Inductive Biases and Generalisation for Deep Reinforcement Learning

Maximilian Igl

Kellogg College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michelmas 2020/21

Abstract

In this thesis we aim to improve generalisation in deep reinforcement learning. Generalisation is a fundamental challenge for any type of learning, determining how acquired knowledge can be transferred to new, previously unseen situations. We focus on reinforcement learning, a framework describing how artificial agents can learn to interact with their environment to achieve goals. In recent years, by using neural networks to represent agents, it has achieved remarkable success and vastly expanded its scope of possible applications. Our goal is to improve the performance of these agents by allowing them to learn faster, to learn better solutions and to react robustly to previously unseen situations. On this quest, we explore a range of different methods and approaches.

We focus on incorporating additional structures, also called inductive biases, into the agent. Focussing on specific, yet widely applicable problem domains, we can develop specialised architectures which greatly improve performance. In Chapter 3 we focus on partially observable environments in which the agent is prevented full access to all task-relevant information at every moment in time. In Chapter 4 we turn our attention to multi-task and transfer learning and devise a novel training method allowing us to train hierarchically structured agents. Our method optimises for re-usability of individual solutions, greatly enhancing performance in transfer settings.

In the second part of this thesis, we turn our attention towards *regularisation*, another form of inductive bias, as a means to improve generalisation of deep agents. In Chapter 5 we first explore stochastic regularisation in reinforcement learning (RL). While these techniques have proven highly effective in supervised learning, we highlight and overcome difficulties in applying them directly to online RL algorithms, one of the most powerful and widely used types of learning in RL. In Chapter 6 we investigate generalisation in deep RL on a more fundamental level by exploring how transient non-stationarity in the training data can interfere with the stochastic gradient training of neural networks and can bias them towards worse solutions. Many state of the art RL algorithms introduce these types of non-stationarity into the training, even in stationary environments, by using a continuously improving policy for data collection. We propose a novel framework to reduce the non-stationarity experienced by the trained policy, thereby allowing for improved generalisation.

Inductive Biases and Generalisation for Deep Reinforcement Learning

Maximilian Igl

Kellogg College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michelmas 2020/21

Acknowledgements

I am immensely grateful for the support, friendship and advice from a large number of people without whom this thesis would not have been possible. They helped me learn, acted as role models, provided inspiration and allowed me to grow as a person.

First and foremost, I would like to thank my advisor Shimon Whiteson for his support and advice, giving me the opportunity to explore ideas freely but always guiding me by providing insightful perspectives and feedback. I would also like to thank Frank Wood for co-advising me during my first year until our ways sadly parted. His advice and supervision influenced my interests long after he left Oxford.

I would like to thank all friends and colleagues from Oxford for countless discussions, their immense help and an incredible supportive working environment. Much of this thesis would not exist without my collaborators' patience in spending many, many hours on whiteboards taming my crazy ideas. A big thank you to Wendelin Bömer, Narayanaswamy Siddharth, Luisa Zintgraf, Greg Farquhar, Jelena Luketina, Jakob Foerster, Jinke He, Tuan-Anh Le, Nantas Nardelli, Christian Schroeder, Vitaly Kurin, Matt Smith, Kristian Hartikainen, Tom Rainforth, Tim Rocktäschel, Tabish Rashid and Matthew Fellows. I would also like to greatly thank my colleagues at Microsoft and DeepMind, including Sam Devlin, Nicolas Heess, Katja Hofmann, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Markus Wulfmeier, Dhruva Tirumala as well as many others. A big thank you also to the AIMS CDT for supporting me and in particular to Wendy Poole. I would also like to thank Erwin Frey and Marianne Bauer without whom I would not have found my way to the opportunities I had in the last few years.

Thank you to all of my friends, old and new, for great times, their support, inspiration and patience when I was unresponsive for long stretches of time: Fabian, Lisa, Yuki, Natasha & David, Tim & Haru, Christina, Katharina, Ernesto, Vincent, Sophia, Sergey, Kris, Ulla, Flo & Sarah, Gattal, Gifti, Andrea, Benny, Martin, Roman and many more.

