuous inputs, tackled by CoCaBO and CASMOPOLITAN, are also frequently faced in material design (Zhang et al., 2020). Therefore, we

hope that the family of BO approaches developed by us can lead to promising solutions for challenging black-box optimisation problems in many other industrial or scientific applications than machine learning.

9.2 Future Directions

We have discussed the potential extensions of each of our methods at the end of their corresponding chapters. Here we discuss several general research directions that we would like to explore in the future

Distributional NAS

Architectures, which have operation types or topology structures generated from the same generative distribution, tend to achieve similar test performance. However, conventional NAS framework treats such architectures as different individuals and requires evaluating them independently. In fact, lots of resources are often wasted on evaluating very similar architectures, leading to inefficient exploration of the search space (Yang et al., 2020; Ru et al., 2020b). A more efficient framework would be *distributional NAS* where the objective is to search for the optimal (generative) distribution of architectures rather than a single architecture. For example, instead of deterministically defining all the wiring within an architecture, we can employ random graph generators such as Watts-Strogatz model (Watts and Strogatz, 1998) to create wiring stochastically (Xie et al., 2019a; Ru et al., 2020b). We can then search for the best distribution over the wiring pattern (i.e. architecture topology) by optimising the hyper-parameters of the random graph generator. Such reformulation essentially converts NAS to back to hyper-parameter optimisation and introduces several benefits. First, the hyper-parameters of the generative distribution are likely to be continuous/discrete, and incur much lower dimensions than the original deterministic search space (Ru et al., 2020b) [1]. Such hyper-parameter space is essentially the same as a latent embedding space where multiple architectures can

[1] Take the example of defining the wiring pattern in a 32-node graph, if we use Watts-Strogatz model,

be projected to the same embedding, and is more amenable to BO compared to the original graph-like space. Second, architectures sampled from the same generative distribution tend to have close test performance (tight performance distribution), especially for generative distributions over good architectures (Ru et al., 2020b). This implies that we may only need to evaluate one architecture sample and use its test performance as a noisy proxy for the entire distribution. As a result, we can spend computation resources and time on evaluating and comparing architectures that are truly different (from different generative distributions), significantly enhancing the exploration efficiency over large search space. Third, searching good architecture distributions can be potentially useful for other NAS variants. For example, in architecture ensemble search, current methods (Zaidi et al., 2020) need to first build a large enough candidate pool of good base learners by searching and fully evaluating hundreds of architectures. Distributional NAS can offer an efficient way to build such candidate pool because good generative distributions are inherently good candidate pool from which we can sample well-performing base learner architectures. Lastly, distributional NAS might better capture global properties important for the network efficacy without distraction by local variations (You et al., 2020b), offering valuable insights on network design and performance estimation.

Joint Optimisation of Hyper-parameters and Architectures

Different AutoML processes such as hyper-parameter optimisation and NAS are often conducted in sequential stages. However, such separate execution requires careful coordination by human experts and often lead to sub-optimal results (Dong et al., 2020; Zela et al., 2018; Dai et al., 2020; Zhou et al., 2021) compared to joint optimisation. Joint optimisation of hyper-parameters and architectures, though being more desirable, remains challenging due to the largely increased search dimension, mixed input types and complex interaction between different inputs. Several attempts have been made: Dong et al. (2020); Zhou et al. (2021) accommodate the high-dimensionality of the augmented search space by adapting the gradient-based NAS methods to update both hyper-parameters and architectures

under the weight-sharing assumption. Dai et al. (2020) trains a predictor to estimate the network performance based on both the architecture and hyper-parameters, and then uses evolutionary algorithm to perform the search. These approaches still have much room for improvement, while elegant and competitive BO solutions are also missing with only a brief attempt by Zela et al. (2018) on a simplified setting. Some of our work discussed in the thesis may help address part of the challenge; we can handle the high-dimensional mixed inputs of the augmented search space by using the graph kernel on architecture subspace (Chapter 6 and 7) and using categorical and continuous kernels (Chapter 5) on corresponding types of hyper-parameter inputs. A combined kernel can then be designed as in Chapter 5 to effectively model the joint search space. Moreover, efficient performance estimation would be challenging but crucial for the success of the joint search as we need to extrapolate the performance of an architecture under different training hyper-parameters. For example, if we want to compare two configurations: A=$\{\text{arch A}, \text{lr} = 0.1\}$ and B=$\{\text{arch B}, \text{lr} = 0.01\}$. While the configuration with lr=0.01 is better that with lr=0.01 when fully trained (e.g. for 200 epochs), the configuration with lr=0.1 would likely lead to higher validation accuracy if we early-stop the training at 5 epochs. Thus, we need estimators that can reliably predict the generalisation performance based on early training curves. Our TSE estimator in Chapter 8 could be potentially modified or combined with other predictors for this purpose.

Interpretability in AutoML

Following our very preliminary attempt to interpretable NAS in Chapter 7, we believe there are a lot more to explore in the exciting direction of making AutoML results more interpretable. Currently, most AutoML processes act like black-boxes to the end users as the algorithms only return the final recommendation without any explanation on the "thinking processes". If AutoML algorithms could offer some human-interpretable insights on what constitutes to a good architecture or how the final decision is made, the human users would be more willing to accept and trust solutions suggested via AutoML, especially in safety-critical applications. This

can encourage the adoption of AutoML in more real-world industrial applications. Moreover, interpretable patterns or insights learnt during AutoML processes can help improve our understanding on the objective task. Take the example of NAS. Most, if not all, current NAS search spaces are well-engineered based on our prior knowledge on good design practices, where, for example, residual connections and structured operation units are manually enforced. Thus, the architectures found by NAS nowadays are mainly close variants to the state-of-the-art manually designed networks such as ResNet (He et al., 2016), DenseNet (Huang et al., 2017) and Transformer (Vaswani et al., 2017). However, the more ambitious goal of NAS should be to discover truly novel architectures in a highly general and expressive search space. And with added interpretability, NAS could potentially discover new design paradigms during the search for such novel architectures so as to enhance our understanding on architecture design and guide future human design. Therefore, at a higher level, interpretablAutoML methods, while being able to find the optimal solution, may also offer the key to open up the black-box objective problem to provide valuable general knowledge on the objective problem for the end users.

Automated Data Collection and Processing

Compared to model selection and hyper-parameter tuning, data collection and processing receive much less attention in the AutoML research community but are, nonetheless, equally important. In a common set-up, we assume the non-machine-learning experts provide task datasets and objectives to the AutoML system and the system returns the most suitable machine learning model for the task with hyper-parameters tuned to maximise the objectives. However, randomly collected datasets might be sub-optimal for model training or evaluation on the objective task, and the amount of data available for a new task or for a task with expensive data collection/labelling might be insufficient. Thus, it would be more desirable if the AutoML system can inform the user how much additional data and which data points are required by the machine learning model. Active learning is an ideal tool for this purpose and many algorithms (Houlsby et al., 2011; Kirsch et al., 2019; Gal

et al., 2017; Kossen et al., 2021) have been proposed to iteratively collect or label data points that maximise the information gain for the target machine learning models. Given that active learning and BO, though serve different purposes, are closely related, we could potentially adapt advanced techniques from BO to active learning to further improve its data selection efficiency. One successful example is BatchBALD (Kirsch et al., 2019) which leverage the batch selection technique in BO to parallelise the sequential BALD algorithm (Houlsby et al., 2011; Gal et al., 2017). A more interesting direction would be to integrate active learning into a BO framework (e.g. FABOLAS (Klein et al., 2016a) [2]) to benefit from the synergy among data collection, model selection and/or hyper-parameter tuning.

On a concluding remark, we hope that the techniques presented in this thesis could contribute to the future of AutoML with BO playing a critical role.

[2] FABOLAS optimises the dataset size along with hyper-parameters in a contextual BO framework but selects the training data randomly.

Appendices

A

Additional Experiments

A.1 Compare Methods for Approximation a Gaussian Mixture Entropy

The following experiments are conducted to validate as well as compare the three entropy approximation methods used in FITBO in Chapter 3:

1. Huber's method that uses Taylor series expansion (Huber),
2. Numerical integration that uses adaptive Simpson quadrature (Quadra) and
3. Simple Monte Carlo integration (MC).

The approximation performance is assessed in terms of accuracy and computational speed. The optimal approximation method is then chosen based on the trade-off between the accuracy and computational demand.

The methodology of the tests can be summarised as follows:

1. Generate a Gaussian mixture as a weighted sum of N 1-D random Gaussian distributions
2. For the Gaussian mixture, estimate its true entropy by using simple Monte Carlo method with large sample size (e.g. MC50000)

3. Use the 3 approximation methods to approximate the entropy of the Gaussian mixture. For the MC method, try it with different sample sizes (e.g. MC10,MC100,MC1000)

4. Compute and record the running time as well as absolute and fractional approximation errors for each method.

5. Repeat the above processes for K different gaussian mixtures and compute the median running time and the median of the approximation errors.

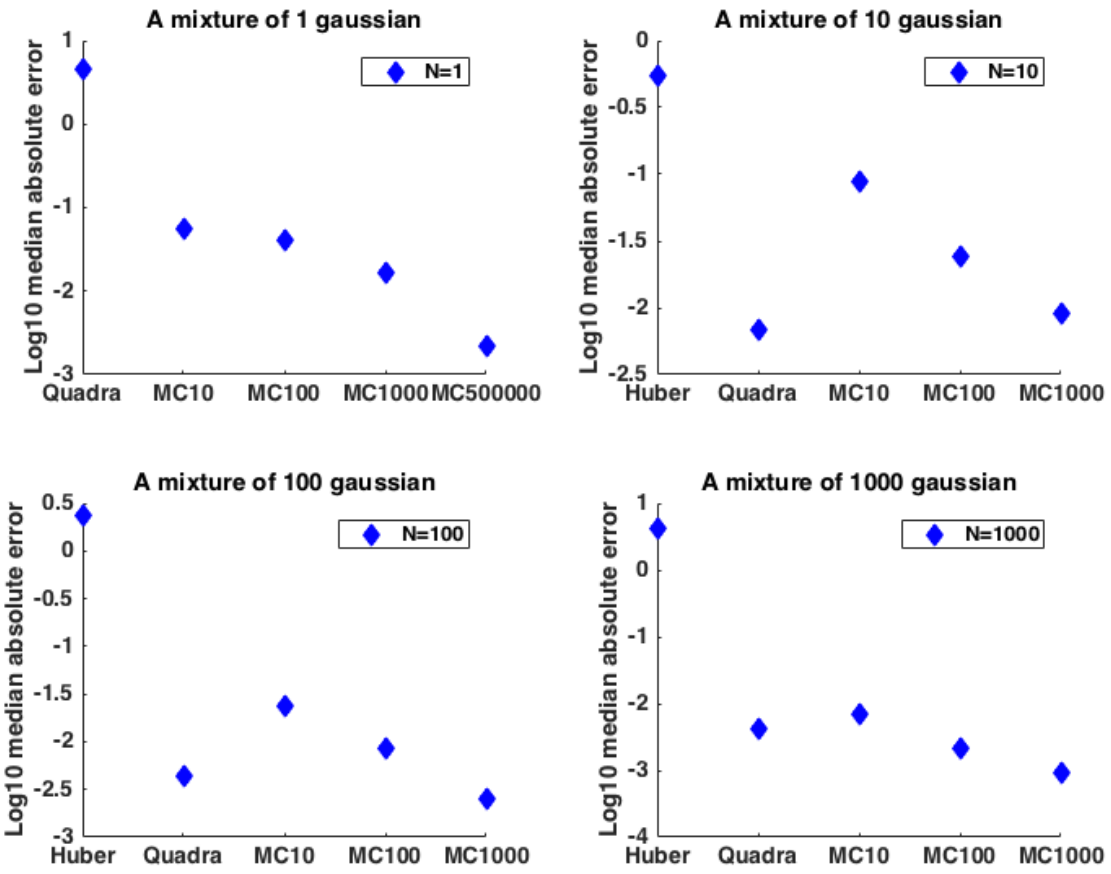


Figure A.1: Log median absolute error in approximating the entropy of a Gaussian mixture.

With reference to Figure A.1 and A.2, in the case of a single Gaussian($N = 1$) distribution, there is a closed-form expression for its entropy. The Huber's method gives the exact true entropy solution, thus having 0 approximation error. The other 2 approximation methods (Quadra and MC) are compared against the true

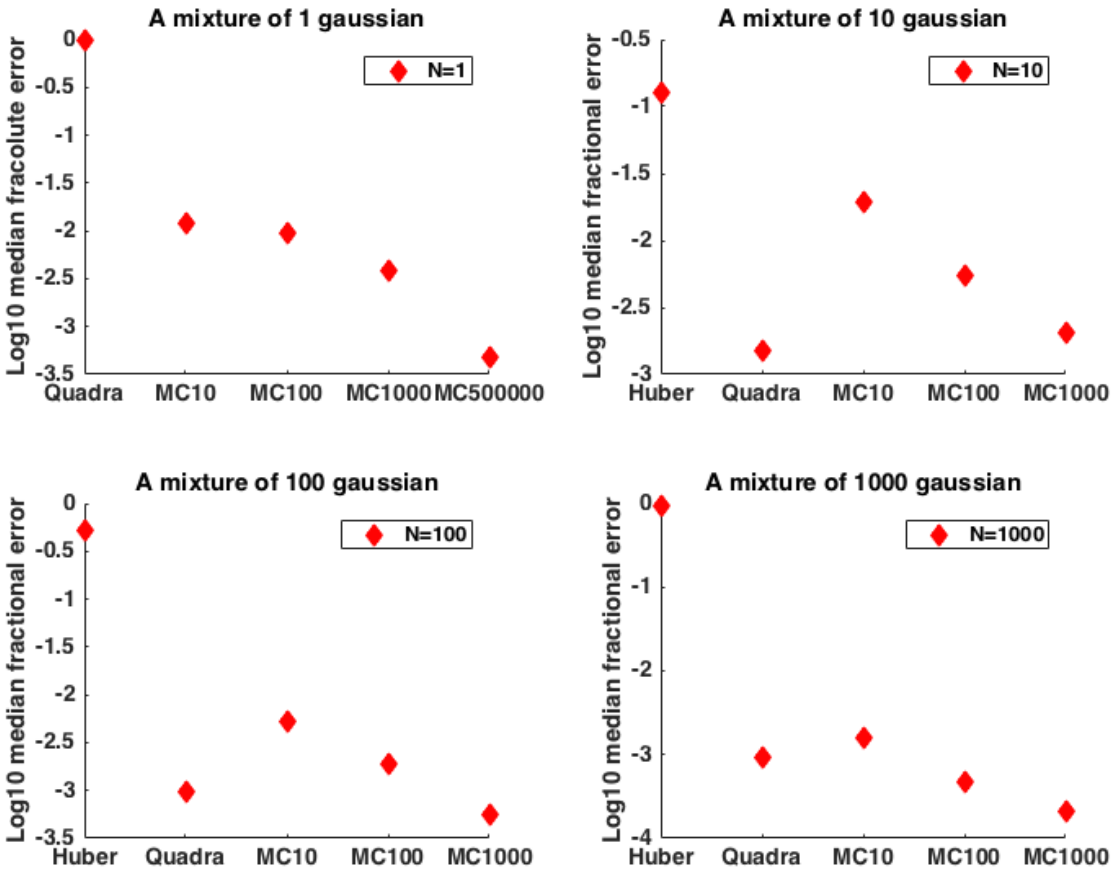


Figure A.2: Log median fractional error in approximating the entropy of a gaussian mixture.

entropy value. It is evident that the approximation by Monte Carlo with 50000 samples (MC50000) is very close to the true value, which justifies our use of the approximation results of MC50000 as our yardstick for the cases of more than one Gaussians in the mixture.

For a mixture of more than one Gaussian distribution ($N > 1$), the performances of all 3 approximation methods (Huber, Quadra, MC) are compared against the entropy value estimated by MC50000. The results in Figure A.1 and A.2 show that Monte Carlo with a sample size of 1000 (MC1000) produces the most accurate approximation in terms of absolute and fractional approximation errors. MC100 and quadrature (Quadra) also have relatively accurate approximation with low absolute and fractional error. The Huber method leads to the highest approximation errors. This may be due to the low order (order of 2) of Taylor-series expansion used in our experiments.

In Figure A.3, the running times of all 3 approximation methods are compared. As expected, the results show that the computation time increases as the number of Gaussians in the mixture rises. This is mainly due to the computation burden associated with the construction of the Gaussian mixture. More importantly, the quadrature method (Quadra) gains speed advantage as the number of gaussians in the mixture increases because the computational cost of approximation using quadrature does not increase with the number of Gaussian components in the mixture. The speed advantage of the quadrature method becomes more salient as we have more Gaussians in the mixture which is reflected in the growing difference among the running times of these methods. Since the number of gaussians in the mixture (N) is determined by the number of hyperparameter samples we use for marginalisation in our algorithm, if we want to use a larger number of hyperparameter sets, we should adopt the quadrature method for fast approximation of the Gaussian mixture entropy at decent accuracy.

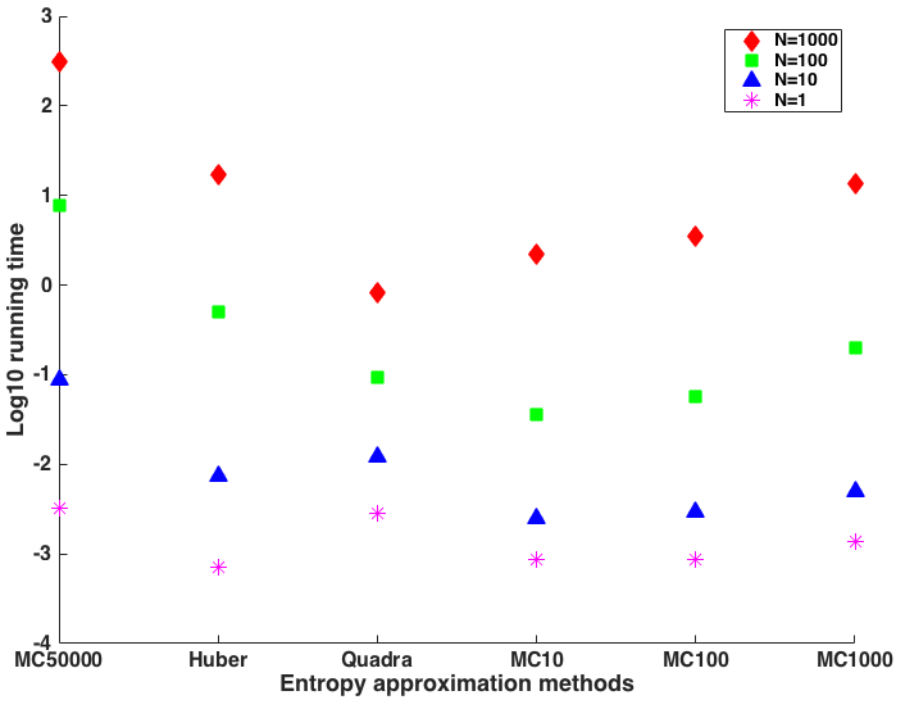


Figure A.3: Compare the running times of the approximation methods.

A.2 Compute a Gaussian Mixture Entropy in FITBO using Maximum Entropy Algorithm

As discussed in Section 3.2.3, the entropy of a Gaussian mixture does not have a closed-form solution and needs to be approximated. Here we develop an effective approach to estimate the Gaussian mixture entropy using Maximum Entropy Method. Thus, we denoted this approach as MEMe.

A.2.1 Moments of a Gaussian Mixture

For a one-dimensional Gaussian $\mathcal{N}(y; \mu, \sigma^2)$, we can compute its m-th moments analytically:

$$\int_{-\infty}^{\infty} y^m \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) dy = \sum_{i=0}^{m} (\sqrt{2}\sigma)^{i+1} \binom{m}{i} \mu^{m-i} \zeta_i, \tag{A.1}$$

where

$$\zeta_i = \int_{-\infty}^{\infty} z^i \mathrm{e}^{-z^2} dz = \begin{cases} \frac{(i-1)!!\sqrt{\pi}}{2^{i/2}}, & i \text{ even}, \\ [1/2(i-1)]!, & i \text{ odd}. \end{cases} \tag{A.2}$$

with $(i-1)!!$ denotes double factorial. Hence, for a mixture of M one-dimensional Gaussians $p(y) = \frac{1}{M} \sum_j^M \mathcal{N}(y; \mu_j, \sigma_j^2)$, the m-th moment is also analytic:

$$\sum_{j}^{M} w_j \sum_{i=0}^{m} (\sqrt{2}\sigma_j)^{i+1} \binom{m}{i} \mu_j^{m-i} \zeta_i. \tag{A.3}$$

where w_j is the weight of j-th Gaussian in the mixture. Since FITBO deals with the one-dimensional output space, the Gaussian mixture involved in the acquisition function is also a mixture of one-dimensional Gaussians, for which we can compute the moments analytically.

A.2.2 Maximum Entropy Distribution

The method of maximum entropy (Pressé et al., 2013) is a procedure for generating the most conservative estimate of a probability distribution with the given information and the most non-committal one with regard to missing information (Jaynes, 1957). Intuitively, in a bounded domain, the most conservative distribution, i.e., the distribution of maximum entropy, is the one that assigns equal probability

to all the accessible states. Hence, the method of maximum entropy can be thought of as choosing the flattest, or most equiprobable distribution, satisfying the given moment constraints.

To determine the maximally entropic density

$$q(y) = \exp(-[1 + \sum_{i=0}^{m} \alpha_i y^i]), \tag{A.4}$$

we need to maximise the entropic functional

$$S = -\int q(y) \log q(y) dy - \sum_{i=0}^{m} \alpha_i \left[\int q(y) y^i dy - \gamma_i \right], \tag{A.5}$$

with respect to $q(y)$, where the second term with $\gamma_i = \mathbb{E}_p[y^i]$ for some density $p(y)$ are the power moment constraints on the density $q(y)$, $\{\alpha_i\}$ are the corresponding Lagrange multipliers, and m is the number of moments.

We develop an algorithm for determining the maximum relative entropy density given moment constraints, which instead of maximising the generic objective of Equation (A.5), minimises its dual form (Boyd and Vandenberghe, 2009):

$$\mathcal{S}(q, q_0) = \int q_0(y) \exp(-[1 + \sum_{i=0}^{m} \alpha_i y^i]) dy + \sum_{i=0}^{m} \alpha_i \mu_i. \tag{A.6}$$

This is because this dual objective admits analytic expressions for both the gradient and the Hessian:

$$\frac{\partial \mathcal{S}(q, q_0)}{\partial \alpha_j} = \mu_j - \int q_0(y) y^j \exp(-[1 + \sum_{i=0}^{m} \alpha_i y^i]) dy, \tag{A.7}$$

$$\frac{\partial^2 \mathcal{S}(q, q_0)}{\partial \alpha_j \partial \alpha_m} = \int q_0(y) y^{j+m} \exp(-[1 + \sum_{i=0}^{m} \alpha_i y^i]) dy. \tag{A.8}$$

For a flat prior in a bounded domain for Gaussian mixture models, $q_0(x)$ is a constant that can be dropped out. The overall maximum entropy algorithm is shown in Algorithm 15.

Given that any polynomial sum can be written as $\sum_i \alpha_i y^i = \sum_i \beta_i f_i(y)$, where $f_i(y)$ denotes another polynomial basis, whether we choose to use power moments in our constraints or another polynomial basis, such as the Legendre basis, does not change the entropy or solution of our objective. However, the performance of

Algorithm 15 The Proposed Maximum Entropy Algorithm

1: **Input:** Moments $\{\mu_i\}$, Tolerance Level ϵ, Jitter variance in Hessian $\eta = 10^{-8}$
2: **Output:** Coefficients $\{\alpha_i\}$
3: Initialise $\alpha_i = 0$
4: Compute gradient: $g_j = \mu_j - \int_0^1 q_\alpha(y) y^j dy$
5: Compute Hessian: $H_{j,k} = \int_0^1 q_\alpha(y) y^{j+k} dy$, $H = \frac{1}{2}(H + H^T) + \eta I$
6: Minimize $\int_0^1 q_\alpha(y) dy + \sum_i \alpha_i \gamma_i$ using Conjugate Gradients until $\forall j$: $g_j < \epsilon$

optimisers working on these different formulations may vary (Skilling, 1989). For simplicity, we have kept all the formulas in terms of power moments. However, we find vastly improved performance and conditioning when we switch to Legendre basis as shown in Section A.2.4.