Thank you, more than I could express in words, to my parents and role-models Elisabeth Bürzle-Igl and Alexander Igl for their limitless love and support, the opportunities they've provided me with and for always encouraging me. Thank you also to my amazing siblings Juliane and Konstantin. Finally, an enormous thank you to my fiancée Austeja who has been incredibly supportive and understanding throughout these years, and with whom I am looking forward to start the next chapters of my life.

Abstract

In this thesis we aim to improve generalisation in deep reinforcement learning. Generalisation is a fundamental challenge for any type of learning, determining how acquired knowledge can be transferred to new, previously unseen situations. We focus on reinforcement learning, a framework describing how artificial agents can learn to interact with their environment to achieve goals. In recent years, by using neural networks to represent agents, it has achieved remarkable success and vastly expanded its scope of possible applications. Our goal is to improve the performance of these agents by allowing them to learn faster, to learn better solutions and to react robustly to previously unseen situations. On this quest, we explore a range of different methods and approaches.

We focus on incorporating additional structures, also called inductive biases, into the agent. Focussing on specific, yet widely applicable problem domains, we can develop specialised architectures which greatly improve performance. In Chapter 3 we focus on partially observable environments in which the agent is prevented full access to all task-relevant information at every moment in time. In Chapter 4 we turn our attention to multi-task and transfer learning and devise a novel training method allowing us to train hierarchically structured agents. Our method optimises for re-usability of individual solutions, greatly enhancing performance in transfer settings.

In the second part of this thesis, we turn our attention towards *regularisation*, another form of inductive bias, as a means to improve generalisation of deep agents. In Chapter 5 we first explore stochastic regularisation in RL. While these techniques have proven highly effective in supervised learning, we highlight and overcome difficulties in applying them directly to online RL algorithms, one of the most powerful and widely used types of learning in RL. In Chapter 6 we investigate generalisation in deep RL on a more fundamental level by exploring how transient non-stationarity in the training data can interfere with the stochastic gradient training of neural networks and can bias them towards worse solutions. Many state of the art RL algorithms introduce these types of non-stationarity into the training, even in stationary environments, by using a continuously improving policy for data collection. We propose a novel framework to reduce the non-stationarity experienced by the trained policy, thereby allowing for improved generalisation.

Contents

List of Figures **viii**

List of Abbreviations **xiv**

1 Introduction **1**

1.1 Generalisation in Machine Learning 1
1.2 Generalisation in Deep Reinforcement Learning 5
1.3 Overview 7

2 Background **9**

2.1 Probability and information theory 10
2.1.1 Probability theory 10
2.1.2 Bayesian inference 12
2.1.3 Information Theory 14
2.2 Reinforcement learning 15
2.2.1 Markov Decision Processes 16
2.2.2 Solving a Markov Decision Process: Value-function and policy 18
2.2.3 Value function methods 18
2.2.4 Policy gradient methods 21
2.2.5 A note on notation 22
2.3 Function approximation 22
2.3.1 Deep Learning 23
2.3.2 Deep variational Bayes 24
2.3.3 Deep reinforcement learning 25

I Inductive Biases for Reinforcement Learning **27**

3 Deep variational reinforcement learning for Partially Observable Markov Decisions Processes **29**

3.1 Introduction 29
3.2 Background 32
3.2.1 Variational Autoencoders for time series 32

3.2.2 Advantage Actor Critic 33
3.3 Method 33
3.3.1 Baseline architecture 34
3.3.2 Extending the latent state 35
3.3.3 Recurrent latent state update 37
3.3.4 Loss function 38
3.4 Related work 39
3.5 Experiments 41
3.5.1 Mountain hike 41
3.5.2 Atari 43
3.5.3 Ablation studies 45
3.6 Conclusion 46

4 Multitask soft option learning **48**
4.1 Introduction 48
4.2 Background 51
4.2.1 Planning as inference 51
4.2.2 Options 52
4.2.3 Multi-task learning 53
4.3 Method 53
4.3.1 Hierarchical posterior policies 55
4.3.2 Hierarchical prior policy 55
4.3.3 Objective 56
4.3.4 Soft vs. classical options 57
4.3.5 Local optima option learning 58
4.4 Related work 59
4.5 Experiments 61
4.5.1 Moving bandits 63
4.5.2 Taxi 65
4.5.3 Out-of-distribution tasks 66
4.6 Discussion 67