A.2.3 Approximate Gaussian Mixture Entropy with MEMe

MEMe essentially computes an analytic upper analytic upper bound of the Gaussian mixture entropy, which is derived from the non-negative relative entropy between the true Gaussian mixture density $p(y)$ and the maximum entropy distribution $q(y)$ (Granziol and Roberts, 2017):

$$\mathcal{D}_{\mathrm{KL}}(p||q) = -H(p) + H(q) \geq 0, \tag{A.9}$$

hence

$$H(p) \leq H(q). \tag{A.10}$$

Notice that $q(y)$ shares the same moment constraints $\{\gamma_i\}$ as $p(y)$ which can be computed analytically using Equation (A.3). The entropy of the maximum entropy distribution $q(y)$ can also be derived analytically:

$$H(q) = 1 + \sum_{i=0}^{m} \alpha_i \gamma_i, \tag{A.11}$$

where $\{\gamma_i\}$ are the moments of a Gaussian mixture and $\{\alpha_i\}$ are the Lagrange multipliers that can be obtained by applying Algorithm 15. The overall algorithm for approximating the Gaussian mixture entropy is then summarised in Algorithm 16.

Algorithm 16 MEMe for Approximating Gaussian Mixture Entropy

1: **Input:** A univariate Gaussian mixture $p_{\mathrm{GM}}(y) = \frac{1}{M}\sum_j^M \mathcal{N}(y; m_j, \sigma_j^2)$
2: **Output:** Gaussian mixture entropy $H_{\mathrm{GM}} \approx H\left[\frac{1}{M}\sum_j^M \mathcal{N}(y; m_j, \sigma_j^2)\right]$
3: Compute the moments of $p_{\mathrm{GM}}(y)$, $\{\gamma_i\}$, analytically using Equation (A.3)
4: Compute the Lagrange multipliers, $\{\alpha_i\}$, using Algorithm 15
5: $H_{\mathrm{GM}} \approx -(1 + \sum_i \alpha_i \mathbb{E}[y^i]) = -(1 + \sum_i \alpha_i \gamma_i)$

A.2.4 Experiments

Estimate the Entropy of a Gaussian Mixture We first test a diverse set of methods to approximate the entropy of two sets of Gaussian mixtures, which are generated using FITBO with 200 hyperparameter samples and 400 hyperparameter samples on a 2D problem. Specifically, we compare the following approaches for entropy approximation: MaxEnt methods using 10 and 30 power moments (MEMe-10 and MEMe-30), MaxEnt methods using 10 and 30 Legendre moments (MEMeL-10 and MEMeL-30), method proposed in (Huber et al., 2008) with 2 Taylor expansion terms (Huber-2), Monte Carlo with 100 samples (MC-100), and simple moment matching (MM).

We evaluate the performance in terms of the fractional error between the approximated entropy value and the true entropy value, the latter of which is estimated via expensive numerical integration. The results of the mean approximation errors by all methods over 80 different Gaussian mixtures are shown in Table A.1. We can see clearly that all MEMe approaches outperform other methods in terms of the approximation error. Among the MEMe approaches, the use of Legendre moments, which apply orthogonal Legendre polynomial bases, outperforms the use of simple power moments.

The mean runtime taken by all approximation methods over 80 different Gaussian mixtures are shown in Table A.2. To ensure a fair comparison, all the methods are implemented in MATLAB and all the timing tests are performed on a 2.3GHz Intel Core i5 processor. As we expect, the moment matching technique, which enables us to obtain an analytic approximation for the Gaussian mixture entropy, is the fastest method. MEMe approaches are significantly faster than the rest of approximation

Table A.1: Mean fractional error in approximating the entropy of the mixture of M Gaussians using various methods.

Methods	**M=200**	**M=400**
MEMe-10	1.24×10^{-2}	1.38×10^{-2}
MEMe-30	1.13×10^{-2}	1.06×10^{-2}
MEMeL-10	1.01×10^{-2}	0.85×10^{-2}
MEMeL-30	$\mathbf{0.50 \times 10^{-2}}$	$\mathbf{0.36 \times 10^{-2}}$
Huber-2	24.7×10^{-2}	35.5×10^{-2}
MC-100	1.60×10^{-2}	2.72×10^{-2}
MM	2.78×10^{-2}	3.22×10^{-2}

methods. This demonstrates that MEMe approaches are highly efficient in terms of both approximation accuracy and computational speed. Among all the MEMe approaches, we choose to apply MaxEnt with 10 Legendre moments in the BO for the next set of experiments, as it is able to achieve lower approximation error than MaxEnt with higher power moments while preserving the computational benefit of FITBO.

Table A.2: Mean runtime of approximating the entropy of the mixture of M Gaussians using various methods.

Methods	**M=200**	**M=400**
MEMe-10	1.38×10^{-2}	1.48×10^{-2}
MEMe-30	2.59×10^{-2}	3.21×10^{-2}
MEMeL-10	1.70×10^{-2}	1.75×10^{-2}
MEMeL-30	4.18×10^{-2}	4.66×10^{-2}
Huber-2	20.9×10^{-2}	82.2×10^{-2}
MC-100	10.6×10^{-2}	40.1×10^{-2}
MM	$\mathbf{2.71 \times 10^{-5}}$	$\mathbf{2.87 \times 10^{-5}}$

Apply to FITBO Framework We now test the effectiveness of MEMe when applied to FITBO. We first illustrate the entropy approximation performance of

MEMe using a 1D example. In Figure A.4, the top plot shows the objective function we want to optimise (red dash line) and the posterior distribution of our surrogate model (blue solid line and shaded area). The bottom plot shows the acquisition functions computed based on Equation (C.1) using the same surrogate model but three different methods for Gaussian mixture entropy approximation, i.e., expensive numerical quadrature or Quad (red solid line), MM (black dash line), and MEMe using 10 Legendre moments (green dash line). In BO, the next query location is obtained by maximising the acquisition function, therefore the location instead of the magnitude of the modes of the acquisition function matters most. We can see from the bottom plot that MEMeL-10 results in an approximation that well matches the true acquisition function obtained by Quad. As a result, MEMeL-10 manages to recommend the same next query location as Quad. In comparison, the loose upper bound of the MM method, though successfully capturing the locations of the peak values, fails to correctly predict the global maximiser of the true acquisition function. MM therefore recommends a query location that is different from the optimal choice. As previously mentioned, the acquisition function in information-theoretic BO represents the information gain about the global optimum by querying at a new location. The sub-optimal choice of the next query location thus imposes a penalty on the optimisation performance as seen in Figure A.5.

In the next set of experiments, we evaluate the optimisation performance of three versions of FITBO that use different approximation methods. Specifically, FITBO-Quad denotes the version that uses expensive numerical quadrature to approximate the entropy of the Gaussian mixture, FITBO-MM denotes the one using simple moment matching, and FITBO-MEMeL denotes the one using MEMe with 10 Legendre moments. We test these BO algorithms on two challenging optimisation problems, i.e., the Michalewicz-5D functio and the Hartmann-6D function, and measure the optimisation performance in terms of the immediate regret as defined in Section 3.3.3. The average (median) result over 10 random initialisations for each experiment is shown in Figure A.5. It is evident that the MEMe approach (FITBO-MEMeL-10), which better approximates the Gaussian

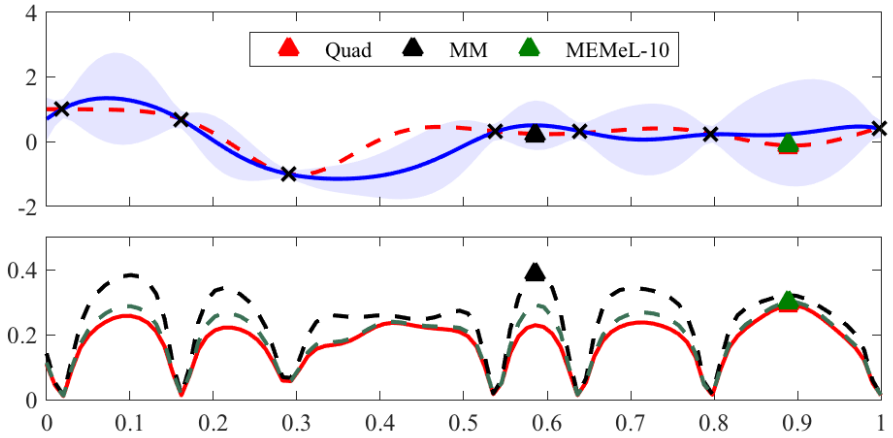
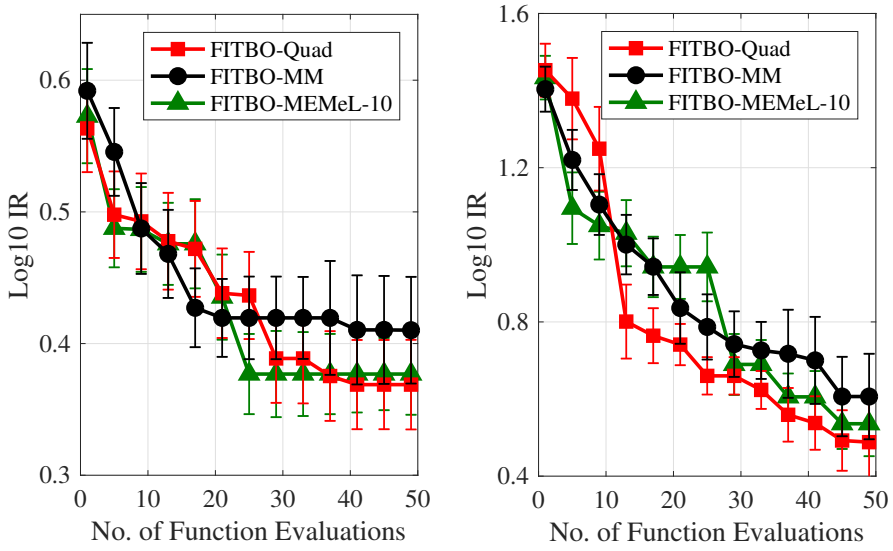


Figure A.4: Bayesian Optimisation (BO) on a 1D toy example with acquisition functions computed by different approximation methods. In the top subplot, the red dash line is the unknown objective function, the black crosses are the observed data points, and the blue solid line and shaded area are the posterior mean and variance, respectively, of the GP surrogate that we use to model the latent objective function. The coloured triangles are the next query point recommended by the BO algorithms, which correspond to the maximiser of the acquisition functions in the bottom subplot. In the bottom plot, the red solid line, black dash line, and green dotted line are the acquisition functions computed by Quad, MM, and MEMe using 10 Legendre moments, respectively.



(a) Michalewicz-5D **(b)** Hartmann-6D

Figure A.5: Performance of various versions of FITBO on 2 benchmark test problems: (**a**) Michalewicz-5D function and (**b**) Hartmann-6D function. The immediate regret (IR) on the y-axis is expressed in the logarithm to the base 10.

mixture entropy, leads to a superior performance of the BO algorithm compared to the BO algorithm using simple moment matching technique (FITBO-MM).

A.3 Additional Experiments for PLAyBOOK

As mentioned in Chapter 4, we conducted empirical evaluations on a large number of synthetic test problems to compare synchronous and asynchronous parallel BO methods:

- The tasks mat-2 and mat-6 refer to functions drawn from a Gaussian process(GP) with Matérn-52 kernel in $\mathbb{R}^2$ and $\mathbb{R}^6$ respectively.
- The global optimisation tasks[1] that we considered are the Ackley function defined on $\mathbb{R}^5$ and $\mathbb{R}^{10}$ (ack-5 and ack-10), the Michalewicz function defined on $\mathbb{R}^5$ and $\mathbb{R}^{10}$ (mic-5 and mic-10) and the Eggholder function in $\mathbb{R}^2$ (egg-2).
- We also selected a robot pushing simulation experiment, which was first explored in a BO context by Wang and Jegelka (2017b). Here the task is to learn the correct pushing action to minimise the distance of the robot to a goal. The problem has 4 inputs: the robot's location (r_x, r_y), the angle of the pushing force r_θ and the pushing duration t_r. We used the input space suggested by Wang and Jegelka (2017b).

A.3.1 Asynchronous vs. Synchronous Parallel BOs

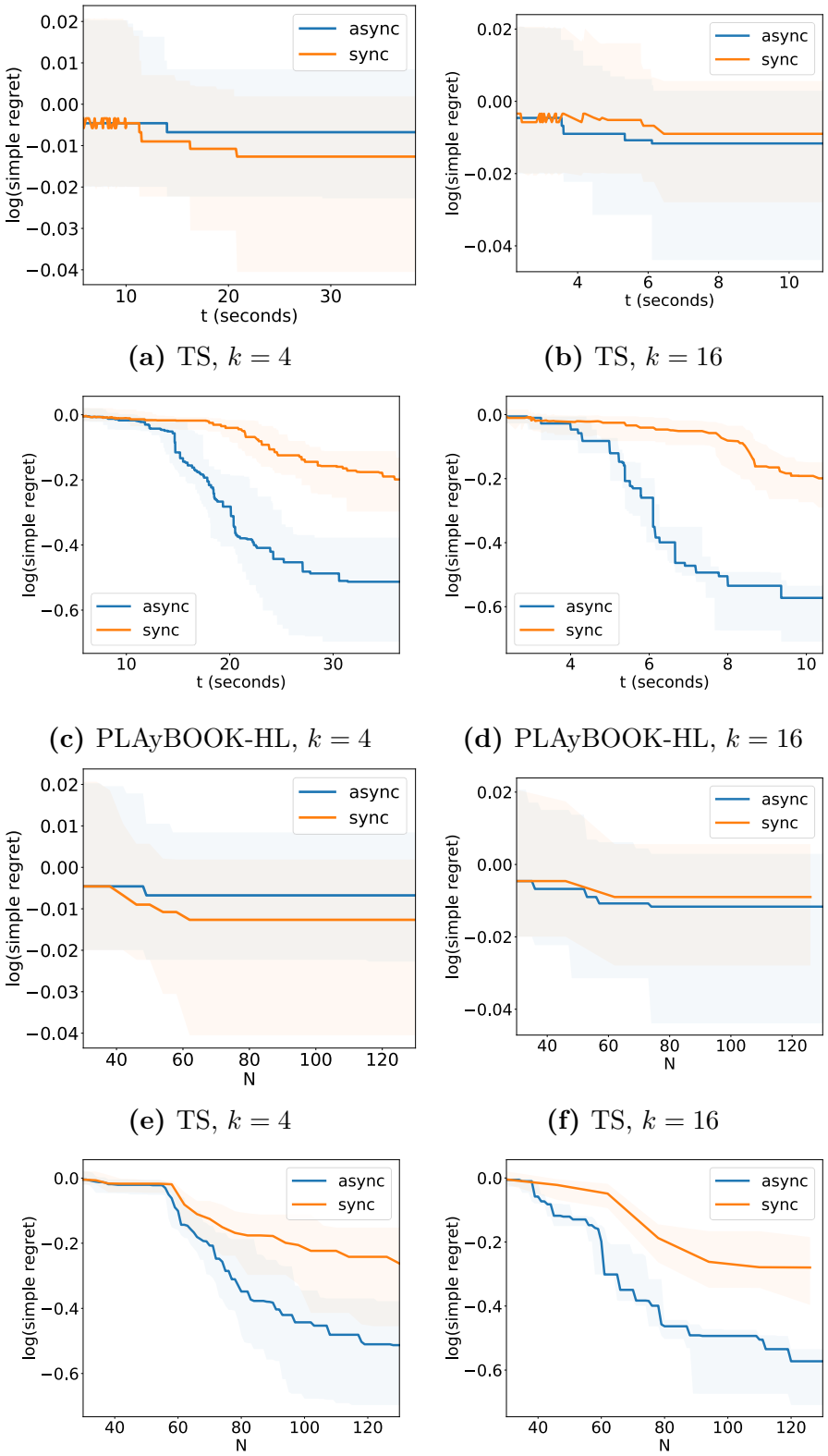
Similar to Figure 4.4 and 4.5 in Section 4.6.2, Figure A.6 and A.7 show head-to-head comparisons of synchronous and asynchronous methods but on the ack-10 task.

A.3.2 Asynchronous Parallel BOs

We conducted a large number of experiments testing the different approaches for asynchronous parallel BO methods. We computed the mean and standard deviation of the log of the simple regret across 30 random initialisations. Tables A.3, A.4 and A.5 show the results after 50, 75 and 100 asynchronous BO steps respectively.

Across all of these experiments, we can see that PLAyBOOK is performing competitively, making it an attractive choice for asynchronous BO problems.

[1]Details for these and other challenging global optimisation test functions can be found at https://www.sfu.ca/~ssurjano/optimization.html



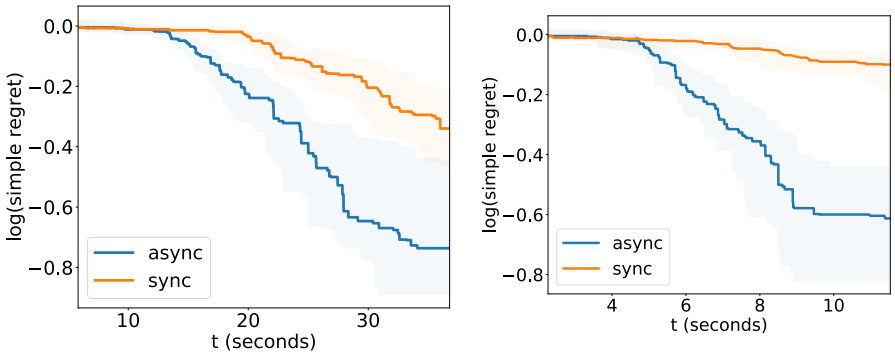
(a) TS, $k = 4$ **(b)** TS, $k = 16$

(c) PLAyBOOK-HL, $k = 4$ **(d)** PLAyBOOK-HL, $k = 16$

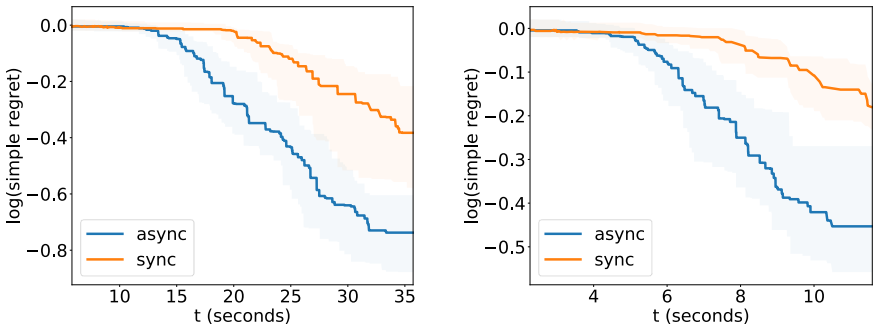
(e) TS, $k = 4$ **(f)** TS, $k = 16$

(g) PLAyBOOK-HL, $k = 4$ **(h)** PLAyBOOK-HL, $k = 16$

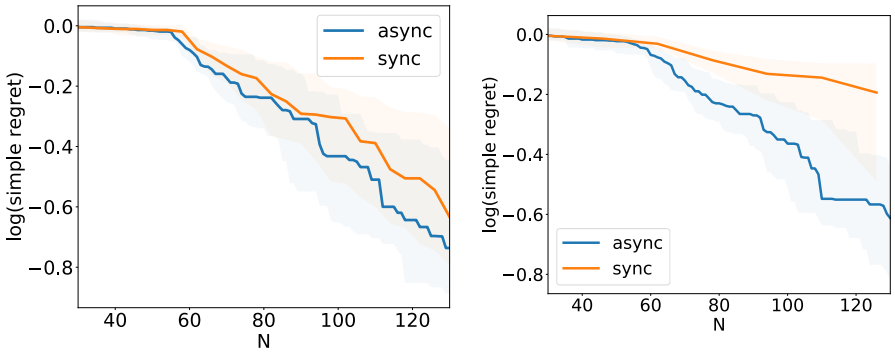
Figure A.6: A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO on ack-10 over *evaluation time, t*.



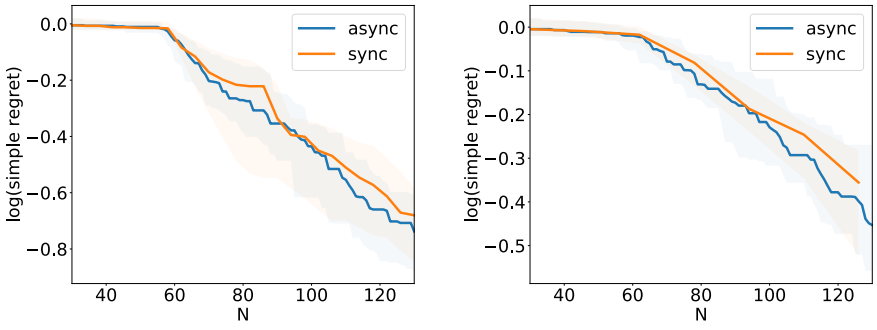
(a) PLAyBOOK-L, $k = 4$ **(b)** PLAyBOOK-L, $k = 16$



(c) KB, $k = 4$ **(d)** KB, $k = 16$



(e) PLAyBOOK-L, $k = 4$ **(f)** PLAyBOOK-L, $k = 16$



(g) KB, $k = 4$ **(h)** KB, $k = 16$

Figure A.7: A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO on ack-10 over *number of evaluations, N*.

Table A.3: Mean and s.t.d. of the log(regret) after 50 steps of asynchronous BO.

k	Task	KB	TS	PLAyBOOK			
				L	LL	H	HL
2	ack-10	-0.30 (0.16)	-0.01 (0.03)	-0.23 (0.15)	**-0.37 (0.25)**	-0.32 (0.18)	-0.27 (0.18)
	ack-5	-0.55 (0.39)	-0.22 (0.20)	**-0.85 (0.52)**	-0.52 (0.40)	-0.64 (0.39)	-0.71 (0.48)
	egg-2	0.28 (0.72)	0.89 (0.92)	-0.12 (1.26)	-0.05 (1.03)	**-0.44 (2.13)**	-0.31 (1.62)
	mat-2	0.81 (0.30)	1.05 (0.26)	**0.78 (0.28)**	0.82 (0.26)	0.81 (0.21)	0.89 (0.20)
	mat-6	1.02 (0.16)	1.13 (0.14)	**0.85 (0.43)**	0.92 (0.33)	0.88 (0.26)	0.86 (0.31)
	mic-10	1.84 (0.08)	1.92 (0.07)	**1.78 (0.11)**	1.80 (0.09)	1.82 (0.11)	1.83 (0.10)
	mic-5	0.71 (0.28)	1.02 (0.13)	0.67 (0.27)	**0.66 (0.26)**	0.69 (0.33)	0.87 (0.16)
	nrobot-4	-0.71 (1.05)	-0.52 (1.21)	-0.88 (0.91)	-0.78 (0.86)	**-0.89 (0.69)**	-0.87 (0.91)
4	ack-10	-0.29 (0.17)	-0.01 (0.03)	-0.26 (0.16)	-0.32 (0.17)	-0.25 (0.14)	**-0.34 (0.18)**
	ack-5	-0.54 (0.36)	-0.21 (0.13)	-0.66 (0.45)	-0.41 (0.27)	-0.53 (0.41)	**-0.71 (0.52)**
	egg-2	0.19 (1.06)	0.79 (0.92)	0.53 (0.91)	0.23 (0.98)	0.29 (0.86)	**-0.06 (1.26)**
	mat-2	**0.77 (0.30)**	0.98 (0.37)	0.94 (0.22)	1.01 (0.28)	0.82 (0.25)	0.87 (0.22)
	mat-6	1.06 (0.17)	1.13 (0.05)	0.99 (0.18)	1.01 (0.26)	0.95 (0.23)	**0.94 (0.20)**
	mic-10	1.82 (0.11)	1.92 (0.07)	1.82 (0.08)	1.83 (0.08)	**1.78 (0.09)**	1.80 (0.08)
	mic-5	**0.66 (0.28)**	1.01 (0.16)	0.78 (0.23)	0.87 (0.19)	0.76 (0.20)	0.82 (0.33)
	nrobot-4	-0.67 (1.00)	-0.39 (1.04)	**-1.01 (0.94)**	-0.94 (0.97)	-0.75 (0.70)	-0.63 (0.79)
6	ack-10	-0.24 (0.19)	-0.01 (0.04)	-0.29 (0.15)	-0.25 (0.14)	-0.20 (0.14)	**-0.31 (0.16)**
	ack-5	-0.51 (0.28)	-0.20 (0.20)	**-0.54 (0.40)**	-0.27 (0.18)	-0.35 (0.24)	-0.49 (0.27)
	egg-2	0.37 (1.05)	0.78 (0.88)	0.71 (0.83)	0.77 (0.82)	**0.07 (1.25)**	0.32 (1.11)
	mat-2	0.85 (0.25)	1.04 (0.19)	0.98 (0.34)	1.06 (0.23)	**0.84 (0.23)**	0.85 (0.25)
	mat-6	1.01 (0.17)	1.07 (0.18)	**0.95 (0.28)**	1.02 (0.28)	1.03 (0.16)	0.99 (0.24)
	mic-10	1.84 (0.08)	1.92 (0.07)	1.84 (0.08)	1.81 (0.08)	1.84 (0.08)	**1.80 (0.09)**
	mic-5	**0.76 (0.21)**	1.04 (0.15)	0.84 (0.23)	0.89 (0.24)	0.88 (0.19)	0.87 (0.20)
	nrobot-4	-0.58 (0.96)	-0.22 (0.95)	**-0.69 (1.04)**	-0.47 (0.85)	-0.63 (1.02)	-0.51 (0.99)
8	ack-10	-0.23 (0.17)	-0.01 (0.03)	-0.26 (0.17)	-0.34 (0.15)	-0.17 (0.11)	**-0.40 (0.14)**
	ack-5	-0.44 (0.27)	-0.19 (0.14)	-0.65 (0.40)	-0.33 (0.21)	-0.38 (0.30)	**-0.68 (0.37)**
	egg-2	0.31 (0.93)	0.65 (0.92)	0.60 (1.20)	0.93 (0.77)	0.42 (0.61)	**0.17 (1.05)**
	mat-2	0.80 (0.22)	0.93 (0.26)	0.81 (0.44)	1.09 (0.20)	0.87 (0.27)	**0.79 (0.30)**
	mat-6	**1.01 (0.19)**	1.13 (0.06)	1.03 (0.17)	1.02 (0.22)	1.03 (0.15)	1.03 (0.20)
	mic-10	1.84 (0.09)	1.91 (0.07)	1.82 (0.08)	1.81 (0.10)	1.80 (0.13)	**1.78 (0.08)**
	mic-5	0.70 (0.33)	1.02 (0.20)	**0.68 (0.32)**	0.79 (0.25)	0.73 (0.32)	0.69 (0.32)
	nrobot-4	-0.61 (0.78)	-0.20 (0.96)	**-0.94 (0.92)**	-0.62 (1.14)	-0.76 (1.11)	-0.39 (0.73)
16	ack-10	-0.13 (0.08)	-0.02 (0.04)	-0.24 (0.13)	-0.24 (0.13)	-0.22 (0.10)	**-0.44 (0.13)**
	ack-5	-0.36 (0.30)	-0.18 (0.16)	-0.49 (0.34)	-0.36 (0.24)	-0.28 (0.19)	**-0.73 (0.26)**
	egg-2	0.58 (0.73)	0.56 (1.10)	1.04 (0.51)	1.22 (0.50)	**0.30 (0.96)**	0.71 (0.53)
	mat-2	**0.82 (0.27)**	0.89 (0.24)	1.07 (0.17)	1.12 (0.20)	0.92 (0.31)	0.90 (0.31)
	mat-6	1.10 (0.05)	1.14 (0.03)	1.05 (0.14)	1.06 (0.21)	1.02 (0.26)	**1.00 (0.26)**
	mic-10	1.86 (0.08)	1.90 (0.07)	1.82 (0.07)	1.82 (0.09)	1.83 (0.08)	**1.79 (0.09)**
	mic-5	**0.81 (0.25)**	1.02 (0.19)	0.85 (0.21)	0.84 (0.23)	0.81 (0.22)	0.82 (0.25)
	nrobot-4	-0.43 (0.94)	-0.22 (0.99)	**-0.78 (0.95)**	-0.14 (0.77)	-0.37 (0.80)	-0.40 (0.81)

Table A.4: Mean and s.t.d. of the log(regret) after 75 steps of asynchronous BO.

k	Task	KB	TS	PLAyBOOK			
				L	LL	H	HL
2	ack-10	-0.43 (0.18)	-0.01 (0.03)	-0.48 (0.27)	**-0.58 (0.26)**	-0.55 (0.32)	-0.45 (0.28)
	ack-5	-0.91 (0.56)	-0.32 (0.22)	**-1.15 (0.58)**	-0.76 (0.50)	-1.03 (0.52)	-0.92 (0.51)
	egg-2	-0.12 (0.92)	0.87 (0.91)	-0.59 (1.16)	**-1.13 (2.14)**	-0.81 (1.99)	-0.82 (1.68)
	mat-2	0.80 (0.30)	1.05 (0.27)	**0.76 (0.28)**	0.81 (0.26)	0.81 (0.21)	0.87 (0.20)
	mat-6	0.95 (0.21)	1.17 (0.14)	**0.74 (0.53)**	0.87 (0.31)	0.84 (0.34)	0.89 (0.29)
	mic-10	1.79 (0.12)	1.92 (0.07)	**1.75 (0.11)**	1.76 (0.14)	1.79 (0.12)	1.79 (0.13)
	mic-5	**0.52 (0.42)**	0.97 (0.17)	0.61 (0.29)	0.56 (0.24)	0.57 (0.37)	0.73 (0.28)
	nrobot-4	-1.06 (1.08)	-0.83 (1.18)	-1.20 (0.86)	**-1.31 (0.75)**	-1.24 (0.78)	-1.29 (0.80)
4	ack-10	**-0.52 (0.21)**	-0.01 (0.03)	-0.51 (0.27)	-0.42 (0.19)	-0.46 (0.22)	-0.50 (0.22)
	ack-5	-0.83 (0.48)	-0.31 (0.23)	**-1.10 (0.53)**	-0.57 (0.33)	-0.75 (0.62)	-0.90 (0.54)
	egg-2	-0.19 (0.87)	0.69 (1.13)	0.16 (1.70)	-0.81 (2.55)	-0.23 (1.10)	**-0.89 (2.48)**
	mat-2	**0.75 (0.29)**	0.98 (0.37)	0.93 (0.22)	1.01 (0.28)	0.80 (0.25)	0.85 (0.22)
	mat-6	0.97 (0.27)	1.18 (0.06)	0.95 (0.23)	1.02 (0.24)	**0.86 (0.28)**	0.87 (0.30)
	mic-10	1.77 (0.12)	1.92 (0.07)	1.77 (0.09)	1.78 (0.09)	**1.74 (0.09)**	1.77 (0.10)
	mic-5	**0.56 (0.30)**	0.97 (0.14)	0.69 (0.25)	0.76 (0.19)	0.63 (0.22)	0.67 (0.34)
	nrobot-4	-0.92 (0.95)	-1.01 (1.31)	**-1.51 (0.88)**	-1.23 (0.89)	-1.07 (0.68)	-1.04 (0.79)
6	ack-10	-0.41 (0.22)	-0.01 (0.04)	**-0.50 (0.26)**	-0.30 (0.15)	-0.32 (0.18)	-0.38 (0.19)
	ack-5	-0.83 (0.46)	-0.32 (0.23)	**-0.86 (0.54)**	-0.36 (0.23)	-0.52 (0.31)	-0.76 (0.37)
	egg-2	0.19 (1.00)	0.64 (1.14)	0.56 (0.93)	0.75 (0.84)	**-0.34 (1.48)**	-0.21 (1.03)
	mat-2	**0.82 (0.26)**	1.03 (0.20)	0.98 (0.34)	1.06 (0.23)	0.82 (0.22)	0.84 (0.25)
	mat-6	0.92 (0.25)	1.14 (0.17)	**0.86 (0.36)**	1.08 (0.25)	0.95 (0.28)	0.94 (0.26)
	mic-10	1.81 (0.08)	1.92 (0.07)	1.81 (0.08)	1.79 (0.09)	1.80 (0.10)	**1.78 (0.10)**
	mic-5	**0.63 (0.28)**	1.00 (0.14)	0.74 (0.29)	0.82 (0.22)	0.76 (0.26)	0.69 (0.33)
	nrobot-4	-1.02 (0.88)	-0.73 (1.20)	-0.99 (1.04)	-0.78 (0.96)	**-1.04 (1.02)**	-1.02 (0.95)
8	ack-10	-0.44 (0.21)	-0.01 (0.03)	-0.47 (0.21)	-0.40 (0.20)	-0.28 (0.17)	**-0.52 (0.16)**
	ack-5	-0.77 (0.40)	-0.33 (0.20)	**-1.04 (0.45)**	-0.39 (0.22)	-0.52 (0.41)	-0.92 (0.36)
	egg-2	**-0.11 (0.96)**	0.57 (1.00)	0.40 (1.22)	0.85 (0.84)	0.22 (0.51)	-0.05 (0.96)
	mat-2	0.78 (0.23)	0.91 (0.26)	0.81 (0.44)	1.09 (0.20)	0.82 (0.26)	**0.76 (0.30)**
	mat-6	0.95 (0.24)	1.18 (0.10)	0.97 (0.20)	1.10 (0.20)	**0.93 (0.25)**	0.97 (0.28)
	mic-10	1.80 (0.09)	1.91 (0.07)	**1.75 (0.09)**	1.78 (0.09)	1.78 (0.13)	1.76 (0.08)
	mic-5	0.58 (0.38)	0.98 (0.19)	0.59 (0.36)	0.70 (0.26)	0.61 (0.33)	**0.55 (0.53)**
	nrobot-4	-0.92 (0.89)	-0.65 (1.01)	**-1.25 (0.82)**	-0.92 (1.08)	-1.07 (1.10)	-0.92 (0.88)
16	ack-10	-0.27 (0.14)	-0.02 (0.04)	-0.43 (0.21)	-0.31 (0.20)	-0.31 (0.14)	**-0.54 (0.16)**
	ack-5	-0.55 (0.34)	-0.29 (0.19)	-0.77 (0.42)	-0.39 (0.25)	-0.39 (0.28)	**-0.94 (0.30)**
	egg-2	0.26 (0.74)	0.41 (1.18)	0.89 (0.63)	1.21 (0.50)	**-0.39 (1.81)**	0.42 (0.61)
	mat-2	**0.79 (0.27)**	0.88 (0.24)	1.07 (0.17)	1.12 (0.20)	0.87 (0.30)	0.86 (0.31)
	mat-6	1.03 (0.20)	1.20 (0.03)	0.95 (0.21)	1.08 (0.25)	**0.94 (0.26)**	0.96 (0.36)
	mic-10	1.83 (0.09)	1.90 (0.07)	1.79 (0.08)	1.79 (0.09)	1.79 (0.10)	**1.75 (0.13)**
	mic-5	**0.66 (0.27)**	0.95 (0.21)	0.77 (0.25)	0.73 (0.39)	0.75 (0.23)	0.75 (0.25)
	nrobot-4	-0.72 (0.87)	-0.80 (1.24)	**-1.00 (1.01)**	-0.61 (0.95)	-0.72 (0.61)	-0.79 (0.78)

Table A.5: Mean and s.t.d. of the log(regret) after 100 steps of asynchronous BO.

k	Task	KB	TS	PLAyBOOK			
				L	LL	H	HL
2	ack-10	-0.67 (0.27)	-0.01 (0.03)	-0.72 (0.34)	-0.73 (0.29)	**-0.80 (0.35)**	-0.58 (0.26)
	ack-5	-1.28 (0.70)	-0.35 (0.22)	**-1.49 (0.65)**	-0.95 (0.53)	-1.39 (0.62)	-1.08 (0.47)
	egg-2	-0.42 (1.62)	0.80 (0.92)	-1.10 (1.56)	**-1.58 (2.49)**	-1.54 (2.28)	-1.30 (2.01)
	mat-2	0.79 (0.30)	1.04 (0.25)	**0.76 (0.28)**	0.81 (0.26)	0.81 (0.21)	0.87 (0.19)
	mat-6	1.00 (0.21)	1.27 (0.17)	**0.93 (0.36)**	1.00 (0.31)	0.94 (0.26)	1.01 (0.27)
	mic-10	1.76 (0.13)	1.92 (0.07)	**1.72 (0.10)**	1.73 (0.13)	1.72 (0.13)	1.76 (0.12)
	mic-5	**0.39 (0.51)**	0.94 (0.18)	0.54 (0.31)	0.45 (0.24)	0.45 (0.36)	0.65 (0.28)
	nrobot-4	-1.33 (1.02)	-1.12 (1.21)	-1.54 (0.84)	-1.51 (0.85)	-1.46 (0.74)	**-1.58 (0.78)**
4	ack-10	**-0.72 (0.24)**	-0.01 (0.03)	-0.68 (0.28)	-0.54 (0.24)	-0.67 (0.24)	-0.56 (0.24)
	ack-5	-1.13 (0.59)	-0.41 (0.25)	**-1.46 (0.59)**	-0.68 (0.32)	-0.98 (0.73)	-1.04 (0.52)
	egg-2	-0.27 (0.82)	0.65 (1.12)	-0.16 (1.71)	-1.09 (2.57)	-0.50 (1.03)	**-1.17 (2.53)**
	mat-2	**0.74 (0.29)**	0.98 (0.37)	0.93 (0.22)	1.01 (0.28)	0.79 (0.25)	0.85 (0.22)
	mat-6	1.03 (0.25)	1.30 (0.06)	1.07 (0.19)	1.12 (0.20)	**0.98 (0.25)**	1.01 (0.23)
	mic-10	1.74 (0.11)	1.92 (0.07)	1.75 (0.09)	1.74 (0.10)	**1.70 (0.09)**	1.74 (0.11)
	mic-5	**0.44 (0.32)**	0.96 (0.14)	0.55 (0.27)	0.63 (0.21)	0.50 (0.27)	0.63 (0.34)
	nrobot-4	-1.14 (0.88)	-1.24 (1.23)	**-1.73 (0.88)**	-1.48 (0.72)	-1.32 (0.83)	-1.24 (0.83)
6	ack-10	-0.63 (0.27)	-0.01 (0.04)	**-0.65 (0.29)**	-0.37 (0.20)	-0.44 (0.25)	-0.43 (0.20)
	ack-5	**-1.19 (0.60)**	-0.39 (0.26)	-1.17 (0.65)	-0.39 (0.22)	-0.68 (0.41)	-0.87 (0.40)
	egg-2	-0.18 (1.05)	0.57 (1.15)	0.50 (0.94)	0.65 (0.98)	**-0.53 (1.40)**	-0.45 (0.99)
	mat-2	0.82 (0.25)	1.02 (0.20)	0.98 (0.34)	1.06 (0.23)	**0.81 (0.22)**	0.83 (0.24)
	mat-6	1.02 (0.24)	1.25 (0.16)	**0.99 (0.28)**	1.18 (0.27)	1.05 (0.23)	1.03 (0.23)
	mic-10	1.77 (0.10)	1.92 (0.07)	**1.76 (0.08)**	1.76 (0.08)	1.76 (0.13)	1.77 (0.10)
	mic-5	**0.51 (0.34)**	0.96 (0.15)	0.66 (0.27)	0.72 (0.25)	0.66 (0.37)	0.63 (0.30)
	nrobot-4	-1.27 (0.80)	-1.12 (1.28)	-1.24 (0.89)	-1.03 (0.94)	**-1.35 (0.95)**	-1.27 (1.00)
8	ack-10	-0.57 (0.22)	-0.01 (0.03)	-0.62 (0.26)	-0.44 (0.23)	-0.41 (0.25)	**-0.63 (0.16)**
	ack-5	-1.16 (0.47)	-0.42 (0.24)	**-1.32 (0.46)**	-0.47 (0.23)	-0.70 (0.51)	-1.12 (0.41)
	egg-2	-0.24 (0.91)	0.42 (1.13)	-0.02 (2.20)	0.78 (0.86)	-0.04 (0.54)	**-0.27 (0.85)**
	mat-2	0.78 (0.23)	0.90 (0.25)	0.81 (0.44)	1.09 (0.20)	0.81 (0.26)	**0.75 (0.30)**
	mat-6	**1.01 (0.23)**	1.30 (0.09)	1.07 (0.17)	1.23 (0.17)	1.02 (0.19)	1.04 (0.25)
	mic-10	1.78 (0.08)	1.91 (0.07)	**1.72 (0.10)**	1.75 (0.09)	1.74 (0.13)	1.74 (0.08)
	mic-5	**0.46 (0.37)**	0.93 (0.19)	0.53 (0.35)	0.68 (0.26)	0.50 (0.32)	0.50 (0.52)
	nrobot-4	-1.24 (0.76)	-1.18 (1.23)	**-1.44 (0.84)**	-1.17 (0.98)	-1.35 (1.02)	-1.13 (0.85)
16	ack-10	-0.44 (0.20)	-0.02 (0.04)	**-0.63 (0.24)**	-0.37 (0.18)	-0.40 (0.18)	-0.61 (0.16)
	ack-5	-0.83 (0.43)	-0.41 (0.23)	-1.03 (0.51)	-0.41 (0.24)	-0.43 (0.30)	**-1.11 (0.35)**
	egg-2	-0.41 (2.18)	0.34 (1.17)	0.83 (0.67)	1.21 (0.49)	**-0.86 (2.23)**	-0.53 (2.03)
	mat-2	**0.79 (0.26)**	0.88 (0.24)	1.07 (0.16)	1.12 (0.20)	0.86 (0.29)	0.83 (0.31)
	mat-6	1.09 (0.18)	1.31 (0.04)	1.02 (0.21)	1.21 (0.20)	**1.01 (0.23)**	1.02 (0.42)
	mic-10	1.80 (0.11)	1.90 (0.07)	1.75 (0.09)	1.77 (0.10)	1.76 (0.10)	**1.72 (0.13)**
	mic-5	**0.53 (0.33)**	0.91 (0.21)	0.68 (0.25)	0.67 (0.38)	0.66 (0.28)	0.60 (0.36)
	nrobot-4	-1.10 (0.86)	-1.15 (1.20)	**-1.39 (0.95)**	-0.95 (1.02)	-0.86 (0.60)	-1.07 (0.92)

A.4 Additional Experiments for BayesOpt Attack

Table A.6: Summary of attack results on 7 MNIST images. Q denotes the query count. *ASR* denotes attack success rate. The standard errors are in parentheses.

Attack method	d_r	*ASR*	*Q (Max, Median, Mean)*	Average L_2 perturbation (per pixel)
GP-BO	$14 \times 14 \times 1$	92%	$899, 53, 119(\pm 28)$	$7.01 \times 10^{-3}(\pm 1.13 \times 10^{-4})$
ADDGP-BO	$14 \times 14 \times 1$	98%	$584, 34, 67(\pm 13)$	$6.50 \times 10^{-3}(\pm 8.39 \times 10^{-5})$
AutoZOOM	$14 \times 14 \times 1$	97%	$892, 186, 247(\pm 27)$	$5.10 \times 10^{-3}(\pm 1.92 \times 10^{-4})$
GenAttack	$14 \times 14 \times 1$	97%	$976, 184, 239(\pm 22)$	$5.56 \times 10^{-3}(\pm 9.25 \times 10^{-5})$
GP-BO	$16 \times 16 \times 1$	97%	$400, 50, 80(\pm 12)$	$7.08 \times 10^{-3}(\pm 1.09 \times 10^{-4})$
ADDGP-BO	$16 \times 16 \times 1$	100%	$540, 42, 72(\pm 12)$	$6.49 \times 10^{-3}(\pm 9.41 \times 10^{-5})$
AutoZOOM	$16 \times 16 \times 1$	98%	$812, 251, 295(\pm 28)$	$5.98 \times 10^{-3}(\pm 2.03 \times 10^{-4})$
GenAttack	$16 \times 16 \times 1$	97%	$928, 202, 237(\pm 19)$	$5.61 \times 10^{-3}(\pm 8.58 \times 10^{-5})$
GP-BO	$28 \times 28 \times 1$	98%	$430, 201, 225(\pm 11)$	$8.71 \times 10^{-3}(\pm 1.94 \times 10^{-4})$
ADDGP-BO	$28 \times 28 \times 1$	100%	$666, 57, 100(\pm 15)$	$9.71 \times 10^{-3}(\pm 1.12 \times 10^{-4})$
AutoZOOM	$28 \times 28 \times 1$	98%	$860, 185, 244(\pm 27)$	$1.01 \times 10^{-2}(\pm 1.63 \times 10^{-4})$
GenAttack	$28 \times 28 \times 1$	83%	$976, 365, 399(\pm 33)$	$6.26 \times 10^{-3}(\pm 6.69 \times 10^{-5})$

Table A.7: Summary of attack results on 27 CIFAR10 images. Q denotes the query count. *ASR* denotes attack success rate. The standard errors are in parentheses.

Attack method	d_r	*ASR*	*Q (Max, Median, Mean)*	Average L_2 perturbation (per pixel)
GP-BO	$8 \times 8 \times 3$	52%	$314, 48, 70(\pm 13)$	$5.78 \times 10^{-4}(\pm 1.44 \times 10^{-5})$
ADDGP-BO	$8 \times 8 \times 3$	75%	$899, 140, 234(\pm 33)$	$5.55 \times 10^{-4}(\pm 8.46 \times 10^{-6})$
AutoZOOM	$8 \times 8 \times 3$	56%	$382, 126, 139(\pm 17)$	$9.58 \times 10^{-4}(\pm 2.66 \times 10^{-5})$
GenAttack	$8 \times 8 \times 3$	67%	$671, 236, 286(\pm 32)$	$7.40 \times 10^{-4}(\pm 1.30 \times 10^{-5})$
GP-BO	$14 \times 14 \times 3$	81%	$899, 142, 192(\pm 11)$	$5.74 \times 10^{-4}(\pm 6.96 \times 10^{-6})$
ADDGP-BO	$14 \times 14 \times 3$	90%	$885, 141, 213(\pm 15)$	$5.79 \times 10^{-4}(\pm 5.16 \times 10^{-6})$
AutoZOOM	$14 \times 14 \times 3$	42%	$376, 95, 133(\pm 12)$	$9.82 \times 10^{-4}(\pm 1.95 \times 10^{-5})$
GenAttack	$14 \times 14 \times 3$	75%	$971, 251, 321(\pm 21)$	$7.35 \times 10^{-4}(\pm 6.92 \times 10^{-6})$
GP-BO	$16 \times 16 \times 3$	83%	$785, 280, 292(\pm 22)$	$5.34 \times 10^{-4}(\pm 1.43 \times 10^{-5})$
ADDGP-BO	$16 \times 16 \times 3$	87%	$878, 168, 229(\pm 29)$	$5.76 \times 10^{-4}(\pm 8.93 \times 10^{-6})$
AutoZOOM	$16 \times 16 \times 3$	3%	$298, 170, 170(\pm 90)$	$7.50 \times 10^{-4}(\pm 1.35 \times 10^{-5})$
GenAttack	$16 \times 16 \times 3$	79%	$986, 281, 339(\pm 36)$	$7.36 \times 10^{-4}(\pm 1.55 \times 10^{-5})$

We conducted more experiments on MNIST and CIFAR images to further verify that the dimension of the latent space d^r has a large impact on the attack success rate, query efficiency and L_2 norm of the successful adversarial perturbations of

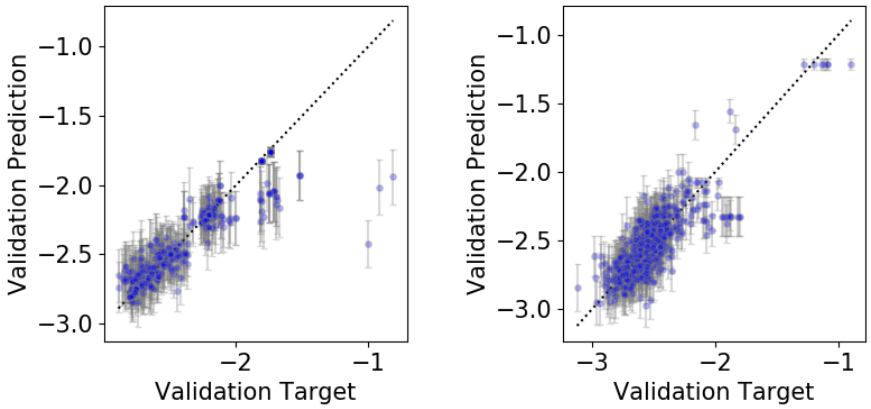
different attack methods and the optimal d^r varies across different victim models as well as attack methods.