II Generalisation in Reinforcement Learning **70**

5 Generalization in Reinforcement Learning with Selective Noise Injection **72**
5.1 Introduction 72
5.2 Background 74
5.2.1 Regularislarisation Techniques in Supervised Learning 74

5.3 The Problem of Using Stochastic Regularisation in RL 76
5.4 Method 77
5.4.1 Selective Noise Injection 77
5.4.2 Information Bottleneck Actor Critic 79
5.5 Experiments 81
5.5.1 Learning Features in the Low-Data Regime 81
5.5.2 Multiroom 82
5.5.3 Coinrun 84
5.6 Related Work 86
5.7 Conclusion 88

6 The Impact of Non-stationarity on Generalisation in Deep Reinforcement Learning 90
6.1 Introduction 90
6.2 Background 92
6.3 The Impact of Non-stationarity on Generalisation 93
6.4 Iterated Relearning 95
6.4.1 Distillation Loss 97
6.4.2 Combining Training and Distillation 98
6.5 Experiments 99
6.5.1 Experimental Results on Multiroom 100
6.5.2 Experimental Results on Sokoban 101
6.5.3 Experimental Results on ProcGen 101
6.5.4 Supervised Learning on CIFAR-10 103
6.6 Related Work 106
6.7 Conclusion 107

7 Afterword 109

Appendices

A Deep Variational Reinforcement Learning 113
A.1 Experiments 113
A.1.1 Implementation Details 113
A.1.2 Additional Experiments and Visualisations 114
A.1.3 Computational Speed 115
A.1.4 Model Predictions 115
A.2 Algorithms 119

B Multitask Soft Option Learning **123**
B.1 Pseudo code 123
B.2 MSOL training details 123
B.2.1 Optimisation 123
B.2.2 Training schedule 125
B.3 Architecture 125
B.4 Hyper-parameters and additional environment details 126
B.4.1 Moving bandits 126
B.4.2 Taxi 127

C Information Bottleneck Actor Critic **130**
C.1 Dropout with SNI 130
C.2 Supervised Classification Task 130
C.3 Multiroom 132
C.4 Coinrun 133

D Iterated Relearning **135**
D.1 Pseudo code 135
D.2 Supervised Learning 136
D.2.1 Multiroom 137
D.2.2 Sokoban 138
D.2.3 ProcGen 138

References **141**

List of Figures

3.1 Comparison of RNN and DVRL encoders. 30

3.2 Overview of deep variational reinforcement learning (DVRL). We do the following K times to compute our new belief $\hat{b}_t$: Sample an ancestor index u_{t-1}^k based on the previous weights $w_{t-1}^{1:K}$ (Eq. 3.9). Pick the ancestor particle value $h_{t-1}^{u_{t-1}^k}$ and use it to sample a new stochastic latent state z_t^k from the encoder q_ϕ (Eq. 3.10). Compute h_t^k (Eq. 3.11) and w_t^k (Eq. 3.12). Aggregate all K values into the new belief $\hat{b}_t$ and summarise them into a vector representation $\hat{h}_t$ using a second recurrent neural network (RNN). Actor and critic can now condition on $\hat{h}_t$ and $\hat{b}_t$ is used as input for the next timestep. Red arrows denote random sampling, green arrows the aggregation of K values. Black solid arrows denote the passing of a value as argument to a function and black dashed ones the evaluation of a value under a distribution. Boxes indicate neural networks. Distributions are normal or Bernoulli distributions whose parameters are outputs of the neural network. 36

3.3 Mountain Hike is a continuous control task with observation noise $\sigma_o = 3$. Background colour indicates rewards. Red line: trajectory for RNN based encoder. Black line: trajectory for DVRL encoder. Dots: received observations. Both runs share the same noise values $\epsilon_{i,t}$. DVRL achieves higher returns (see Fig. 3.4) by better estimating its current location and remaining on the high reward mountain ridge. 42

3.4 Returns achieved in Mountain Hike. Solid lines: DVRL. Dashed lines: RNN. Colour: Noise levels. *Inset:* Difference in performance between RNN and DVRL for same level of noise: $\Delta\bar{J}(\sigma_o) = \bar{J}(\text{DVRL}, \sigma_o) - \bar{J}(\text{RNN}, \sigma_o)$. DVRL achieves slighly higher returns for the fully observable case and, crucially, its performance deteriorates more slowly for increasing observation noise, showing the advantage of DVRL's inference computations in encoding the history in the presence of observation noise. 43

3.5 Ablation studies on flickering ChopperCommand (Atari). 45

4.1 Two hierarchical posterior policies (left and right) with common priors (middle). For each task i, the policy conditions on the current state s_t^i and the last selected option z_{t-1}^i. It samples, in order, whether to terminate the last option (b_t^i), which option to execute next (z_t^i) and what primitive action (a_t^i) to execute in the environment. 54

4.2 Hierarchical learning of two concurrent tasks (a and b) using two options (z_1 and z_2) to reach two relevant targets (A and B). a) Local optimum when simply sharing options across tasks. b) Escaping the local optimum by using prior ($\bar{z}_i$) and posterior ($z_i^{(j)}$) policies. c) Learned options after training. Details are given in the text in Section 4.3.5. 58

4.3 Performance of applying the learned options and exploration priors to new tasks. Each line is the median over 5 random seeds (2 for MLSH) and shaded areas indicated standard deviations. Performance during the training phase is shown in Fig. B.1. *Moving Bandits* (a) is a simple environment capturing the effects described in Section 4.3.5. The results show that Meta Learning of Shared Hierarchies (MLSH), which uses hard options, struggles with local minima during the learning phase, whereas Multitask Soft Option Learning (MSOL) is able to learn a diverse set of options. *Taxi* (b) and *Directional Taxi* (c) additionally require good termination policies, which MLSH cannot learn as it uses a fixed option duration. See Fig. 4.4 for a visualization of the options and terminations learned by MSOL. DISTRAL(+action) is a strong non-hierarchical baseline which uses the last action as option-heuristic, but suffers when that action is not very informative, for example in (c). 62

4.4 Options learned with MSOL on the taxi domain, one option per column, before (top) and after pickup (bottom). The light gray area indicates walls. The colored arrows indicate the direction of the most likely action and their size indicates its probability. A square indicates the pickup/dropoff action. On the other hand, the intensity and size of the grey circles indicate the option-termination probability. 64

4.5 We compare MSOL against Option Critic (OC), hard options and flat policies trained from scratch or with a pre-trained encoder. For a fair comparison, the soft option prior is identical to the hard option in these experiments. *Left:* Since the options in Option Critic (OC) are not task-agnostic, they fail to generalize to previously unseen tasks. *Middle and right:* Transfer performance of options to environments in which the pickup and dropoff locations where shifted, making the options misspecified. Only soft options provide utility over flat policies in this setting. The middle figure shows results on a small grid in which exploration is simple, whereas the right figure shows that transfer learning can accelerate exploration especially on larger tasks. 69

5.1 We show the loss on the test-data (lower is better). *Left:* Higher ω^f result in a larger difference in generality between features f^c and g^c, making it easier to fit to the more general g^c. *Right:* Learning g^c with fewer datapoints is more challenging, but needed early in training RL agents. 81

5.2 Typical layout of the Multiroom environment. The red triangle denotes the agent and its direction, the green full square is the goal, colored boxes are doors and grey squares are walls. 83

5.3 *Left:* Probability of finding the goal depending on level size for models trained on all levels. Shown are mean and standard error across 30 different seeds. *Right:* Mean and standard error over of the return of the same models averaged across all room sizes. 83

5.4 *Left:* Performance of various agents on the test environments. We note that 'BatchNorm' corresponds to the best performing agent in [145]. Furthermore, 'Dropout-Selective Noise Injection (SNI) ($\lambda = 1$)' is similar to the Dropout implementation used in [145] but was previously not evaluated with weight decay and data augmentation. *Middle:* Difference between test performance and train performance (see Fig. C.4). Without standard deviation for readability. *Right:* Averaged approximate KL-Divergence between rollout policy and updated policy, used as proxy for the variance of the importance weight. Mean and standard deviation are across three random seeds. 85

6.1 Accuracy on CIFAR-10 when the training data is non-stationary over the first 1000 epochs (dashed line). The remaining epochs are trained on the full, unaltered training data. Testing is performed on unaltered data throughout. While final training performance (left) is almost unaffected, test accuracy (right) is significantly reduced by initial, transient non-stationarity. 95