A.5 Combining Different Kernels to Improve GPWL

Table A.8: Regression performance (i.t.o rank correlation) of additive kernels

Kernel	NB101-CIFAR10	RWNN-Flower102
WL + MLP	**0.871**±0.02	**0.813**±0.018
WL†	0.862±0.03	0.804±0.018
MLP†	0.458±0.07	0.492±0.12

†: Taken directly from Table 7.1.



(a) NB101-CIFAR10 **(b)** RWNN-Flower102

Figure A.8: Predictive vs ground-truth validation error of GPWL with additive kernel on NB101-CIFAR10 and RWNN-Flower102 in log-log scale. Error bar denotes ±1 SD from the GP posterior predictive distribution.

In general, the sum or product of valid kernels gives another valid kernel, as such, combining different kernels to yield a better-performing kernel is commonly used in GP and Multiple Kernel Learning (MKL) literature (Rasmussen, 2003; Gönen and Alpaydın, 2011). In this section, we conduct a preliminary discussion on its usefulness to GPWL. As a singular example, we consider the *additive* kernel

that is a linear combination of the WL kernel and the MLP kernel:

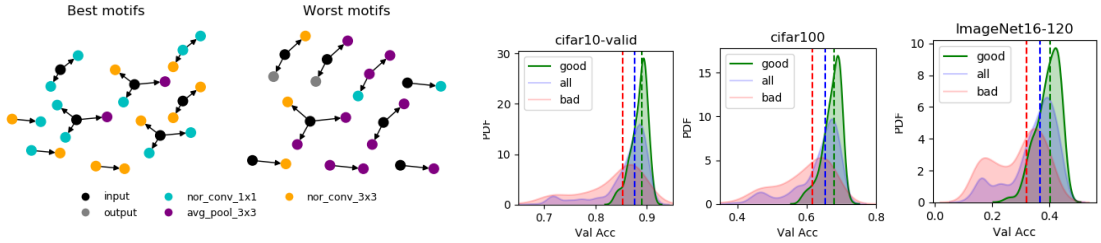
$$k_{\text{add}}(G_1, G_2) = \alpha k_{\text{WL}}(G_1, G_2) + \beta k_{\text{MLP}}(G_1, G_2) \text{ s.t. } \alpha + \beta = 1,\ \alpha, \beta \geq 0 \quad \text{(A.12)}$$

where α, β are the kernel weights. We choose WL and MLP because we expect them to extract diverse information: whereas WL processes the graph node information directly, MLP consider the spectrum of the graph Laplacian matrix, which often reflect the global properties such as the topology and the graph connectivity. We expect the more diverse features captured by the constituent kernels will lead to a more effective additive kernel. While it is possible to determine the weights in a more principled way such as jointly optimising them in the GP log-marginal likelihood, in this example we simply set $\alpha = 0.7$ and $\beta = 0.3$. We then perform regression on NB101-CIFAR10 and RWNN-Flower102 datasets following the setup as in Section 7.5.1. We repeat each experiment 20 times and report the mean and standard deviation in Table A.8, and we show the uncertainty estimate of additive kernel in Figure A.8. In both search spaces the additive kernel outperforms the constituent kernels but the gain over the WL kernel is marginal. Interestingly, while MLP performs poorly on its own, it can be seen that the complementary spectral information extracted by it can be helpful when used alongside our WL kernel. Generally, we hypothesise that as the search space increases in complexity (e.g., larger graphs, more edge connections permitted, etc), we expect that the benefits from combining different kernels to increase and we defer a more comprehensive discussion on this to a future work. As a starting point, one concrete proposal would be applying a MKL method such as ALIGNF (Cortes et al., 2012) in our context directly.

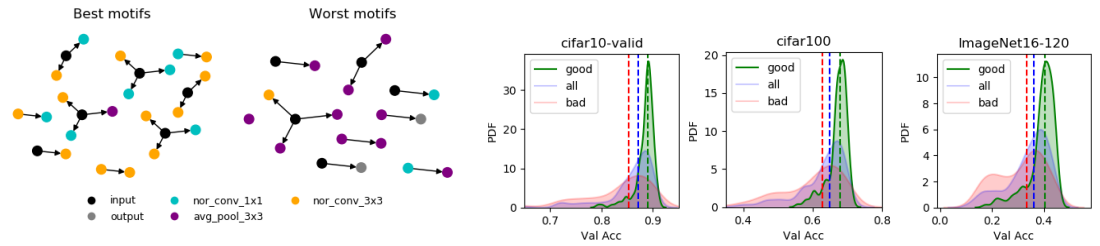
A.6 More Motif Discovery on NB101 and Other NB201 Tasks

Supplementary to Figure 7.2 in Section 7.3.3, here we outline the motifs discovered by GPWL also on the NB101 search space and on the other NB201 tasks (CIFAR100, ImageNet120) in Figure A.9. We follow the identical setup as described in Section

(a) Best and worst motifs identified on NB101-CIFAR10 (left), and the validation accuracy distributions in the validation architectures (right).



(b) Best and worst motifs identified on NB201-CIFAR100 (left), and the validation accuracy distributions transferred on CIFAR10, CIFAR100 and ImageNet120 of NB201 (right 3).

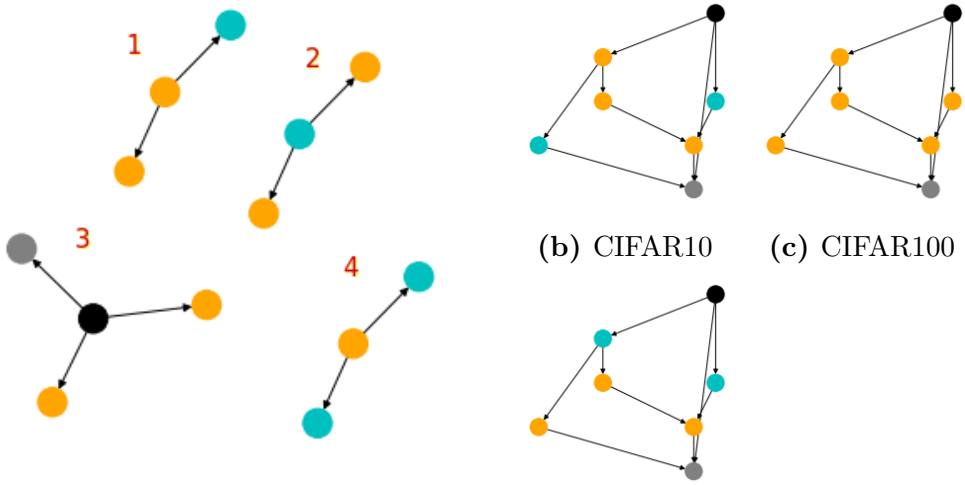


(c) Best and worst motifs identified on NB201-ImageNet120 (left), and the validation accuracy distributions transferred on CIFAR10, CIFAR100 and ImageNet120 of NB201 (right 3).

Figure A.9: Motif discovery on NB101 and CIFAR100 and ImageNet120 tasks of NB201. Note that since NB101 is trained on CIFAR10 only, it is not possible to show the results transferred on another task. All symbols and legends have the same meaning as in Figure 7.2 in Section 7.3.3.

7.3.3. In all cases, the motifs are highly effective in separating the architecture pool, and it is also noteworthy that the motifs found in the other NB201 tasks are highly consistent with those shown in Figure 7.2 in the main text with only minor differences, further supporting our claim that the GPWL is capable of identifying transferable features without unduly overfitting to a particular task.

To give further concrete evidence on the working and advantage of the proposed method in NB201, we show the top-4 motifs in terms of the derivatives computed from one experiment on CIFAR10 *only* in Figure A.10 (a) and show the ground-truth



(a) Top motifs searched on CIFAR10 *only*

(b) CIFAR10

(c) CIFAR100

(d) ImageNet120

Figure A.10: Computed motifs and ground-truth optimal cells for all 3 tasks of NB201. Note that optimal architecture cell for CIFAR10 contains motifs 1 and 3, optimal architecture cell for CIFAR100 contains motif 3 and optimal architecture cell for ImageNet120 contains motif 2.

best architecture cell in each of the three image tasks in Figure A.10 (b) to (d). In this case, while the optimal cells for the different tasks are similar (but not identical) and reflective of a high-level transferability of the cells, transferring the optimal cell in one task directly to another will be sub-optimal. However, using our method as described in Algorithm 11 by transferring the *motifs* in Figure A.10(a) onto CIFAR100 and ImageNet120 tasks, we reduce the search space and resultantly search time drastically (as any cell to be evaluated now needs to contain one of the top motifs in Figure A.10(a)) yet we do not preemptively rule out the optimal cell (as all optimal cells contain ≥ 1 "good" motifs). As such, our method strikes a balance between performance and efficiency.

A.7 Additional NAS-BOWL Experiments

We show the validation errors against number of evaluations using both stochastic and deterministic validation errors of NAS-Bench datasets in Figure A.11 and A.12. It is worth noting that regardless of whether the validation errors are stochastic or not, the test errors are always averaged to deterministic values for fair comparison.

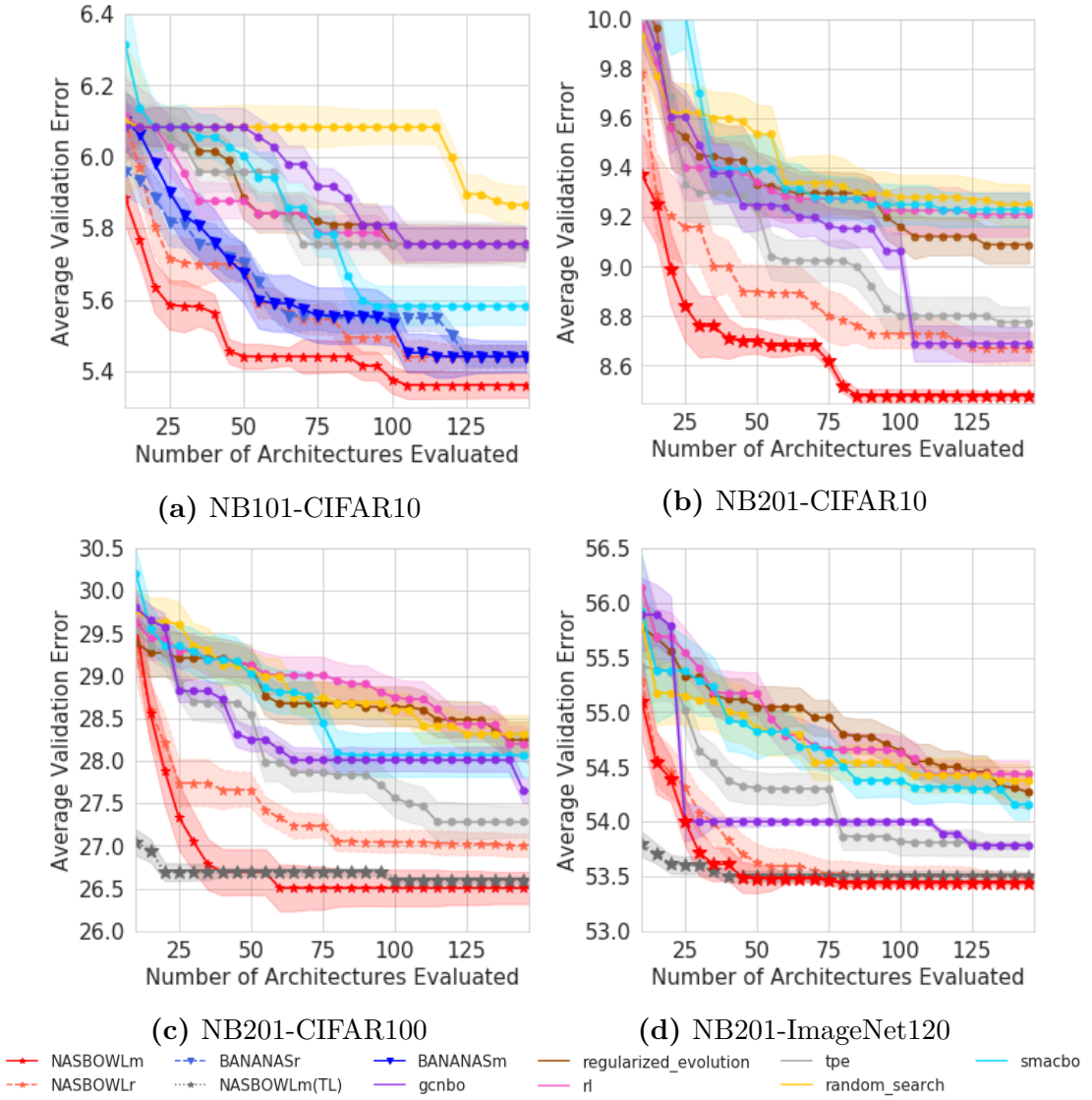
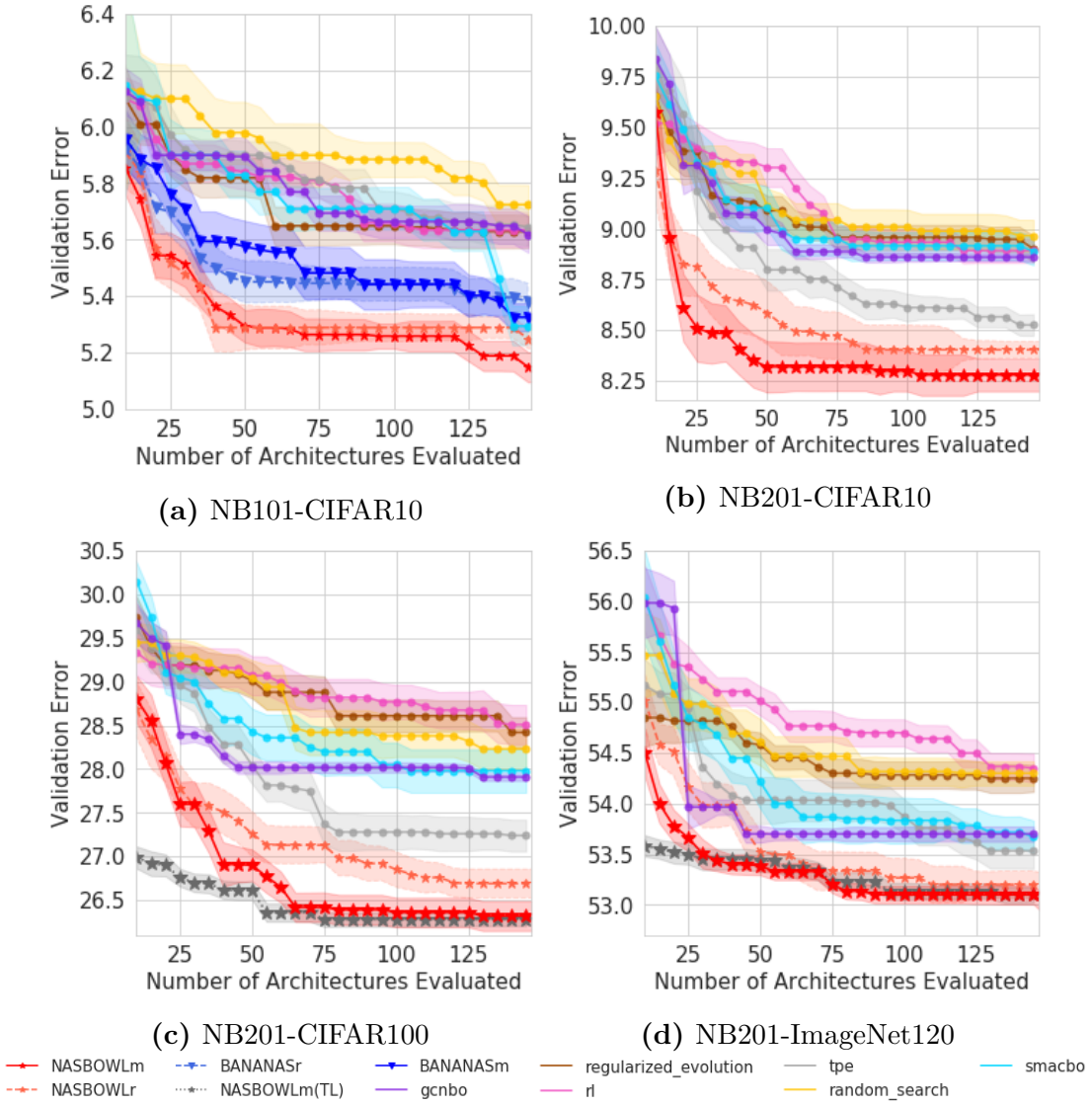


Figure A.11: Median validation error on NAS-Bench datasets with *deterministic* observations from 20 trials. Shades denote ±1 standard error.

It is obvious that NAS-BOWL still outperforms the other methods under this metric in achieving lower test error or enjoying faster convergence, or having both under most circumstances. This corresponds well with the results on the test error in Figure 7.7 and double-confirms the superior performance of our proposed NAS-BOWL in searching optimal architectures.



(a) NB101-CIFAR10 (b) NB201-CIFAR10

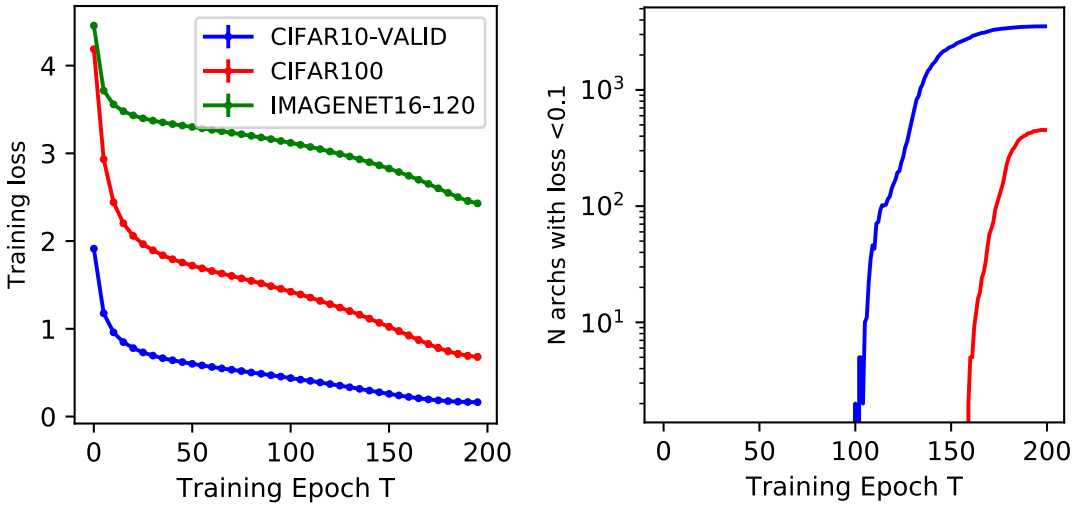
(c) NB201-CIFAR100 (d) NB201-ImageNet120

Figure A.12: Median validation error on NAS-Bench datasets with *noisy* observations from 20 trials. Shades denote ±1 standard error.

A.8 Additional Experiments for TSE

A.8.1 Overfitting on CIFAR10 and CIFAR100

In Figure 8.10 in Section 8.4.6, the rank correlation achieved by TSE-E on CIFAR10 and CIFAR100 drops slightly after around $T = 150$ epochs but a similar trend is not observed for ImageNet120. We hypothesise that this is because many architectures converge to very small training losses on CIFAR10 and CIFAR100 in the later training phase, making it more difficult to distinguish between these good architectures based on their later-epoch training losses. However, this does



(a) Training loss **(b)** No. of arch with loss< 0.1

Figure A.13: Mean and 5 standard error of training losses and validation losses on all architectures on different NB201 image datasets. (a) shows the training curves and (b) shows the number of architectures whose training losses go below 0.1 as the training proceeds. Many architectures reach very small training loss in the later phase of the training on CIFAR10 and CIFAR100, thus may overfitting on these two datasets. But all the architectures suffer high training losses on ImageNet120, which is a much more challenging classification task, and none of them overfits.

not happen on ImageNet120 because it is a more challenging dataset. We test this by visualising the training loss curves of all architectures in Figure A.13a, where the solid line and error bar correspond to the mean and standard error, respectively. We also plot out the number of architectures with training losses below 0.1 [2] in Figure A.13b. It is evident that CIFAR10 and CIFAR100 both see an increasing number of overfitted architectures as the training proceeds whereas all architectures still have high training losses on ImageNet120 at end of the training $T = 200$ and none have overfit. Thus, our hypothesis is confirmed. In addition, similar observation is also shared in (Jiang et al., 2020) where the authors find the number of optimisation iterations required to reach loss of 0.1 correlates well with generalisation but the number of iterations required to go from a loss of 0.1 to 0.01 does not.

[2]The threshold 0.1 is chosen following the threshold for optimisation-based measures in (Jiang et al., 2020)

B

Additional Informations

B.1 Kriging Believer

The Kriging Believer (KB) is a method proposed in (Ginsbourger et al., 2010a) to sequentially select batch points in the continuous space. In KB, as the name suggested, we fully trust the predictive posterior and use the posterior mean $\mu(\mathbf{x}_t^{(1)})$ at a selected batch location $\mathbf{x}_t^{(1)}$ as a proxy for the true function value $f(\mathbf{x}_t^{(1)})$. We then augment the observation data D_{t-1} with this hallucinated data $\{\mathbf{x}_t^{(1)}, \mu(\mathbf{x}_t^{(1)})\}$ to update the surrogate model as well as the the acquisition function. The next point in the batch is then selected by maximising the updated acquisition function. This process repeats until all b points in the batch are selected as shown in Algorithm 17.

Algorithm 17 Kriging Believer

Input: Observation data $\mathcal{D}_{t-1}$, batch size b
Output: The batch points $\mathcal{B}_t = \{\mathbf{x}_t^{(1)}, \ldots, \mathbf{x}_t^{(b)}\}$
$\mathcal{D}'_{t-1} = \mathcal{D}_{t-1}$
for $j = 1, \ldots, b$ **do**
  $\mathbf{x}_t^{(j)} = \arg\max \alpha(\mathbf{x}|\mathcal{D}'_{t-1})$
  Compute $\mu(\mathbf{x}_t^{(j)})$
  $\mathcal{D}'_{t-1} \leftarrow \mathcal{D}'_{t-1} \cup (\mathbf{x}_t, \mu(\mathbf{x}_t^{(j)}))$
end for

B.2 Algorithmic Description of the WL kernel

Complementary to Figure 7.1 in Section 7.3.1, here we include a formal, algorithmic description of the WL procedure in Algorithm 18.

Algorithm 18 Weisfeiler-Lehman subtree kernel computation between two graphs Shervashidze et al. (2011)

Input: Graphs $\{G_1, G_2\}$, Maximum WL iterations H
Output: The kernel function value between the graphs k
Initialise the feature vectors $\{\boldsymbol{\phi}(G_1), \boldsymbol{\phi}(G_2)\}$ with the respective counts of original node labels i.e. the $h = 0$ WL features. (E.g. $\phi^i(G_1)$ is the count of i-th node label of graph G_1)
for $h = 1, \ldots, H$ **do**
    Assign a multiset-label $M_h(v)$ to each node v in G consisting of the multiset $\{l_{h-1}|u \in \mathcal{N}(v)$, where $l_{h-1}(v)$ is the node label of node v of the $h-1$-th WL iteration; $\mathcal{N}(v)$ are the neighbour nodes of node v
    Sort each elements in $M_h(v)$ in ascending order and concatenate them into string $s_h(v)$
    Add $l_{h-1}(v)$ as a prefix to $s_h(v)$.
    Compress each string $s_h(v)$ using hash function f so that $f(s_h(v)) = f(s_h(w))$ iff $s_h(v) = s_h(w)$ for two nodes $\{v, w\}$.
    Set $l_h(v) := f(s_h(v)) \forall v \in G$.
    Concatenate the $\boldsymbol{\phi}(G_1), \boldsymbol{\phi}(G_2)$ with the respective counts of the *new* labels
end for
Compute inner product between the feature vectors in RKHS $k = \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$

B.3 Other Implementation Details of NAS-BOWL and Competing Baselines

NAS-BOWL We use a batch size $B = 5$ (i.e., at each BO iteration, architectures yielding top 5 acquisition function values are selected to be evaluated in parallel). When mutation algorithm described in Section 7.3.2 is used, we use a pool size of $P = 200$, and half of which is generated from mutating the top-10 best performing architectures already queried and the other half is generated from random sampling to encourage more explorations in NASBench-101. In NASBench-201, accounting for the much smaller search space and consequently the lesser need to exploration, we simply generate all architectures from mutation. For experiments with random

acquisition, we also use $P = 200$ throughout, and we also study the effect of varying P later in this section. We use WL with optimal assignment (OA) (Kriege et al., 2016) for all datasets apart from NASBench-201. Denoting the feature vectors of two graphs G_1 and G_2 as $\boldsymbol{\phi}(G_1)$ and $\boldsymbol{\phi}(G_2)$ respectively, the OA inner product in the WL case is given by the histogram intersection $\langle \boldsymbol{\phi}(G_1), \boldsymbol{\phi}(G_2) \rangle = \sum_j \min(\phi^j(G_1), \phi^j(G_2)$, where $\phi^j(\cdot)$ is the j-th element of the vector. On NASBench-201 which features a much smaller search space which we find a simple dot product of the feature vectors $\boldsymbol{\phi}(G_1)^{\mathrm{T}}\boldsymbol{\phi}(G_2)$ to perform empirically better. We always use 10 random samples to initialise NAS-BOWL.

On NASBench-101 dataset, we always apply pruning (which is available in the NASBench-101 API) to remove the invalid nodes and edges from the graphs. On NASBench-201 dataset, since the architectures are defined over a DARTS-like, edge-labelled search space, we first convert the edge-labelled graphs to node-labelled graphs as a pre-processing step. It is worth noting that it is possible to use WL kernel defined over edge-labelled graphs directly (e.g the WL-edge kernel proposed by (Shervashidze et al., 2011)), although in this paper we find the WL kernels over node-labelled graphs to perform empirically better.

On the transfer learning setup of N201, we first run a standard optimisation task on the CIFAR10 task (we term this the *base task*) but we allow for up to an expanded budget of 250 architecture evaluations to build up the confidence of GPWL derivative estimates. We then extract the "good motifs" identified by GPWL (i.e. those features with derivatives in the top quantile). For the subsequent CIFAR100/ImageNet120 optimisations (we term this the *transferred tasks*). On the transferred tasks, with everything else unmodified from standard runs (e.g. budget, pool size, batch size, acquisition function choice, etc), we additionally enforce the pruning rule such that only candidates in the pool with at least 1 match to the previously identified "good motifs" are allowed for evaluations and the rest are removed. The key difference is that under standard runs, the pool of size B is generated *once* per BO iteration via random sampling/mutation algorithm since all

candidates are accepted; here, this procedure is executed for many times as required until we have a pool of B architectures where each meets the pruning criteria.

BANANAS We use the code made public by the authors (White et al., 2021a) (https://github.com/naszilla/bananas), and use the default settings contained in the code with the exception of the number of architectures queried at each BO iteration (i.e. BO batch size): the default is 10, but to conform to our test settings we use 5 instead. While we do not change the default pool size of $P = 200$ at each BO iteration, instead of filling the pool entirely from mutation of the best architectures, we only mutate 100 architectures from top-10 best architectures and generate the other 100 randomly to enable a fair comparison with our method. It is worth noting that neither changes led to a significant deterioration in the performance of BANANAS: under the deterministic validation error setup, the results we report are largely consistent with the results reported in (White et al., 2021a); under the stochastic validation error setup, our BANANAS results actually slightly outperform results in the original paper. It is finally worth noting that the public implementation of BANANAS on NASBench-201 was not released by the authors.

GCNBO for NAS We implemented the GCN surrogate by ourselves following the description in the most recent work (Shi et al., 2019), which uses a graph convolution neural network in combination with a Bayesian linear regression layer to predict architecture performance in its BO-based NAS. To ensure fair comparison with our NAS-BOWL, we then define a normal Expected Improvement (EI) acquisition function based on the predictive distribution by the GCN surrogate to obtain another BO-based NAS baseline in Section 7.5.2, GCNBO. Similar to all the other baselines including our NASBOWLr and BANANASr, we use random sampling to generate candidate architectures for acquisition function optimisation. However, different from NAS-BOWL and BANANAS, GCNBO uses a batch size $B = 1$, i.e. at each BO iteration, NAS-BOWL and BANANAS select 5 new architectures to evaluate next but GCNBO select 1 new architecture to evaluate next. This setup should favour GCNBO if we measure the optimisation performance against the

number of architecture evaluations which is the metric used in Figs. 4 and 7.7 because at each BO iteration, GCNBO selects the next architecture G_t based on the most up-to-date information $\alpha_t(G|\mathcal{D}_{t-1})$ whereas NAS-BOWL and BANANAS only select one architecture $G_{t,1}$ in such fully informed way but select the other four architectures $\{G_{t,i}\}_{i=2}^5$ with outdated information. Specifically, in the sequential case ($B = 1$), $G_{t,2}$ is selected only after we have evaluated $G_{t,1}$, $G_{t,2}$ is selected by maximising $\alpha_t(G|\{\mathcal{D}_{t-1}, (G_{t,1}, y_{t,1})\})$; the same procedure applies for $G_{t,3}$, $G_{t,4}$ and $G_{t,5}$. However, in the batch case ($B = 5$) where $G_{t,i}$ for $2 \leq i \leq 5$ need to be selected before $G_{t,i-1}$ is evaluated, $\{G_{t,i}\}_{i=2}^5$ are all decided based on $\alpha_t(G|\mathcal{D}_{t-1})$ like $G_{t,1}$.

Other Baselines For all the other baselines: random search (Bergstra and Bengio, 2012), TPE (Bergstra et al., 2011b), Reinforcement Learning (Zoph and Le, 2016), BO with SMAC (Hutter et al., 2011b), regularised evolution (Real et al., 2019), we follow the implementation available at `https://github.com/automl/nas_benchmarks` for NASBench-101 (Ying et al., 2019). We modify them to be applicable on NASBench-201 (Dong and Yang, 2020b). Note that like GCNBO, all these methods are sequential $B = 1$, and thus should enjoy the same advantage mentioned above when measured against the number of architectures evaluated.

B.4 Experiment Set-up for the DARTS Space

We mostly follow the setup and the code base (`https://github.com/quark0/darts`) from DARTS (Liu et al., 2018b), and we detail the setup in full below:

Architecture Search During the architecture search, we use half of the CIFAR10 training data and leave the other half as the validation set. We use stack the search cell 8 times to produced a small network, and use batch size of 64 and initial number of channels of 16. As discussed, we only search one cell, and use this for both normal and reduction cells (in DARTS the two cells are searched separately). We use SGD optimiser with momentum of 0.9, weight decay of 3×10^{-4} and an initial learning rate of 0.025 which is cosine annealed to zero over 50 epochs. As known by previous

works (Liu et al., 2018b), the validation performance on CIFAR10 is very volatile, and to ameliorate this we feed the average of the validation accuracy of the final 5 epochs as the observed accuracy to the GPWL surrogate. We use the identical setup for GPWL surrogate as the NAS-Bench experiments and we use the standard mutation algorithm described to generate the candidates every BO iteration.

Architecture Evaluation After the search budget (set to 150 architectures) is exhausted, we evaluate the neural network stacked from the best architecture found, *based on the validation accuracy during the search stage*. During evaluation, we construct a larger network of 20 cells and is trained for 600 epochs with batch size 96 and initial number of channels of 36. Additional enhancements that are almost universally used in previous works, such as path dropout of probability 0.2, cutout, and auxiliary towers with weight 0.4 are also applied in this stage (these techniques are all identical to those used in Liu et al. (2018b)). Any other enhancements not used in DARTS such as mixup, AutoAugment and test-time data augmentation are not applied. The optimiser setting is identical to that during architecture search, with the exception that the cosine annealing is over the full 600 epochs instead of 50 during search. During this stage, we use the entire CIFAR10 training set for training, and report best accuracy encountered during evaluation on the validation set in Table 7.2 of Section 7.5.3. We finally train the final architecture for 4 random repeats on CIFAR10 dataset.

B.5 NAS Datasets Description

The NAS datasets we experiment with are:

- **NASBench-101 (NB101)** (Ying et al., 2019): this dataset contains 423,624 unique neural architectures, each of which is trained with SGD optimiser for 108 epochs and evaluated on CIFAR10 data. The training and evaluation are repeated over 3 random initialisation seeds. The training accuracy/loss, validation accuracy/loss and test accuracy/loss at 4-th, 12-th, 36-th and 108-th epochs, as well as architecture meta-information such as number of parameters

are all accessible from the dataset. The search space is an acyclic directed graph with 7 nodes and a maximum of 9 edges. Besides the `input` node and `output` node, the remaining 5 operation nodes can choose one of the three possible operations: { `conv3×3-bn-relu`, `conv1×1-bn-relu`, `maxpool3×3` }. The dataset is available at https://github.com/google-research/nasbench/.

- **NASBench-201 (NB201)** (Dong and Yang, 2020a): this dataset contains information of 15,625 different neural architectures, each of which is trained with SGD optimiser for 200 epochs and evaluated on 3 different datasets: CIFAR10, CIFAR100, ImageNet120 for 3 random initialisation seeds. The training accuracy/loss, validation accuracy/loss after every training epoch as well as architecture meta-information such as number of parameters, and FLOPs are all accessible from the dataset. The search space of the NASBench-201 dataset is an acyclic directed graph with 4 nodes and 6 edges. Each edge corresponds to an operation selected from the set of 5 possible options: { `conv1×1`, `conv3×3`, `avgpool3×3`, `skip-connect`, `zeroize` }. This search space is applicable to almost all up-to-date NAS algorithms. The dataset is available at https://github.com/D-X-Y/NAS-Bench-201.

- **NASBench-301 (NB301)** (Siems et al., 2020): this dataset contains 23000 8-cell architectures drawn from the DARTS search space and evaluated on CIFAR10. DARTS search space (Liu et al., 2019) is more general than those of NASBench-201 and contains over 10^{18} architectures. It's also the most widely adopted space in NAS (Zoph et al., 2018b; Liu et al., 2019; Chen et al., 2019; Xie et al., 2019b; Xu et al., 2019b; Real et al., 2019; Li and Talwalkar, 2020; Pham et al., 2018; Shaw et al., 2019; Zhou et al., 2020b). Particularly, this search space comprises a cell of 7 nodes: the first two nodes in cell k are the input nodes which equals to the outputs of cell $k-2$ and cell $k-1$ respectively. The last node in the cell k is the output node which gives a depthwise concatenation of all the intermediate nodes. The remaining four intermediate nodes are operation nodes take can take one out of eight operation

choices: { `sep-conv-3×3`, `sep-conv-5×5`, `dil-conv-3×3`, `dil-conv-5×5`, `max-pool-3×3`,`avg-pool-3×3`, `skip-connect`, `none` }. Each architecture in this dataset is trained for 100 epochs using a SGD optimiser with an initial learning rate of 0.025, a cosine annealing schedule and a batch size of 96. The training also adopts regularisation techniques such as an auxiliary tower with a weight of 0.4 and DropPath with probability of 0.2. For these architectures, we can assess their training accuracy/loss, validation accuracy after every training epoch from the dataset. Moreover, this dataset also provides a well trained surrogate model which can accurately predict the final test accuracy of any other possible architectures formed by 8 cells from the DARTS search space. We use this surrogate model to predict the ground-truth test accuracy of the subnetworks, when being trained from scratch independently, in the one-shot experiments in Section 8.4.5. The dataset is available at https://github.com/automl/nasbench301.

- **DARTS**: this group of datasets correspond to the DARTS architectures used for final evaluation and deployment; each architecture is formed by stacking the DARTS cells 20 times, thus much larger than the 9-cell counterparts in NASBench-301 above. Specifically, we generate three 20-cell DARTS architecture datasets; each dataset contains 150 architectures randomly sampled from the search space but follows a different training set-up. Specifically, for the dataset used in Figure 8.5 in Section 8.4.2, we use an initial learning rate of 0.025, a cosine-annealing schedule and a batch size of 96 (i.e. the setting for CIFAR10 complete training in (Liu et al., 2019). For the dataset used in Figure 8.5c in Section 8.4.2, we use an initial learning rate of 0.1, a step-decay schedule and a batch size of 128 (i.e. the setting for ImageNet complete training in (Liu et al., 2019)). Finally, for the dataset used in Figure 8.5d in Section 8.4.2, we use an initial learning rate of 0.05, a cosine-annealing schedule and a batch size of 128, modified from the setting in Figure 2 (b). The other training setups are the same: all architectures are trained for 150 epochs using SGD optimiser with momentum of 0.9 and regularised using a

Cut-Out of 16 and a DropPath with probability of 0.2 following (Liu et al., 2019). For these datasets, we also record the training loss for each mini-batch (TLmini) on top of the conventional training and validation loss/accuracies. The mini-batch training loss is used to verify our claim that it is the sum of training losses, which has nice theoretical motivation, instead of training loss itself that gives good correlation with the generalisation performance of the architectures.

- **RandWiredNN (RWNN)** : we produce this dataset by generating 552 randomly wired neural architectures from the random graph generators proposed in (Xie et al., 2019a) and evaluate their performance on the image dataset Flower102 (Nilsback and Zisserman, 2008). We explore 69 sets of hyper-parameter values for the random graph generators and for each set of hyper-parameter values, we sample 8 randomly wired neural networks from the generator. A randomly wired neural network comprises 3 cells connected in sequence and each cell is a 32-node random graph. The nodes within the graph can take one of the three possible options: {`input`, `output`, `relu-conv3×3-bn` }. The wiring/connection within the graph is generated with one of the three classic random graph models in graph theory: Erdos-Renyi (ER), Barabasi-Albert (BA) and Watt-Strogatz (WS) models. Each random graph model has 1 or 2 hyper-parameters that decide the generative distribution over edge/node connection in the graph. All the architectures are trained with SGD optimiser for 250 epochs and other training set-ups follow those in (Liu et al., 2019). This dataset allows us to evaluate the performance of our simple estimator on hyper-parameter/model selection for the random graph generator.

- **ResNet and ResNeXt** (Radosavovic et al., 2019): this group of search spaces featuring two ResNet model families: ResNet and ResNeXt. The number of architecture samples as well as the architecture parameters and their corresponding range are shown in Table B.1. Each architecture is trained

on CIFAR10 for 100 epochs using SGD with an initial learning rate of 0.1, a cosine annealing schedule and a batch size of 128. The dataset is available at https://github.com/facebookresearch/nds.

Table B.1: Search spaces for ResNet and ResNeXt. Each network is formed of three stages and for each of the stage i, there are d_i the number of blocks, w_i number of channels per block. For ResNeXt, we also need to decide on the bottleneck width ratio r_i and the number of groups g_i per stage. The total number of possible architectures is $(dw)^3$ and $(dwrg)^3$ for ResNet and ResNeXt.

	d_i	w_i	r_i	g_i	$N_{samples}$	N_{total}
ResNet	[1,24]	[16,256]			25000	1259712
ResNeXt-A	[1,16]	[16,256]	[1,4]	[1,4]	25000	11,390,625
ResNeXt-B	[1,16]	[64,1024]	[1,4]	]1,16]	25000	52,734,375

C

Additional Derivatives

C.1 Compute the derivatives of the FITBO Acquisition Functions

As explained in Section 3.2.2, the final acquisition function of our FITBO approach has the following form:

$$\begin{aligned}\alpha(\mathbf{x}|\mathcal{D}_n) =& H\Big[\frac{1}{M}\sum_{j}^{M} p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)})\Big] \\ & - \frac{1}{2M}\sum_{j}^{M} \log\Big[2\pi e\Big(\sigma_f(\mathbf{x}|D, \boldsymbol{\theta}^{(j)}, \eta^{(j)})^2 + \sigma_n^2\Big)\Big] \end{aligned} \tag{C.1}$$

where $\sigma_f(\mathbf{x}|D, \boldsymbol{\theta}^{(j)}, \eta^{(j)})^2 = K_f^{(j)}(\mathbf{x}, \mathbf{x}') = m_g^{(j)}(\mathbf{x}) K_g^{(j)}(\mathbf{x}, \mathbf{x}') m_g^{(j)}(\mathbf{x}')$.

If the kernel function adopted is differentiable, we can compute the derivative of $\alpha(\mathbf{x}|\mathcal{D}_n)$ with respect to x_d, the d^{th} dimension of $\mathbf{x}$, to facilitate the optimisation of the acquisition function.

To simplify our notation, let $G(y|\mathbf{x}) = \frac{1}{M}\sum_j^M p(y|\mathbf{x}, \boldsymbol{\psi}^{(j)}) = \frac{1}{M}\sum_j^M p(y|\mathcal{D}_n, \mathbf{x}, \boldsymbol{\theta}^{(j)}, \eta^{(j)})$ and $v_j(\mathbf{x}) = \sigma_f(\mathbf{x}|D, \boldsymbol{\theta}^{(j)}, \eta^{(j)})^2 + \sigma^2_{\text{noise}}$. The acquisition function of FITBO then becomes:

$$\alpha(\mathbf{x}|\mathcal{D}_n) = H\Big[G(y|\mathbf{x})\Big] - \frac{1}{2M}\sum_{j}^{M} \log\Big[2\pi e\Big(v_j(\mathbf{x})\Big)\Big].$$

C.1.1 Derivative of FITBO

If we approximate the entropy of the Gaussian mixture using numerical method, the derivative of $\alpha(\mathbf{x}|\mathcal{D}_n)$ can be computed as follows:

$$\begin{aligned}\frac{\partial\alpha(\mathbf{x}|\mathcal{D}_n)}{\partial x_d} &= \frac{\partial\Big[-\int G(y|\mathbf{x})\log G(y|\mathbf{x})\mathrm{d}y\Big]}{\partial x_d} \\ &\quad - \frac{1}{2M}\sum_j^M \frac{\partial\log\Big[2\pi e\Big(v_j(\mathbf{x})\Big)\Big]}{\partial x_d} \\ &= \Big[-\int \frac{\partial G(y|\mathbf{x})}{\partial x_d} + \log G(y|\mathbf{x})\frac{\partial G(y|\mathbf{x})}{\partial x_d}\mathrm{d}y\Big] \\ &\quad - \frac{1}{2M}\sum_j^M \frac{1}{v_j(\mathbf{x})}\frac{\partial v_j(\mathbf{x})}{\partial x_d}\end{aligned} \tag{C.2}$$

The key partial derivative needs to solve is $\frac{\partial G(y|\mathbf{x})}{\partial x_d}$ and for a 1D Gaussian mixture:

$$\begin{aligned}\frac{\partial G(y|\mathbf{x})}{\partial x_d} &= \frac{1}{M}\sum_j^M \frac{\partial p(y|\mathbf{x},\boldsymbol{\psi}^{(j)})}{\partial x_d} \\ &= \frac{1}{M}\sum_j^M \frac{\partial\Big(\frac{1}{\sqrt{2\pi e v_j(\mathbf{x})}}\exp(-\frac{(y-\mu_j(\mathbf{x}))^2}{2v_j(\mathbf{x})})\Big)}{\partial x_d} \\ &= \frac{1}{M}\sum_j^M \Big[-p(y|\mathbf{x},\boldsymbol{\psi}^{(j)})\frac{1}{2v_j(\mathbf{x})}\frac{\partial v_j(\mathbf{x})}{\partial x_d} \\ &\quad + p(y|\mathbf{x},\boldsymbol{\psi}^{(j)})\frac{(y-\mu_j(\mathbf{x}))^2}{2(v_j(\mathbf{x}))^2}\frac{\partial v_j(\mathbf{x})}{\partial x_d} \\ &\quad + p(y|\mathbf{x},\boldsymbol{\psi}^{(j)})\frac{(y-\mu_j(\mathbf{x}))}{v_j(\mathbf{x})}\frac{\partial\mu_j(\mathbf{x})}{\partial x_d}\Big] \\ &= \frac{1}{M}\sum_j^M p(y|\mathbf{x},\boldsymbol{\psi}^{(j)})\Bigg[\frac{(y-\mu_j(\mathbf{x}))}{v_j(\mathbf{x})}\frac{\partial\mu_j(\mathbf{x})}{\partial x_d} \\ &\quad + \Big(\frac{(y-\mu_j(\mathbf{x}))^2 - v_j(\mathbf{x})}{2(v_j(\mathbf{x}))^2}\Big)\frac{\partial v_j(\mathbf{x})}{\partial x_d}\Bigg]\end{aligned} \tag{C.3}$$

Thus, $\int \frac{\partial G(y|\mathbf{x})}{\partial x_d} \mathrm{d}y = 0$ and the first derivative term in function C.2 can be expanded in the following way:

$$
\begin{aligned}
&- \int \frac{\partial G(y|\mathbf{x})}{\partial x_d} + \log G(y|\mathbf{x}) \frac{\partial G(y|\mathbf{x})}{\partial x_d} \mathrm{d}y \\
&= - \int \log G(y|\mathbf{x}) \frac{\partial G(y|\mathbf{x})}{\partial x_d} \mathrm{d}y \\
&= - \int \log G(y|\mathbf{x}) \frac{1}{M} \sum_j^M \Bigg\{ \\
&\quad p(y|\mathbf{x}, \boldsymbol{\psi}^{(j)}) \Bigg[\frac{(y - \mu_j(\mathbf{x}))}{v_j(\mathbf{x})} \frac{\partial \mu_j(\mathbf{x})}{\partial x_d} \\
&\quad + \Big(\frac{(y - \mu_j(\mathbf{x}))^2 - v_j(\mathbf{x})}{2(v_j(\mathbf{x}))^2} \Big) \frac{\partial v_j(\mathbf{x})}{\partial x_d} \Bigg] \Bigg\} \mathrm{d}y
\end{aligned}
\tag{C.4}
$$

The derivative of the acquisition function then has the form:

$$
\begin{aligned}
&\frac{\partial \alpha(\mathbf{x}|\mathcal{D}_n)}{\partial x_d} \\
&= - \int \log G(y|\mathbf{x}) \frac{1}{M} \sum_j^M \Bigg\{ p(y|\mathbf{x}, \boldsymbol{\psi}^{(j)}) \Bigg[\\
&\quad \Big(\frac{(y - \mu_j(\mathbf{x}))^2 - v_j(\mathbf{x})}{2(v_j(\mathbf{x}))^2} \Big) \frac{\partial v_j(\mathbf{x})}{\partial x_d} \\
&\quad + \frac{(y - \mu_j(\mathbf{x}))}{v_j(\mathbf{x})} \frac{\partial \mu_j(\mathbf{x})}{\partial x_d} \Bigg] \Bigg\} \mathrm{d}y \\
&\quad - \frac{1}{2M} \sum_j^M \frac{1}{\Big(v_j(\mathbf{x}) + \sigma_n^2 \Big)} \frac{\partial v_j(\mathbf{x})}{\partial x_d}
\end{aligned}
\tag{C.5}
$$

where

$$
\frac{\partial \mu_j(\mathbf{x})}{\partial x_d} = m_g^{(j)}(\mathbf{x}) \frac{\partial m_g^{(j)}(\mathbf{x})}{\partial x_d}, \tag{C.6}
$$

$$
\begin{aligned}
\frac{\partial v_j(\mathbf{x})}{\partial x_d} = &\frac{\partial K_g^{(j)}(\mathbf{x}, \mathbf{x}')}{\partial x_d} \Big(m_g^{(j)}(\mathbf{x}) \Big)^2 \\
&+ 2 K_g^{(j)}(\mathbf{x}, \mathbf{x}') m_g^{(j)}(\mathbf{x}) \frac{\partial m_g^{(j)}(\mathbf{x})}{\partial x_d}
\end{aligned}
\tag{C.7}
$$

and $\frac{\partial m_g(\mathbf{x})}{\partial x_d}$ and $\frac{\partial K_g(\mathbf{x}, \mathbf{x}')}{\partial x_d}$ can be computed from the definition of the chosen kernel function.

C.1.2 Derivative of FITBO-MM acquisition function

A faster alternative to approximate the entropy of the Gaussian mixture is to use simple moment-matching:

$$G(y|\mathbf{x}) = \frac{1}{M}\sum_{j}^{M} p(y|\mathbf{x}, \boldsymbol{\psi}^{(j)}) \approx \mathcal{N}(y|m_G(\mathbf{x}), v_G(\mathbf{x}))$$

where

$$m_G(\mathbf{x}) = \sum_{j}^{M} \frac{1}{M}\mu_j(\mathbf{x})$$

$$v_G(\mathbf{x}) = \sum_{j}^{M} \frac{1}{M}\Big(v_j(\mathbf{x}) + (\mu_j(\mathbf{x}))^2\Big) - \Big(m_G(\mathbf{x})\Big)^2.$$

This leads to an analytical form of the acquisition function:

$$\alpha(\mathbf{x}|\mathcal{D}_n) \approx \frac{1}{2}\log\Big[2\pi e\Big(v_G(\mathbf{x})\Big)\Big] - \frac{1}{2M}\sum_{j}^{M}\log\Big[2\pi e\Big(v_j(\mathbf{x})\Big)\Big].$$

The derivative of the acquisition function then has a neat analytical form:

$$\frac{\partial\alpha(\mathbf{x}|\mathcal{D}_n)}{\partial x_d} = \frac{1}{2v_G(\mathbf{x})}\frac{\partial v_G(\mathbf{x})}{\partial x_d} - \frac{1}{2M}\sum_{j}^{M}\frac{1}{v_j(\mathbf{x})}\frac{\partial v_j(\mathbf{x})}{\partial x_d}$$

where

$$\begin{aligned}\frac{\partial v_G(\mathbf{x})}{\partial x_d} = &\sum_{j}^{M}\frac{1}{M}\Big(\frac{\partial v_j(\mathbf{x})}{\partial x_d} + 2\mu_j(\mathbf{x})\frac{\partial\mu_j(\mathbf{x})}{\partial x_d}\Big)\\ &- 2m_G(\mathbf{x})\Big(\sum_{j}^{M}\frac{1}{M}\frac{\partial\mu_j(\mathbf{x})}{\partial x_d}\Big)\end{aligned}$$

with $\frac{\partial\mu_j(\mathbf{x})}{\partial x_d}$ and $\frac{\partial v_j(\mathbf{x})}{\partial x_d}$ have the same expressions as Equations (C.6) and (C.7).

C.2 Learn the Hyper-parameters of the CoCaBO Kernel

We present the derivative for estimating the variable λ in our CoCaBO kernel.

$$\begin{aligned}k_z(\mathbf{z}, \mathbf{z}') =&(1-\lambda)\left(k_h(\mathbf{h}, \mathbf{h}') + k_x(\mathbf{x}, \mathbf{x}')\right)\\ &+ \lambda k_h(\mathbf{h}, \mathbf{h}')k_x(\mathbf{x}, \mathbf{x}').\end{aligned} \tag{C.8}$$

The hyperparameters of the kernel are optimised by maximising the log marginal likelihood (LML) of the GP surrogate

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}), \tag{C.9}$$

where we collected the the hyperparameters of both kernels as well as the CoCaBO hyperparameter into $\boldsymbol{\theta} = \{\theta_h, \theta_x, \lambda\}$. The LML and its derivative are defined as in (Rasmussen and Williams, 2006a)

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\intercal \mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}| + \text{constant} \tag{C.10}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{2}\left(\mathbf{y}^\intercal \mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\mathbf{K}^{-1}\mathbf{y} - \operatorname{tr}\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\right)\right), \tag{C.11}$$

where $\mathbf{y}$ are the function values at sample locations and $\mathbf{K}$ is the kernel matrix of $k_z(\mathbf{z}, \mathbf{z}')$ evaluated on the training data.

Optimisation of the LML was performed via multi-started gradient descent. The gradient in Equation (C.11) relies on the gradient of the kernel k_z w.r.t. each of its parameters:

$$\frac{\partial k_z}{\partial \theta_h} = (1-\lambda)\frac{\partial k_h}{\partial \theta_h} + \lambda k_x \frac{\partial k_h}{\partial \theta_h} \tag{C.12}$$

$$\frac{\partial k_z}{\partial \theta_x} = (1-\lambda)\frac{\partial k_x}{\partial \theta_x} + \lambda \frac{\partial k_x}{\partial \theta_x} k_h \tag{C.13}$$

$$\frac{\partial k_z}{\partial \lambda} = -(k_h + k_x) + k_h k_x, \tag{C.14}$$

where we used the shorthand $k_z = k_z(\mathbf{z}, \mathbf{z}')$, $k_h = k_h(\mathbf{h}, \mathbf{h}')$ and $k_x = k_x(\mathbf{x}, \mathbf{x}')$.

C.3 Theoretical Analysis for CASMOPOLITAN

In this section, we first proof the categorical kernel in Equation (5.11) and mixed kernel in Equation (5.10) used for CASMOPOLITAN are positive semi-definite. We then provide upper bounds on the maximum information gains of our proposed categorical kernel and mixed kernel (Theorem 1). We then prove that after a restart, under Assumptions 1 and 2, CASMOPOLITAN converges to a local maxima after a finite number of iterations or converges to the global maximum (Theorem 2). Finally, we prove that with our UCB-restart strategy, under Assumptions 1, 2 and

some assumptions described in (Srinivas et al., 2009), CASMOPOLITAN converges to the global maximum with a sublinear rate over the number of restarts in both categorical (Theorem 3) and mixed space settings (Theorem 4).

C.3.1 Lemma on Validity of Proposed Kernels

Lemma 1 *The proposed categorical kernel in Equation* (5.11) *and mixed kernel in Equation* (5.10) *are positive semi-definite and thus valid kernels.*

Proof of Lemma 1. For the categorical kernel in Equation (5.11), we have that exponential of a kernel is also a kernel, and since the categorical overlap kernel in Equation (5.7) is a valid kernel, its exponentiated version is also a valid kernel. For the mixed kernel in Equation (5.10), since addition and multiplication between kernels result in valid kernels, and since both $k_x(.,.)$ and $k_h(.,.)$ are valid kernels, therefore, the mixed kernel in Equation (5.10) is also a valid kernel. □

C.3.2 Theorem on Maximum Information Gains of Proposed Kernels

Theorem 1 *Let us define* $\gamma(T;k;V) := \max_{A\subseteq V,|A|\leq T} \frac{1}{2}\log|I+\sigma^{-2}[k(\mathbf{v},\mathbf{v}')]_{\mathbf{v},\mathbf{v}'\in A}|$ *as the maximum information gain achieved by sampling* T *points in a GP defined over a set* V *with a kernel* k*. Let us define* $\tilde{N} := \prod_{j=1}^{d_h} n_j$*, then we have,*

1. *For the categorical kernel* k_h*,* $\gamma(T;k_h;\mathcal{H}) = \mathcal{O}(\tilde{N}\log T)$*;*
2. *For the mixed kernel* k*,* $\gamma(T;k;[\mathcal{H},\mathcal{X}]) \leq \mathcal{O}\Big((\lambda\tilde{N}+1-\lambda)\gamma(T;k_x;\mathcal{X}) + (\tilde{N}+2-2\lambda)\log T\Big)$*.*

Proof of Theorem 1. We derive the maximum information gain of the categorical kernel k_h first and then the mixed kernel k.

Maximum Information Gain of the Categorical Kernel

We derive the maximum information gain of the categorical kernel k_h proposed in Equation 5.11) by bounding $\gamma(T;k_h;\mathcal{H})$ directly. Let us first consider the case when the objective function f has only one categorical variable h with n distinct values

(i.e. $h \in \{A_1, A_2, \ldots, A_n\}$ where A_i is a categorical value and $A_i \neq A_j$ when $i \neq j$). Let us consider T data points $h_1, h_2, \ldots, h_T$, then its corresponding covariance matrix K_T is $[k_h(h_i, h_j)]_{i,j=1}^T$. As the maximum information gain $\gamma(T; k_h; \mathcal{H})$ is equal to $\log|I_T + \sigma^{-2}K_T|$ where I_T is the identity matrix of size T,[1] thus, we will bound $\gamma(T; k_h; \mathcal{H})$ by bounding $\log|I_T + \sigma^{-2}K_T|$. Our general idea is to perform a decomposition of K_T, i.e. expressing $K_T = \Phi E \Psi^T$ where $\Phi \in \mathbb{R}^{T\times n}$, $\Psi \in \mathbb{R}^{T\times n}$, and $E \in \mathbb{R}^{n\times n}$, and then apply the Sylvester's determinant theory and the Hadamard's inequality to derive an upper bound for $\log|I_T + \sigma^{-2}K_T|$.

In the sequel, for ease of notation, we define the function q as a mapping from A_i to i. In particular, $q(A_i) = i,\ \forall i = 1, \ldots, n$. With the categorical kernel $k_h(h, h') = \exp(l\delta(h, h'))$, in the following, we will prove that K_T can be decomposed as,[2]

$$K_T = \Phi E \Psi^T, \tag{C.15}$$

where $\Phi, \Psi \in \mathbb{R}^{T\times n}, E \in \mathbb{R}^{n\times n}$, and

$$\Phi = \begin{bmatrix} \phi(h_1) \\ \phi(h_2) \\ \cdots \\ \phi(h_T) \end{bmatrix}, \Psi = \begin{bmatrix} \psi(h_1) \\ \psi(h_2) \\ \cdots \\ \psi(h_T) \end{bmatrix},$$
$$E = \mathrm{diag}(\exp(l) + n - 1, \exp(l) - 1, ..., \exp(l) - 1),$$

with $\phi(h_i)$ being an n-dimensional row vector with 1 at the 1st column, (-1) at the $q(h_i)$-th column, and 1 at the $(q(h_i) + 1)$-th column, i.e.,

$$\phi(h_i) = \begin{cases} [1\ 1\ 0\ ...\ 0\ 0], & \text{if } q(h_i) = 1 \\ [1\ 0\ 0\ ...\ 0\ (-1)\ 1\ 0 \ldots 0], & \text{if } 1 < q(h_i) < n \\ [1\ 0\ 0\ ...\ 0\ (-1)], & \text{if } q(h_i) = n \end{cases},$$

[1] $|S|$ denotes the determinant of matrix S.

[2] When $T = n$, this decomposition is equivalent to the eigendecomposition. That is, the diagonal of matrix E consists of the eigenvalues of K_T and each column of Φ is an eigenvector of K_T.

and $\psi(h_i)$ being an n-dimensional row vector with the following formula,

$$\psi(h_i) = \begin{cases} \frac{1}{n}[1\ (n-1)\ (n-2)\ ...\ 1], & \text{if } q(h_i) = 1 \\ \frac{1}{n}[1\ (-1)\ ...\ -(q(h_i)-1)\ (n-q(h_i))\ ...\ 1], & \\ & \text{if } 1 < q(h_i) < n \\ \frac{1}{n}[1\ (-1)\ ...\ -(n-1)], & \text{if } q(h_i) = n. \end{cases}$$

To prove the decomposition in Equation (C.15), we compute the element at the i-th row and j-th column of $\Phi E \Psi^T$, i.e. $[\Phi E \Psi^T]_{ij}$, and then prove that $[\Phi E \Psi^T]_{ij}$ is equal to $[K_T]_{ij}$. To compute $[\Phi E \Psi^T]_{ij}$, it can be directly seen that,

$$[\Phi E \Psi^T]_{ij} = \sum_{r=1}^{n} \phi_r(h_i) E_r \psi_r(h_j),$$

where $\phi_r(h_i)$ denotes the r-th element of $\phi(h_i)$, $\psi_r(h_j)$ denotes the r-th element of $\psi(h_j)$ and E_r denotes the r-th element on the diagonal of matrix E. We then consider the following three cases:

Case 1: $q(h_j) = q(h_i)$. First, let us consider $1 < q(h_i) < n$, then we have,

$$\begin{aligned} [\Phi E \Psi^T]_{ij} =& 1 \times (\exp(l) + n - 1) \times \frac{1}{n} \\ &+ (-1) \times (\exp(l) - 1) \times \frac{(-q(h_i)+1)}{n} \\ &+ 1 \times (\exp(l) - 1) \times \frac{(n - q(h_i))}{n} \\ =& \exp(l). \end{aligned}$$

Note that when $q(h_j) = q(h_i)$, we will have $h_j = h_i$, thus, the element $[K_T]_{ij}$ is equal to $\exp(l)$. Similar arguments can be made when $q(h_i) = 1$ or $q(h_i) = n$, that is, $[K_T]_{ij}$ is equal to $\exp(l)$. Therefore, $[\Phi E \Psi^T]_{ij} = [K_T]_{ij} = \exp(l)$.

Case 2: $q(h_j) \geq q(h_i) + 1$. Let us first consider $1 < q(h_i)$, then,

$$\begin{aligned} [\Phi E \Psi^T]_{ij} =& 1 \times (\exp(l) + n - 1) \times \frac{1}{n} \\ &+ (-1) \times (\exp(l) - 1) \times \frac{(-q(h_i)+1)}{n} \\ &+ 1 \times (\exp(l) - 1) \times \frac{(-q(h_i))}{n} \\ =& 1. \end{aligned}$$

In this case, with $q(h_j) \geq q(h_i) + 1$, we will have $h_i \neq h_j$, hence, the element $[K_T]_{ij}$ is equal to 1. Similar arguments can be made when $q(h_i) = 1$, that is, in this case, $[K_T]_{ij}$ is also equal to 1. Therefore, $[\Phi E \Psi^T]_{ij} = [K_T]_{ij} = 1$.

Case 3: $q(h_j) \leq q(h_i) - 1$. Let us first consider $q(h_i) < n$, then,

$$\begin{aligned}[\Phi E \Psi^T]_{ij} =& 1 \times (\exp(l) + n - 1) \times \frac{1}{n} \\ &+ (-1) \times (\exp(l) - 1) \times \frac{(n - q(h_i))}{n} \\ &+ 1 \times (\exp(l) - 1) \times \frac{(n - q(h_i) - 1)}{n} \\ =& 1.\end{aligned}$$

Similar to *Case* 2, we also have $h_i \neq h_j$. Similar arguments can be made when $q(h_i) = n$, $[K_T]_{ij}$ is also equal to 1. Hence, $[\Phi E \Psi^T]_{ij} = [K_T]_{ij} = 1$.

Combining *Cases* 1, 2, 3, we proved the decomposition in Equation (C.15). Now using this decomposition, we have,

$$\gamma(T; k_h; \mathcal{H}) = \log |I_T + \sigma^{-2} K_T| = \log |I_T + \sigma^{-2} \Phi E \Psi^T|.$$

By Sylvester's determinant theorem (Sylvester, 1851),

$$\gamma(T; k_h; \mathcal{H}) = \log |I_n + \sigma^{-2} \Psi^T \Phi E|. \tag{C.16}$$

Next, we prove the matrix $\Psi^T \Phi E$ is a positive semi-definite (p.s.d.) matrix, and the maximum element on the diagonal of $\Psi^T \Phi E$ is equal or less than $T(\exp(l)+n-1)$. Let us denote m_i as the number of times the categorical value A_i appears in T data points. It can be directly seen that,

$$[\Psi^T \Phi]_{ij} = \sum_{r=1}^{T} \psi_i(h_r) \phi_j(h_r) = \sum_{r=1}^{n} m_r \psi_i(A_r) \phi_j(A_r).$$

Hence, the matrix $\Psi^T \Phi$ can be written as,

$$\Psi^T \Phi = \Psi_A^T F \Phi_A, \qquad \Phi_A, \Psi_A, E \in \mathbb{R}^{n \times n}, \tag{C.17}$$

where

$$\Phi_A = \begin{bmatrix} \phi(A_1) \\ \phi(A_2) \\ \dots \\ \phi(A_n) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & ... & 0 & 0 \\ 1 & -1 & 1 & 0 & \dots & 0 & 0 \\ 1 & 0 & -1 & 1 & \dots & 0 & 0 \\ & & & & \dots & & \\ 1 & 0 & 0 & 0 & \dots & -1 & 1 \\ 1 & 0 & 0 & 0 & \dots & 0 & -1 \end{bmatrix},$$

$$\Psi_A = \begin{bmatrix} \psi(A_1) \\ \psi(A_2) \\ \dots \\ \psi(A_n) \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & n-1 & n-2 & ... & 2 & 1 \\ 1 & -1 & n-2 & ... & 2 & 1 \\ 1 & -1 & -2 & ... & 2 & 1 \\ & & & & ... & \\ 1 & -1 & -2 & ... & -(n-2) & 1 \\ 1 & -1 & -2 & ... & -(n-2) & -(n-1) \end{bmatrix},$$

$$F = \text{diag}(m_1, m_2, \dots, m_n).$$

It is straightforward that $\Psi_A^T \Phi_A = I_n$, thus, from Equation (C.17), we can see that $\Psi_A^T F \Phi_A$ is an eigendecomposition of $\Psi^T \Phi$, and hence, the eigenvalues of $\Psi^T \Phi$ are $m_1, m_2, \dots, m_n$. As $m_i \geq 0, \forall i = 1, ..., n$, so $\Psi^T \Phi$ is a p.s.d. matrix, and therefore, $\Psi^T \Phi E$ is also a p.s.d. matrix. Besides, note that the r-th element on the diagonal of $\Psi^T \Phi E$ can be computed as $[\Psi^T \Phi]_{rr} E_r$ where $[\Psi^T \Phi]_{rr}$ and E_r are the r-th elements on the diagonal of $\Psi^T \Phi$ and E, respectively. Since $[\Psi^T \Phi]_{rr} \leq \sum_{i=1}^{T} (n-1)/n \times 1 \leq T$, and $E_r \leq (\exp(l)+n-1)$, hence, $[\Psi^T \Phi]_{rr} E_r \leq T(\exp(l)+n-1)$. This results that the maximum element on the diagonal of $\Psi^T \Phi E$ is equal or smaller than $T(\exp(l)+n-1)$.

Combining Equation (C.16) and the Hadamard's inequality (Mazya and Shaposhnikova, 1999) on the positive semi-definite matrix $\Psi^T \Phi E$, we have,

$$\gamma(T; k_h; \mathcal{H}) \leq \log |I_n + \sigma^{-2} W|,$$

where $W = \text{diag}(\text{diag}^{-1}(\Psi^T \Phi E))$. Since the maximum element on the diagonal of $\Psi^T \Phi E$ is equal or smaller than $T(\exp(l) + n - 1)$. Therefore, $\gamma(T; k_h; \mathcal{H}) = \mathcal{O}(n \log(1 + \sigma^{-2} T(\exp(l) + n - 1))) = \mathcal{O}(n \log T)$. □

Now let consider the case when the objective function f has d_h categorical variables where each variable has n_j distinct values. This can be considered to be

equivalent to the case when f has one variable with $\prod_{j=1}^{d_h} n_j$ distinct values. Thus, the same proof can be used, and we have $\gamma(T; k_h; \mathcal{H}) = \mathcal{O}\Big((\prod_{j=1}^{d_h} n_j) \log T\Big)$. □

Maximum information gain of the mixed kernel

We make use of Theorems 2 and 3 in (Krause and Ong, 2011) to bound the maximum information gain of the mixed kernel k. In particular, Theorem 2 states that given two kernels: k_h on $\mathcal{H}$ and k_x on $\mathcal{X}$, and if k_h is a kernel on $\mathcal{H}$ with rank at most m, then $\gamma(T; k_h k_x; [\mathcal{H}, \mathcal{X}]) \leq m\gamma(T; k_x; \mathcal{X}) + m \log T$. On the other hand, Theorem 3 states that for any two kernels k_h on $\mathcal{H}$ and k_x on $\mathcal{X}$, then $\gamma(T; k_h + k_x; [\mathcal{H}, \mathcal{X}]) \leq \gamma(T; k_x; \mathcal{X}) + \gamma(T; k_h; \mathcal{X}) + 2 \log T$.

As proven above for the categorical kernel, the kernel k_h has at most rank $\tilde{N} = \prod_{j=1}^{d_h} n_j$ (based on the eigen-decomposition). Thus, using Theorem 2 in Krause and Ong (2011), we have

$$\gamma(T; k_h k_x; [\mathcal{H}, \mathcal{X}]) \leq \tilde{N}\gamma(T; k_x; X) + \tilde{N} \log T. \tag{C.18}$$

Similarly, using Theorem 3 in (Krause and Ong, 2011), we obtain

$$\gamma(T; k_h + k_x; [\mathcal{H}, \mathcal{X}]) \leq \mathcal{O}\Big(\gamma(T; k_x; X) + (\tilde{N} + 2) \log T\Big). \tag{C.19}$$

We have the mixed kernel k defined as $\lambda(k_x k_h) + (1-\lambda)(k_h + k_x)$ where $\lambda \in [0, 1]$ is a trade-off parameter. By combining Equations (C.18) and (C.19), we have,

$$\begin{aligned}
\gamma(T; k; [\mathcal{H}, \mathcal{X}] &\leq \lambda\mathcal{O}\Big(\tilde{N}\gamma(T; k_x; X) + \tilde{N} \log T\Big) \\
&\quad + (1-\lambda)(\gamma(T; k_x; X) + (\tilde{N} + 2) \log T) \\
&\leq \mathcal{O}\Big((\tilde{N}\lambda + 1 - \lambda)\gamma(T; k_x; X) \\
&\quad + (\tilde{N} + 2 - 2\lambda) \log T\Big). \quad \square
\end{aligned}$$

C.3.3 Theorem on Local Convergence

Assumption 1 *The objective function $f(\mathbf{z})$ is bounded in $[\mathcal{H}, \mathcal{X}]$, i.e. $\exists F_l, F_u \in \mathbb{R} : \forall \mathbf{z} \in [\mathcal{H}, \mathcal{X}]$, $F_l \leq f(\mathbf{z}) \leq F_u$.*

Assumption 2 *Let us denote $L^h_{\min}$, $L^x_{\min}$ and L^h_0, L^x_0 be the minimum and initial TR lengths for the categorical and continuous variables, respectively. Let us also denote α_s as the shrinking rate of the TRs. In the categorical setting, for any TR with length $\leq \lceil (L^h_{\min}+1)/\alpha_s \rceil - 1$,[3] the corresponding local GP approximates f accurately. That is, the GP posterior mean approximates f accurately whilst the GP posterior variance is negligible within this TR. In the mixed space setting, the local GP approximates f accurately within any TR with length $L^x \leq \max\left(L^x_{\min}/\alpha_s, L^x_0(\lceil (L^h_{\min}+1)/\alpha_s \rceil - 1)/L^h_0\right)$ and $L^h \leq \max\left(\lceil (L^h_{\min}+1)/\alpha_s \rceil - 1, \lceil L^h_0 L^x_{\min}/(\alpha_s L^x_0) \rceil\right)$.*

Theorem 2 *Given Assumptions 1 and 2, after a restart, CASMOPOLITAN converges to a local maxima after a finite number of iterations or converges to the global maximum.*

Proof of Theorem 2. We prove that under Assumptions 1 and 2, after a restart, (1) if CASMOPOLITAN terminates after a finite number of iterations, then it converges to a local maxima of f, or, (2) if CASMOPOLITAN does not terminate after a finite number of iterations, then it converges to the global maximum of f. We prove this property by contradiction.

First, let us assume after a restart, case (2) occurs, i.e. CASMOPOLITAN does not terminate after a finite number of iterations. This means when the iteration t goes to infinity, the TR length L^h is not shrunk below $L^h_{\min}$ in the categorical setting, or, both L^h and L^x are not shrunk below $L^h_{\min}$ and $L^x_{\min}$ in the mixed space setting. From the algorithm description, the TR is shrunk after `fail_tol` consecutive failures. Thus, if after $N_{\min} = \texttt{fail\_tol} \times m$ iterations where $m = \lceil \log_{\alpha_e}(L^h_0/L^h_{\min}) \rceil$[4] in the categorical setting and $m = \max(\lceil \log_{\alpha_e}(L^h_0/L^h_{\min}) \rceil, \lceil \log_{\alpha_e}(L^x_0/L^x_{\min}) \rceil)$ in the mixed space setting, there is no success, CASMOPOLITAN terminates. This means, in order for case (2) to occur, CASMOPOLITAN needs to have at least one improvement per $N_{\min}$ iterations. Let consider the series $\{f(\mathbf{z}^k)\}_{k=1}^{\infty}$ where $f(\mathbf{z}^k) = \max_{i=(k-1)N_{\min}+1,\ldots,kN_{\min}}\{f(\mathbf{z}_i)\}$ and $f(\mathbf{z}_i)$ is the function value at iteration

[3] The operator $\lceil . \rceil$ denotes the ceiling function.
[4] The operator $\lceil . \rceil$ denotes the ceiling function

i. This series is strictly increasing and the objective function $f(\mathbf{z})$ is bounded (Assumption 1). Thus, using the monotone convergence theorem (Bibby, 1974), this series converges to the global maximum of the objective function f.

Second, let consider case (1) occurs, i.e. CASMOPOLITAN terminates after a finite number of iterations. We will prove that in this case, CASMOPOLITAN converges to a local maxima of $f(\mathbf{z})$ given Assumption 2. For simplicity, let us consider the categorical setting first. Let us denote L_s as the largest TR length that after being shrunk, the algorithm terminates. By the definition of L_s, we have $\lfloor \alpha_s L_s \rfloor \leq L^h_{\min}$.[5] Due to $\lfloor \alpha_s L_s \rfloor \leq \alpha_s L_s < \lfloor \alpha_s L_s \rfloor + 1$, we have $L_s < (L^h_{\min} + 1)/\alpha_s$. And because L_s is an integer, we finally have $L_s \leq \lceil (L^h_{\min} + 1)/\alpha_s \rceil - 1$. By choosing $L_s = \lceil (L^h_{\min} + 1)/\alpha_s \rceil - 1$, we have that $\forall L > L_s, \alpha_s L \geq \alpha_s \lceil (L^h_{\min} + 1)/\alpha_s \rceil > L^h_{\min}$. This says that for all TR with length $L > L_s$, after being shrunk one time, the algorithm doesn't terminate yet. Therefore, $L_s = \lceil (L^h_{\min} + 1)/\alpha_s \rceil - 1$ is the largest TR length that after being shrunk, the algorithm terminates. This tells us when the TR length first becomes smaller or equal than L_s, CASMOPOLITAN does not terminate yet (*Conclusion* 1). In addition, since GP can fit f accurately within a TR with length L_s (Assumption 2), for any TR with length $L \leq L_s$, the solution of BO is a success (*Conclusion* 2). Combining *Conclusions* 1 and 2, we have that when TR length first becomes smaller or equal than L_s, if the current TR center is not a local maxima, CASMOPOLITAN can find a new data point whose function value larger than the function value of current TR center. Thus, in the next iteration, the TR still keeps the same length whilst having center as the new found data point. This process occurs iteratively until a local maxima is reached (i.e. when CASMOPOLITAN fails to improve from the current center), and CASMOPOLITAN terminates.

Similar arguments can be made for the mixed space setting. Let us remind that for the mixed space setting, CASMOPOLITAN terminates when either the continuous TR length $\leq L^x_{\min}$ or the categorical TR length $\leq L^h_{\min}$. Now let us consider two cases. Case (i): when the continuous TR reaches $L^x_{\min}/\alpha_s$, the corresponding length of the categorical TR is $\lceil L^h_0 L^x_{\min}/(\alpha_s L^x_0) \rceil$. Case (ii): when

[5] The operator $\lfloor . \rfloor$ denotes the floor function

the categorical TR length reaches $\lceil (L^h_{\min}+1)/\alpha_s \rceil - 1$, the corresponding length of the continuous TR is $L^x_0(\lceil (L^h_{\min}+1)/\alpha_s \rceil - 1)/L^h_0$. Based on Assumption 2, GP can fit accurately a TR with continuous length $L^x \leq \max\left(L^x_{\min}/\alpha_s, L^x_0(\lceil (L^h_{\min}+1)/\alpha_s \rceil - 1)/L^h_0\right)$ and $L^h \leq \max\left(\lceil (L^h_{\min}+1)/\alpha_s \rceil - 1, \lceil L^h_0 L^x_{\min}/(\alpha_s L^x_0) \rceil\right)$, then when Case (i) or Case (ii) occurs, the GP approximates accurately the objective function f within the corresponding TR, and thus similar argument as in the categorical setting can be made. That is, if the current TR center is not a local maxima, then CASMOPOLITAN can find a new data point whose function value larger than the function value of current TR center. And this process occurs iteratively until a local maxima is reached, and CASMOPOLITAN terminates. □

C.3.4 Theorem on Global Convergence of Categorical Setting

Theorem 3 *Let us consider the categorical setting, $f : \mathcal{H} \to \mathbb{R}$. Let $\zeta \in (0,1)$ and $\beta_i = 2\log(|\mathcal{H}| i^2 \pi^2 / 6\zeta)$ at the i-th restart. Suppose the objective function f satisfies that: there exists a class of functions which pass through all the local maxima of f,[6] share the same global maximum with f, and is sampled from the auxiliary global GP $GP(0, k_h)$. Then given Assumptions 1 and 2, CASMOPOLITAN obtains a regret bound of $\mathcal{O}^*\left(\sqrt{I\gamma(I; k_h, \mathcal{H}) \log |\mathcal{H}|}\right)$ w.h.p. Formally,*

$$Pr\left\{ R_I \leq \sqrt{C_1 I \beta_I \gamma(I; k_h, \mathcal{H})} \quad \forall I \geq 1 \right\} \geq 1 - \zeta,$$

with $C_1 = 8/\log(1+\sigma^{-2})$, $\gamma(I; k_h, \mathcal{H}) = \mathcal{O}(\tilde{N}\log(\tilde{N})\log(I))$ and $\tilde{N} = \prod_{j=1}^{d_h} n_j$.

C.3.5 Proof of Theorem 3

Let us first remind our restart strategy in the categorical setting. At the i-th restart, we first fit an auxiliary global GP model $GP(0, k_h)$ on a subset of data $\mathcal{D}^*_{i-1} = \{\mathbf{h}^*_j, f(\mathbf{h}^*_j)\}_{j=1}^{i-1}$, where $\mathbf{h}^*_j$ is the local maxima found after the j-th restart, or, a random data point, if the found local maxima after the j-th restart is same as one of previous restart. Let us also denote $\mu_{gl}(\mathbf{h}; \mathcal{D}^*_{i-1})$ and $\sigma^2_{gl}(\mathbf{h}; \mathcal{D}^*_{i-1})$ as the

[6] This means for every function g belonging to this class of functions, $g(\mathbf{h}^*_j) = f(\mathbf{h}^*_j)$ where $\mathbf{h}^*_j$ is a local maxima of f.

posterior mean and variance of the global GP learned from $\mathcal{D}^*_{i-1}$. Then, at the i-th restart, we select the following location $\mathbf{h}_i^{(0)}$ as the initial centre of the new TR:

$$\mathbf{h}_i^{(0)} = \arg\max_{\mathbf{h}\in\mathcal{H}} \mu_{gl}(\mathbf{h}; \mathcal{D}^*_{i-1}) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{h}; \mathcal{D}^*_{i-1}),$$

where β_i is the trade-off parameter in GP-UCB (Srinivas et al., 2009).

To prove the convergence property of CASMOPOLITAN, apart from Assumptions 1 & 2, let us also assume that at the i-th restart, there exists a function $g_i(\mathbf{h})$ that: (a) is a sample from the global $GP(0, k_h)$, (b) shares the same global maximum $\mathbf{h}^*$ with f, and, (c) passes through the all the local maxima of f and any data point $\mathbf{h}'$ in $\mathcal{D}^*_{i-1} \cup \{\mathbf{h}_i^{(0)}\}$ that are not local maxima (i.e. $g_i(\mathbf{h}') = f(\mathbf{h}')\ \forall \mathbf{h}' \in \mathcal{D}^*_{i-1} \cup \{\mathbf{h}_i^{(0)}\}$). In layman's terms, the function $g_i(\mathbf{h})$ is a function that passes through all the maxima of f and is a sample from the auxiliary global $GP(0, k_h)$. It is worth noting that our assumption is more relaxed than the assumption in (Srinivas et al., 2009) where it is assumed that the objective function f must be sampled from the global $GP(0, k_h)$. Specifically, it can be seen that if the assumption in (Srinivas et al., 2009) holds, our assumption also holds because if $f(\mathbf{h})$ is a sample from $GP(0, k_h)$, then a choice for $g_i(\mathbf{h})$ is $f(\mathbf{h})$, thus, our assumption holds.

Using Lemmas 5.1 and 5.2 in (Srinivas et al., 2009) for the function g_i, when $\beta_i = 2\log(|\mathcal{H}|i^2\pi^2/6\zeta)$, for all i, with probability $1-\zeta$, we have,

$$\begin{aligned}
&\mu_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}^*_{i-1}) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}^*_{i-1}) \\
&\quad \geq \mu_{gl}(\mathbf{h}^*; \mathcal{D}^*_{i-1}) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{h}^*; \mathcal{D}^*_{i-1}) \\
&\quad \geq g_i(\mathbf{h}^*).
\end{aligned}$$

Thus, with probability $1-\zeta$,

$$\begin{aligned}
&g_i(\mathbf{h}^*) - g_i(\mathbf{h}_i^{(0)}) \\
&\leq \mu_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}^*_{i-1}) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}^*_{i-1}) - g_i(\mathbf{h}_i^{(0)}) \\
&\leq 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}^*_{i-1}).
\end{aligned}$$

Combining this inequality with the fact that $g_i(\mathbf{h}_i^{(0)}) = f(\mathbf{h}_i^{(0)})$, and $g_i(\mathbf{h}_i^*) = f(\mathbf{h}_i^*)$, we have, with probability $1-\zeta$,

$$f(\mathbf{h}^*) - f(\mathbf{h}_i^{(0)}) \leq 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}^*_{i-1}).$$

Let us denote $\mathbf{h}_i^*$ as the local maxima found by CASMOPOLITAN at the i-th restart. As $f(\mathbf{h}_i^{(0)}) \leq f(\mathbf{h}_i^*)$, therefore,

$$f(\mathbf{h}^*) - f(\mathbf{h}_i^*) \leq 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}_{i-1}^*).$$

This results that, with probability $1 - \zeta$,

$$R_I = \sum_{i=1}^{I}(f(\mathbf{h}^*) - f(\mathbf{h}_i^*)) \leq \sum_{i=1}^{I} 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{h}_i^{(0)}; \mathcal{D}_{i-1}^*).$$

Finally, using Lemmas 5.3 and 5.4 in (Srinivas et al., 2009), we can bound R_I as $R_I \leq \sqrt{IC_1\beta_I\gamma(I; k_h, \mathcal{H})}$ with $C_1 = 8/\log(1 + \sigma^{-2})$ and $\gamma(I; k_h, \mathcal{H})$ being the maximum information gain for the categorical kernel derived in Theorem 1. □

C.3.6 Theorem on Global Convergence of Mixed Setting

Theorem 4 *Let us consider the mixed space setting, $f : [\mathcal{H}, \mathcal{X}] \to \mathbb{R}$. Let $\zeta \in (0, 1)$. Suppose the objective function f satisfies that: there exists a class of functions g which pass through all the local maximas of f, share the same global maximum with f and lies in the RKHS $\mathcal{G}_k([\mathcal{H}, \mathcal{X}])$ corresponding to the kernel k of the auxiliary global GP model. Suppose that the noise ϵ_i has zero mean conditioned on the history and is bounded by σ almost surely. Assume $\|g\|_k^2 \leq B$, and let $\beta_i = 2B + 300\gamma_i \log(i/\zeta)^3$, then given Assumptions 1 and 2, CASMOPOLITAN obtains a regret bound of $\mathcal{O}^*\Big(\sqrt{I\gamma(I; k, [\mathcal{H}, \mathcal{X}])\beta_I}\Big)$ w.h.p. Specifically,*

$$Pr\Big\{R_I \leq \sqrt{C_1 I\beta_I\gamma(I; k; [\mathcal{H}, \mathcal{X}])} \quad \forall I \geq 1\Big\} \geq 1 - \zeta,$$

with $C_1 = 8/\log(1 + \sigma^{-2})$, $\gamma(I; k; [\mathcal{H}, \mathcal{X}]) = \mathcal{O}\Big((\lambda\tilde{N} + 1 - \lambda)\gamma(T; k_x; \mathcal{X}) + (\tilde{N} + 2 - 2\lambda)\log T\Big)$ and $\tilde{N} = \prod_{j=1}^{d_h} n_j$.

C.3.7 Proof of Theorem 4

Similar to the proof for categorical setting in Section C.3.5, let us first remind our restart strategy in the mixed space setting. Suppose we are restarting the i-th time, we first fit the global GP model on a subset of data $\mathcal{D}_{i-1}^* = \{\mathbf{z}_j^*, f(\mathbf{z}_j^*)\}_{j=1}^{i-1}$, where $\mathbf{z}_j^*$ is the local maxima found after the j-th restart, or, a random data point, if the

found local maxima after the j-th restart is same as one of previous restart. Let us also denote $\mu_{gl}(\mathbf{z}; \mathcal{D}^*_{i-1})$ and $\sigma^2_{gl}(\mathbf{z}; \mathcal{D}^*_{i-1})$ as the posterior mean and variance of the global GP learned from $\mathcal{D}^*_{i-1}$. Then, at the i-th restart, we select the following location $\mathbf{z}_i^{(0)}$ as the initial centre of the new TR:

$$\mathbf{z}_i^{(0)} = \arg\max_{\mathbf{z}\in[\mathcal{H},\mathcal{X}]} \mu_{gl}(\mathbf{z}; \mathcal{D}^*_{i-1}) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{z}; \mathcal{D}^*_{i-1}),$$

where β_i is the trade-off parameter in GP-UCB Srinivas et al. (2009).

To prove the convergent property of CASMOPOLITAN in the mixed space setting, apart from Assumptions 1 & 2, let us also assume that at the i-th restart, there exists a function $g_i(\mathbf{z})$: (a) lies in the RKHS $\mathcal{G}_k([\mathcal{H}, \mathcal{X}])$ and $\|g_i\|_k^2 \leq B$, (b) shares the same global maximum $\mathbf{z}^*$ with f, and, (c) passes through all the local maxima of f and any data point $\mathbf{z}'$ in $\mathcal{D}^*_{i-1} \cup \{\mathbf{z}_i^{(0)}\}$ which are not local maxima (i.e. $g_i(\mathbf{z}') = f(\mathbf{z}')\ \forall \mathbf{z}' \in \mathcal{D}^*_{i-1} \cup \{\mathbf{z}_i^{(0)}\}$). In layman's terms, the function $g_i(\mathbf{z})$ is a function that passes through the maxima of f whilst lying in the RKHS $\mathcal{G}_k([\mathcal{H}, \mathcal{X}])$ and satisfying $\|g_i\|_k^2 \leq B$. Our assumption is more relaxed than (Srinivas et al., 2009) which assumed that the objective function f lies in the RKHS $\mathcal{G}_k([\mathcal{H}, \mathcal{X}])$. Specifically, it can be seen that if the assumption in (Srinivas et al., 2009) holds, our assumption also holds because if $f(\mathbf{z})$ lies in the RKHS $\mathcal{G}_k([\mathcal{H}, \mathcal{X}])$, then a choice for $g_i(\mathbf{z})$ is $f(\mathbf{z})$, thus, our assumption holds.

Using Theorem 6 in (Srinivas et al., 2009) for function g_i, when $\beta_i = 2\|g_i\|_k^2 + 300\gamma_i \log(i/\zeta)^3$, $\forall i$, $\forall z \in [\mathcal{H}, \mathcal{X}]$, we have,

$$\Pr\{|\mu_{gl}(\mathbf{z}; D^*_{i-1}) - g_i(\mathbf{z})| \leq \sqrt{\beta_i}\sigma_{gl}(\mathbf{z}; D^*_{i-1})|\} \geq 1 - \zeta. \tag{C.20}$$

Note since $\|g_i\|_k^2 \leq B$, Eq. (C.20) is also correct using $\beta_i = 2B + 300\gamma_i \log(i/\zeta)^3$. By using the inequality in Eq. (C.20), the proof technique is similar to that in Section C.3.5. In particular, with probability $1 - \zeta$, we have that,

$$\begin{aligned}&\mu_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}^*_{i-1}) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}^*_{i-1}) \\ &\quad \geq \mu_{gl}(\mathbf{h}^*; \mathcal{D}^*_{i-1}) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{z}^*; \mathcal{D}^*_{i-1}) \geq g_i(\mathbf{z}^*).\end{aligned} \tag{C.21}$$

Thus, with probability $1-\zeta$, we have

$$\begin{aligned} & g_i(\mathbf{z}^*) - g_i(\mathbf{z}_i^{(0)}) \\ & \leq \mu_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}_{i-1}^*) + \sqrt{\beta_i}\sigma_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}_{i-1}^*) - g_i(\mathbf{z}_i^{(0)}) \\ & \leq 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}_{i-1}^*). \end{aligned}$$

Since $g_i(\mathbf{z}_i^{(0)}) = f(\mathbf{z}_i^{(0)})$, and $g_i(\mathbf{z}_i^*) = f(\mathbf{z}_i^*)$, hence, $f(\mathbf{z}^*) - f(\mathbf{z}_i^{(0)}) \leq 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}_{i-1}^*)$ with probability $1-\zeta$. With $\mathbf{z}_i^*$ as the local maxima found by CASMOPOLITAN at the i-th restart. As $f(\mathbf{z}_i^{(0)}) \leq f(\mathbf{z}_i^*)$, therefore,

$$f(\mathbf{z}^*) - f(\mathbf{z}_i^*) \leq 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}_{i-1}^*).$$

This results, with probability $1-\zeta$,

$$R_I = \sum_{i=1}^{I}(f(\mathbf{z}^*) - f(\mathbf{z}_i^*)) \leq \sum_{i=1}^{I} 2\sqrt{\beta_i}\sigma_{gl}(\mathbf{z}_i^{(0)}; \mathcal{D}_{i-1}^*).$$

Finally, using Lemmas 5.3 and 5.4 in (Srinivas et al., 2009), we can bound R_I as $R_I \leq \sqrt{IC_1\beta_I\gamma(I; k, [\mathcal{H}, \mathcal{X}])}$ with $C_1 = 8/\log(1+\sigma^{-2})$ and $\gamma(I; k, [\mathcal{H}, \mathcal{X}])$ is the maximum information gain for the mixed kernel derived in Theorem 1. □

Discussion We show in Theorem 2 that our TR-based algorithm with BO converges to a local maxima or global maximum after a restart. We note that similar convergence can be found in the original TR-based algorithms using gradient-descent Yuan (2000). However, our proof technique is very different from (Yuan, 2000). In addition, in Theorems 3 and 4, the fact that CASMOPOLITAN converges to the global maximum with a sublinear rate over the number of restarts - not over the number of iterations as in (Srinivas et al., 2009) - can be considered as the price paid for a more relaxed assumption. In particular, Srinivas et al. (2009) assume that it is possible to model the objective function f with a GP with kernel k on the whole search space. On the other hand, we relax this assumption in Theorems 3 and 4 by assuming that there is a class of functions, which pass through the local maxima and share the same global maximum with f, that we can model with a GP with kernel k. Further details on this class of functions can be found in Apps. C.3.5 and C.3.7.

Despite the aforementioned strengths, there are some limitations with our theoretical analysis. First, the maximum information gains $\gamma(T; k_h; \mathcal{H})$ and $\gamma(T; k; [\mathcal{H}, \mathcal{X}])$ derived in Theorem 1 increase exponentially with the dimension of the categorical input (d_h). Thus, these terms can be large when the categorical dimension is high. As we are solving a noisy NP-hard combinatorial problem, it might not be possible to get away these exponential terms without a strict assumption. Second, as briefly discussed above, Assumption 2 is true asymptotically, resulting Theorems 2, 3 and 4 to hold asymptotically. One way to eliminate this assumption is to instead prove CASMOPOLITAN achieves ϵ - *accuracy*, that is, CASMOPOLITAN can find a point whose function value is within ϵ of the objective function global maximum, where ϵ is a small positive value depending on the minimum TR lengths $L^x_{\text{min}}, L^h_{\text{min}}$. We consider these directions for future work.

Bibliography

Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas Donald Lane. Zero-cost proxies for lightweight {nas}. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0cmMMy8J5q.

Robin Allesiardo, Raphaël Féraud, and Odalric-Ambrym Maillard. The non-stationary stochastic multi-armed bandit problem. *International Journal of Data Science and Analytics*, 3(4):267–283, 2017.

Ahsan Alvi, Binxin Ru, Jan-Peter Calliess, Stephen Roberts, and Michael A Osborne. Asynchronous batch bayesian optimisation with improved local penalisation. In *International Conference on Machine Learning*, pages 253–262. PMLR, 2019.

Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, and Mani Srivastava. Genattack: Practical black-box attacks with gradient-free optimization. *arXiv preprint arXiv:1805.11090*, 2018.

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. *arXiv preprint arXiv:1711.06104*, 2017.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

Peter Auer, Nicolo Cesa–Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002a.

Peter Auer, Nicolo Cesa–Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002b.

The GPyOpt authors. GPyOpt: A Bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

Javad Azimi, Alan Fern, and Xiaoli Z Fern. Batch Bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pages 109–117, 2010.

Javad Azimi, Ali Jalali, and Xiaoli Fern. Hybrid batch Bayesian optimization. *arXiv preprint arXiv:1202.5597*, 2012.

Kevin Bache and Moshe Lichman. Uci machine learning repository. 2013.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

Ricardo Baptista and Matthias Poloczek. Bayesian optimization of combinatorial structures. *arXiv preprint arXiv:1806.08838*, 2018.

J S Bergstra, R Bardenet, Y Bengio, and B Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011a.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011b.

John Bibby. Axiomatisations of the average and a further generalisation of monotonic sequences. *Glasgow Mathematical Journal*, 15(1):63–65, 1974.

Mickaël Binois, David Ginsbourger, and Olivier Roustant. A warped kernel improving robustness in Bayesian optimization via random embeddings. In *International Conference on Learning and Intelligent Optimization*, pages 281–286. Springer, 2015.

Mickaël Binois, David Ginsbourger, and Olivier Roustant. On the choice of the low-dimensional domain for global optimization via random embeddings. *Journal of global optimization*, 76(1):69–90, 2020.

Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

Laurens Bliek, Sicco Verwer, and Mathijs de Weerdt. Black-box mixed-variable optimisation using a surrogate model that satisfies integer constraints. *arXiv preprint arXiv:2006.04508*, 2020.

Salomon Bochner. *Lectures on Fourier Integrals: With an Author's Suppl. on Monotonic Functions, Stieltjes Integrals and Harmonic Analysis. Transl. from the Orig. by Morris Tennenbaum and Harry Pollard*. University Press, 1959.

Salomon Bochner. *Lectures on Fourier Integrals.(AM-42)*, volume 42. Princeton University Press, 2016.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2009.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010a.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010b.

Keith T Butler, Daniel W Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555, 2018.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence*, 2018a.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018b.

Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 10836–10846. Curran Associates, Inc., 2019.

Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. Robustness guarantees for bayesian inference with gaussian processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7759–7768, 2019.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 513–530, 2016.

Fabio Maria Carlucci, Pedro M Esperança, Marco Singh, Victor Gabillon, Antoine Yang, Hang Xu, Zewei Chen, and Jun Wang. {MANAS}: Multi-agent neural architecture search, 2020. URL https://openreview.net/forum?id=ryedqa4FwS.

Binghong Chen, Tianzhe Wang, Chengtao Li, Hanjun Dai, and Le Song. Molecule optimization by explainable evolution. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=jHefDGsorp5.

Jinyin Chen, Dunjie Zhang, Zhaoyan Ming, and Kejie Huang. Graphattacker: A general multi-task graphattack framework. *arXiv preprint arXiv:2101.06855*, 2021b.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Thirty-second AAAI conference on artificial intelligence*, 2018a.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Dr{nas}: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021c. URL https://openreview.net/forum?id=9FWas6YbmB3.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search. In *International Conference on Computer Vision (ICCV)*, 2019.

Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in AlphaGo. *arXiv preprint arXiv:1812.06855*, 2018b.

Clément Chevalier and David Ginsbourger. Fast computation of the multi-points expected improvement with applications in batch selection. In *International Conference on Learning and Intelligent Optimization*, pages 59–69. Springer, 2013.

Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013a.

Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013b.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13 (1):795–828, 2012.

Dennis D Cox and Susan John. A statistical method for global optimization. In *Systems, Man and Cybernetics, 1992., IEEE International Conference on*, pages 1241–1246. IEEE, 1992.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. FBNetV3: Joint architecture-recipe search using neural acquisition function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Erik Daxberger, Anastasia Makarova, Matteo Turchetta, and Andreas Krause. Mixed-variable Bayesian optimization. *arXiv preprint arXiv:1907.01329*, 2019.

Nathan de Lara and Edouard Pineau. A simple baseline algorithm for graph classification. *arXiv preprint arXiv:1810.09155*, 2018.

Thomas Desautels, Andreas Krause, and Joel W Burdick. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *The Journal of Machine Learning Research*, 15(1):3873–3923, 2014.

Aryan Deshwal, Syrine Belakaria, Janardhan Rao Doppa, and Alan Fern. Optimizing discrete spaces via expensive evaluations: A learning to search framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3773–3780, 2020.

Laurence Charles Ward Dixon and Giorgio P Szegö. *Towards global optimisation*. North-Holland Amsterdam, 1978.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=HJxyZkBKDr.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020b.

Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le. AutoHAS: Differentiable hyper-parameter and architecture search. *arXiv preprint arXiv:2006.03656*, 2020.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174, 2013.

David K Duvenaud, Hannes Nickisch, and Carl E Rasmussen. Additive gaussian processes. In *Advances in neural information processing systems*, pages 226–234, 2011.

Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv:1808.05377*, 2018.

Andries Petrus Engelbrecht, Ian Cloete, and Jacek M Zurada. Determining the significance of input parameters using sensitivity analysis. In *International Workshop on Artificial Neural Networks*, pages 382–388. Springer, 1995.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *Advances in Neural Information Processing Systems*, 32:5496–5507, 2019.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning (ICML)*, pages 1436–1445, 2018.

Stanislav Fort, Paweł Krzysztof Nowak, Stanislaw Jastrzebski, and Srini Narayanan. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

Peter Frazier, Warren Powell, and Savas Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

Peter I Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

Jacob Gardner, Chuan Guo, Kilian Weinberger, Roman Garnett, and Roger Grosse. Discovering and exploiting additive structure for bayesian optimization. In *Artificial Intelligence and Statistics*, pages 1311–1319. PMLR, 2017.

Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. *Neurocomputing*, 2019.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer, 2003.

Swarnendu Ghosh, Nibaran Das, Teresa Gonçalves, Paulo Quaresma, and Mahantapas Kundu. The journey of graph kernels through two decades. *Computer Science Review*, 27:88–111, 2018.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. A multi-points criterion for deterministic parallel global optimization based on gaussian processes. 2008.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. Kriging is well-suited to parallelize optimization. In *Computational Intelligence in Expensive Optimization Problems*, pages 131–162. Springer, 2010a.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. Kriging is well-suited to parallelize optimization. In *Computational intelligence in expensive optimization problems*, pages 131–162. Springer, 2010b.

David Ginsbourger, Janis Janusevskis, and Rodolphe Le Riche. Dealing with asynchronicity in parallel gaussian process based global optimization. In *4th International Conference of the ERCIM WG on computing & statistics (ERCIM'11)*, 2011.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12:2211–2268, 2011.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch bayesian optimization via local penalization. In *Artificial Intelligence and Statistics*, pages 648–657, 2016a.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil D Lawrence. Batch Bayesian optimization via local penalization. In *International Conference on Artificial Intelligence and Statistics*, pages 648–657, 2016b.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Shivapratap Gopakumar, Sunil Gupta, Santu Rana, Vu Nguyen, and Svetha Venkatesh. Algorithmic assurance: An active approach to algorithmic testing using Bayesian optimisation. In *Advances in Neural Information Processing Systems*, pages 5465–5473, 2018.

Diego Granziol and Stephen Roberts. An Information and Field Theoretic approach to the Grand Canonical Ensemble, 2017.

Diego Granziol, Binxin Ru, Stefan Zohren, Xiaowen Dong, Michael Osborne, and Stephen Roberts. Meme: An accurate maximum entropy method for efficient approximations in large-scale machine learning. *Entropy*, 21(6):551, 2019.

Antoine Grosnit, Rasul Tutunov, Alexandre Max Maraval, Ryan-Rhys Griffiths, Alexander I Cowen-Rivers, Lin Yang, Lin Zhu, Wenlong Lyu, Zhitang Chen, Jun Wang, et al. High-dimensional bayesian optimisation with variational autoencoders and deep metric learning. *arXiv preprint arXiv:2106.03609*, 2021.

Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

Tom Gunter, Michael A Osborne, Roman Garnett, Philipp Hennig, and Stephen J Roberts. Sampling for inference in probabilisktic models with fast Bayesian quadrature. In *Advances in neural information processing systems*, pages 2789–2797, 2014.

Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A distribution-free theory of nonparametric regression*. Springer Science & Business Media, 2006.

William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, pages 1225–1234, 2016.

David Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California ..., 1999.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

JB Heaton, NG Polson, and Jan Hendrik Witte. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13:1809–1837, 2012a.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012b.

Philipp Hennig, Michael A Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proc. R. Soc. A*, 471(2179):20150142, 2015.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pages 918–926, 2014.

José Miguel Hernández-Lobato, Michael Gelbart, Matthew Hoffman, Ryan Adams, and Zoubin Ghahramani. Predictive entropy search for Bayesian optimization with unknown constraints. In *International Conference on Machine Learning*, pages 1699–1707, 2015.

José Miguel Hernández-Lobato, James Requeima, Edward O Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1470–1479. JMLR. org, 2017.

Tobias Hinz, Nicolás Navarro-Guerrero, Sven Magg, and Stefan Wermter. Speeding up the hyperparameter optimization of deep convolutional neural networks. *International Journal of Computational Intelligence and Applications*, page 1850008.

Matthew W. Hoffman and Zoubin Ghahramani. Output-space predictive entropy search for flexible global optimization. In *the NIPS workshop on Bayesian optimization*, 2015.

Frank Höppner and Maximilian Jahnke. Enriched weisfeiler-lehman kernel for improved graph clustering of source code. In *International Symposium on Intelligent Data Analysis*, pages 248–260. Springer, 2020.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Yingjie Hu, JianQiang Hu, Yifan Xu, Fengchun Wang, and Rong Zeng Cao. Contamination control in food supply chain. In *Proceedings of the 2010 Winter Simulation Conference*, pages 2678–2681. IEEE, 2010.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

Marco F Huber, Tim Bailey, Hugh Durrant-Whyte, and Uwe D Hanebeck. On entropy approximation for Gaussian mixture random vectors. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pages 181–188. IEEE, 2008.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011a.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011b.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598*, 2018.

Ali Jalali, Javad Azimi, Xiaoli Fern, and Ruofei Zhang. A lipschitz exploration-exploitation scheme for Bayesian optimization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 210–224. Springer, 2013.

E. T. Jaynes. Information theory and statistical mechanics. *Phys. Rev.*, 106:620–630, May 1957. doi: 10.1103/PhysRev.106.620. URL http://link.aps.org/doi/10.1103/PhysRev.106.620.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgIPJBFvH.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, pages 1–11, 2021.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning*, pages 295–304, 2015.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. Parallelised bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 133–142, 2018a.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2016–2025, 2018b.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 321–328, 2003.

Henry Kenlay, Dorina Thanou, and Xiaowen Dong. Interpretable stability bounds for spectral graph filters, 2021.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Donghwan Kim and Jeffrey A Fessler. Adaptive restart of the optimized gradient method for convex optimization. *Journal of Optimization Theory and Applications*, 178(1): 240–263, 2018.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Andreas Kirsch, Joost Van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems*, 32:7026–7037, 2019.

Johannes Kirschner, Mojmir Mutny, Nicole Hiller, Rasmus Ischebeck, and Andreas Krause. Adaptive and safe bayesian optimization in high dimensions via one-dimensional subspaces. In *International Conference on Machine Learning*, pages 3429–3438. PMLR, 2019.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. *arXiv:1605.07079*, 2016a.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. 2016b.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894. JMLR. org, 2017.

Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *Advances in Neural Information Processing Systems*, pages 2990–2998, 2016.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth. Active testing: Sample-efficient model evaluation. *arXiv preprint arXiv:2103.05331*, 2021.

Andreas Krause and Cheng Soon Ong. Contextual gaussian process bandit optimization. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 2447–2455, 2011.

Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*, pages 1623–1631, 2016.

Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Brian Kulis and Michael I Jordan. Revisiting k-means: New algorithms via Bayesian nonparametrics. *arXiv preprint arXiv:1111.0352*, 2011.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Ben Letham, Roberto Calandra, Akshara Rai, and Eytan Bakshy. Re-examining linear embeddings for high-dimensional bayesian optimization. *Advances in Neural Information Processing Systems*, 33, 2020.

Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv:1902.07638*, 2019.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR, 2020.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020a.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv:1603.06560*, 2016.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Zhihang Li, Teng Xi, Jiankang Deng, Gang Zhang, Shengzhao Wen, and Ran He. Gp-nas: Gaussian process based neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11933–11942, 2020b.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Chih-Long Lin. Hardness of approximating graph transformation problem. In *International Symposium on Algorithms and Computation*, pages 74–82. Springer, 1994.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *European Conference on Computer Vision (ECCV)*, pages 19–34, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Tongliang Liu, Gábor Lugosi, Gergely Neu, and Dacheng Tao. Algorithmic stability and hypothesis complexity. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2159–2167. JMLR. org, 2017.

Yi Liu, Jie Ling, Zhusong Liu, Jian Shen, and Chongzhi Gao. Finger vein secure biometric template generation based on deep learning. *Soft Computing*, 22(7): 2257–2265, 2018c.

Daniel James Lizotte. *Practical bayesian optimization*. University of Alberta, 2008.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 7816–7827, 2018.

Clare Lyle, Lisa Schut, Binxin Ru, Mark van der Wilk, and Yarin Gal. A Bayesian perspective on training speed and model selection. *Thirty-fourth Conference on Neural Information Processing Systems*, 2020.

Wenlong Lyu, Fan Yang, Changhao Yan, Dian Zhou, and Xuan Zeng. Batch bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design. In *International Conference on Machine Learning*, pages 3312–3320, 2018.

Lizheng Ma, Jiaxu Cui, and Bo Yang. Deep neural architecture search with deep graph Bayesian optimization. In *Web Intelligence (WI)*, pages 500–507. IEEE/WIC/ACM, 2019a.

Yao Ma, Suhang Wang, Tyler Derr, Lingfei Wu, and Jiliang Tang. Attacking graph convolutional networks via rewiring. *arXiv preprint arXiv:1906.03750*, 2019b.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Vladimir Mazya and Tatyana Shaposhnikova. *Jacques Hadamard: A Universal Mathematician*. 1st edition, 1999.

David A McAllester. Pac-bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 164–170, 1999.

Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. *arXiv preprint arXiv:2006.04647*, 2020.

J Močkus, V Tiesis, and A Žilinskas. Toward global optimization, volume 2, chapter the application of Bayesian methods for seeking the extremum, 1978.

Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

Seungyong Moon, Gaon An, and Hyun Oh Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. *arXiv preprint arXiv:1905.06635*, 2019.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip HS Torr, and Puneet K Dokania. Calibrating deep neural networks using focal loss. *arXiv preprint arXiv:2002.09437*, 2020.

Iain Murray, Ryan Prescott Adams, and David JC MacKay. Elliptical slice sampling. 2010.

Mojmir Mutny and Andreas Krause. Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9019–9030, 2018.

Amin Nayebi, Alexander Munteanu, and Matthias Poloczek. A framework for bayesian optimization in embedded subspaces. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4752–4761. PMLR, 2019.

Jeffrey Negrea, Mahdi Haghifam, AI Element, Gintare K Dziugaite, Ashish Khisti, and Daniel M Roy. Information-theoretic generalization bounds for sgld via data-dependent estimates. *33rd International Conference on Information Processing Systems*, 2019.

Gergely Neu. Information-theoretic generalization bounds for stochastic gradient descent, 2021.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

Dang Nguyen, Sunil Gupta, Santu Rana, Alistair Shilton, and Svetha Venkatesh. Bayesian optimization for categorical and category-specific continuous inputs. *arXiv preprint arXiv:1911.12473*, 2019.

Dang Nguyen, Sunil Gupta, Santu Rana, Alistair Shilton, and Svetha Venkatesh. Bayesian optimization for categorical and category-specific continuous inputs. In *Proc. of AAAI*, volume 34, pages 5256–5263, 2020.

Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *arXiv preprint arXiv:1904.12218*, 2019.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

ChangYong Oh, Efstratios Gavves, and Max Welling. Bock: Bayesian optimization with cylindrical kernels. *arXiv preprint arXiv:1806.01619*, 2018.

Changyong Oh, Jakub Tomczak, Efstratios Gavves, and Max Welling. Combinatorial bayesian optimization using the graph cartesian product. In *Advances in Neural Information Processing Systems*, pages 2910–2920, 2019.

Nicolas Papernot and Patrick McDaniel. On the effectiveness of defensive distillation. *arXiv preprint arXiv:1607.05113*, 2016.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519. ACM, 2017.

Julien Pelamatti, Loïc Brevault, Mathieu Balesdent, El-Ghazali Talbi, and Yannick Guerin. Overview and comparison of gaussian process-based surrogate models for mixed continuous and discrete variables: Application on aerospace design problems. In *High-Performance Simulation-Based Optimization*, pages 189–224. Springer, 2020.

Valerio Perrone, Huibin Shen, Aida Zolic, Iaroslav Shcherbatyi, Amr Ahmed, Tanya Bansal, Michele Donini, Fela Winkelmolen, Rodolphe Jenatton, Jean Baptiste Faddoul, et al. Amazon sagemaker automatic model tuning: Scalable gradient-free optimization. 2021.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, pages 4092–4101, 2018.

Steve Pressé, Kingshuk Ghosh, Julian Lee, and Ken A. Dill. Principles of Maximum Entropy and Maximum Caliber in Statistical Physics. *Reviews of Modern Physics*, 85: 1115–1141, Jul 2013.

Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *ICCV*, 2019.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

Alvin Rajkomar, Jeffrey Dean, and Isaac Kohane. Machine learning in medicine. *New England Journal of Medicine*, 380(14):1347–1358, 2019.

Dhanesh Ramachandram, Michal Lisicki, Timothy J Shields, Mohamed R Amer, and Graham W Taylor. Bayesian optimization on graph-structured search spaces: Optimizing deep multimodal fusion architectures. *Neurocomputing*, 298:80–89, 2018.

Santu Rana, Cheng Li, Sunil Gupta, Vu Nguyen, and Svetha Venkatesh. High dimensional Bayesian optimization with elastic Gaussian process. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 2883–2891, 2017.

C E Rasmussen and C K I Williams. *Gaussian processes for machine learning*. 2006a.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

Carl Edward Rasmussen and Christopher KI Williams. Gaussian processes for machine learning. 1, 2006b.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017a.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning (ICML)*, pages 2902–2911, 2017b.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv:1802.01548*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

James Ryan Requeima. Integrated predictive entropy search for bayesian optimization. 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

Paul Rolland, Jonathan Scarlett, Ilija Bogunovic, and Volkan Cevher. High-dimensional bayesian optimization via additive models with overlapping groups. In Amos J. Storkey and Fernando Pérez-Cruz, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pages 298–307. PMLR, 2018.

Nikitas Rontsis, Michael A Osborne, and Paul J Goulart. Distributionally robust optimization techniques in batch bayesian optimization. *arXiv preprint arXiv:1707.04191*, 2017.

Binxin Ru, Mark McLeod, Diego Granziol, and Michael A Osborne. Fast information-theoretic Bayesian optimisation. In *International Conference on Machine Learning*, 2018.

Binxin Ru, Ahsan Alvi, Vu Nguyen, Michael A Osborne, and Stephen Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning*, pages 8276–8285. PMLR, 2020a.

Binxin Ru, Pedro Esperança, and Fabio Maria Carlucci. Neural architecture generator optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12057–12069. Curran Associates, Inc., 2020b. URL https://proceedings.neurips.cc/paper/2020/file/8c53d30ad023ce50140181f713059ddf-Paper.pdf.

Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=j9Rv7qdXjd.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1): 1–96, 2018.

Amar Shah and Zoubin Ghahramani. Parallel predictive entropy search for batch global optimization of expensive objective functions. In *Advances in Neural Information Processing Systems*, pages 3312–3320, 2015a.

Amar Shah and Zoubin Ghahramani. Parallel predictive entropy search for batch global optimization of expensive objective functions. In *Advances in Neural Information Processing Systems*, pages 3330–3338, 2015b.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016a.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016b.

Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Advances in Neural Information Processing Systems*, pages 11227–11237, 2019.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.

Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T Kwok, and Tong Zhang. Multi-objective neural architecture search via predictive network performance optimization, 2019.

Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. In *International Conference on Learning Representations*, 2019.

Oleg V Shylo, Timothy Middelkoop, and Panos M Pardalos. Restart strategies in optimization: parallel and serial cases. *Parallel Computing*, 37(1):60–68, 2011.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

John Skilling. The eigenvalues of mega-dimensional matrices. In *Maximum Entropy and Bayesian Methods*, pages 455–466. Springer, 1989.

Samuel L. Smith, Benoit Dherin, David G. T. Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2021.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012a.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012b.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 4134–4142, 2016a.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4134–4142, 2016b.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy?–a comprehensive study on the robustness of 18 deep image classification models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–648, 2018.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

Fnu Suya, Yuan Tian, David Evans, and Paolo Papotti. Query-limited black-box attacks to classifiers. *NIPS Workshop*, 2017.

Kevin Swersky, Yulia Rubanova, David Dohan, and Kevin Murphy. Amortized bayesian optimization over discrete spaces. In *Conference on Uncertainty in Artificial Intelligence*, pages 769–778. PMLR, 2020.

Laura P Swiler, Patricia D Hough, Peter Qian, Xu Xu, Curtis Storlie, and Herbert Lee. Surrogate models for mixed discrete-continuous variables. In *Constraint Programming and Decision Making*, pages 181–202. Springer, 2014.

James Joseph Sylvester. Xxxvii. on the relation between the minor determinants of linearly equivalent quadratic functions. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1(4):295–305, 1851. doi: 10.1080/14786445108646735.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. *arXiv preprint arXiv:1805.11770*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Jean-Philippe Vert. The optimal assignment kernel is not positive definite. *arXiv preprint arXiv:0801.4061*, 2008.

Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009. URL http://www.springerlink.com/index/T670U067V47922VK.pdf.

Xingchen Wan, Henry Kenlay, Binxin Ru, Arno Blaas, Michael Osborne, and Xiaowen Dong. Attacking graph classification via bayesian optimisation. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021a.

Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A Osborne. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. *International Conference on Machine Learning (ICML) 38*, 2021b.

Bochao Wang, Hang Xu, Jiajin Zhang, Chen Chen, Xiaozhi Fang, Ning Kang, Lanqing Hong, Wei Zhang, Yong Li, Zhicheng Liu, Zhenguo Li, Wenzhi Liu, and Tong Zhang. Vega: Towards an end-to-end configurable automl pipeline, 2020.

Jialei Wang, Scott C Clark, Eric Liu, and Peter I Frazier. Parallel Bayesian global optimization of expensive functions. *arXiv preprint arXiv:1602.05149*, 2016a.

Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning action space. *arXiv preprint arXiv:1906.06832*, 2019.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. *arXiv preprint arXiv:1703.01968*, 2017a.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning (ICML)*, 2017b.

Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional bayesian optimization via structural kernel learning. In *International Conference on Machine Learning*, pages 3656–3664, 2017.

Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale bayesian optimization in high-dimensional spaces. In Amos J. Storkey and Fernando Pérez-Cruz, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pages 745–754. PMLR, 2018.

Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, Nando De Freitas, et al. Bayesian optimization in high dimensions via random embeddings.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016b.

David Warde-Farley and Ian Goodfellow. 11 adversarial perturbations of deep neural networks. *Perturbations, Optimization, and Statistics*, 311, 2016.

Duncan J Watts and Steven H Strogatz. Collective dynamics of ?small-world?networks. *nature*, 393(6684):440–442, 1998.

Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9): 12–16, 1968.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. 2021a.

Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*, 2021b.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

David P Wipf, Srikantan S Nagarajan, J Platt, D Koller, and Y Singer. A new view of automatic relevance determination. In *NIPS*, pages 1625–1632, 2007.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 199–214. Springer, 2016.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018.

Jian Wu and Peter Frazier. The parallel knowledge gradient method for batch Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 3126–3134, 2016a.

Jian Wu and Peter I. Frazier. The parallel knowledge gradient method for batch Bayesian optimization. In *NIPS*, 2016b.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1284–1293, 2019a.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019b.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. *arXiv preprint arXiv:1907.05737*, 2019b.

Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly hard. In *International Conference on Learning Representations (ICLR)*, 2020.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, pages 7105–7114, 2019.

Jiaxuan You, J. Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks. *International Conference on Machine Learning*, abs/2007.06559, 2020a.

Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks. In *International Conference on Machine Learning (ICML)*, pages 10881–10891. PMLR, 2020b.

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1803–1811, 2019.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1loF2NFwr.

Ya-xiang Yuan. A review of trust region algorithms for optimization. In *Iciam*, volume 99, pages 271–282. Citeseer, 2000.

Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris Holmes, Frank Hutter, and Yee Whye Teh. Neural ensemble search for performant and calibrated predictions. *arXiv preprint arXiv:2006.08573*, 2020.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJx9ngStPH.

Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rkgW0oA9FX.

Yichi Zhang, Daniel W Apley, and Wei Chen. Bayesian optimization for materials design with mixed quantitative and qualitative variables. *Scientific reports*, 10(1):1–13, 2020.

Pu Zhao, Sijia Liu, Pin-Yu Chen, Nghia Hoang, Kaidi Xu, Bhavya Kailkhura, and Xue Lin. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 121–130, 2019.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020a.

Kaichen Zhou, Lanqing HONG, Fengwei Zhou, Binxin Ru, Zhenguo Li, Trigoni Niki, and Jiashi Feng. Diffauto{ml}: Differentiable joint optimization for efficient end-to-end automated machine learning, 2021. URL https://openreview.net/forum?id=pQ-AoEbNYQK.

Pan Zhou, Caiming Xiong, Richard Socher, and Steven CH Hoi. Theory-inspired path-regularized differential network architecture search. *arXiv preprint arXiv:2006.16537*, 2020b.

Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, 2018a.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018b.