6.2 Evaluation on *Multiroom* and *Sokoban*. Shown are mean and standard error over twelve training seeds. *Left:* Return for Proximal Policy Optimisation (PPO) with and without Iterated Relearning (ITER) on *Multiroom*. Dotted lines indicate when the network was replaced by a new student. *Middle:* Evaluation on layouts with a fixed number of rooms; training is still with a random number of rooms. ITER's advantage is more pronounced for harder levels. *Right:* Return on Sokoban. 100

6.3 Evaluation on *ProcGen*. Dashed lines indicate replacing the teacher. *Left:* Test performance averaged over six environments (*StarPilot*, *Dodgeball*, *Climber*, *Ninja*, *Fruitbot* and *BigFish*). Shown are mean and standard error over all 30 runs (five per environment). Results are normalised by the final test-performance of the PPO baseline on each respective environment to make them comparable. We also compare against the previous state of the art method IBAC-SNI [28]. *Middle:* Evaluation on *Climber*. ITER improves test performance without improving training, supporting our claim that ITER improves the latent representation of the agent. *Right:* Evaluation on *BigFish*. On some environments, ITER improves both train- and test- performance. 102

6.4 *Left:* Ablation studies with sequential ITER and ITER without terms $\mathcal{L}_{\text{PG}}$ and $\mathcal{L}_{\text{TD}}$ (Eq. (6.4)). *Right:* Schematic depiction of training setup for Fig. 6.5 (middle and right). More details are given in Section 6.5.4. $\mathcal{D}$ is the unmodified CIFAR-10 training data while for $\mathcal{D}^{f,m}$ modification $m \in \{\texttt{Noisy Labels}, \texttt{Wrong Labels}, \texttt{Dataset Size}\}$ is applied to the fraction $(1-f)$ of all data-points. In this two phase training setup, we first train on $\mathcal{D}^{f,m}$ during phase 1 and continue on $\mathcal{D}$ during phase 2. A linear predictor parameterised by $(W_i^{f,m}, b_i^{f,m})$ is trained on $\mathcal{D}$ after each phase i, while holding the encoder $\phi_i^{f,m}(x)$ fixed. Evaluation of the resulting classifiers is performed on the original test data. Classifier $i=1$ measures the relevance of the legacy features while classifier $i=2$ measure the final generalisation performance. 103

6.5 *Left:* Test accuracy of students (solid lines) that only learn to mimic the behaviour of poorly generalising teachers in Fig. 6.1 (dashed lines). *Middle:* Final test accuracy of networks trained consecutively on two different datasets. The x-axis shows the accuracy of using encoders trained on the first dataset, retraining only the last layer on the second: nearly useless earlier representations impact future learning much less than slightly sub-optimal ones. Markers indicate modifications to first dataset; colours indicate the fraction of unmodified data points f. Dashed line shows accuracy for $f=1$. *Right:* Singular values of feature matrix Φ, normalised by the largest SV. Solid lines show intermediate values of f with low test accuracy, dashed lines small values of f with higher accuracy. More plots can be found in the appendix. 104

A.1 Training curves on the full set of evaluated Atari games, in the case of flickering and *deterministic* environments. 117

A.2 Training curves on the full set of evaluated Atari games, in the case of flickering and *stochastic* environments. 119
A.3 Reconstructions and predictions using the learned generative model for several Atari games. First column: Current obseration (potentially blank). Second column: Encoded and decoded reconstruction of the current observation. Columns 3 to 7: Predicted observations using the learned generative model for timesteps $dt \in \{0, 1, 2, 3, 10, 30\}$ into the future. 120

B.1 Performance during training phase. Note that MSOL and MSOL(frozen) share the same training as they only differ during testing. Further, note that the highest achievable performance for Taxi and Directional Taxi is higher during training as they can be initialized closer to the final goal (i.e. with the passenger on board). 126
B.2 Results on a 'further modified' taxi environment in which the goal locations at test time were shifted compared to training, making the learned options misspecified, similar to Figs. 4.5b and 4.5c. Here, the goal locations were shifted further, making the options more misspecified. *Left:* Results on a 'small' 8x8 grid. *Right:* Results on a 'large' 10x10 grid. 127

C.1 Generation of the input data x: We embed the information about c twice, once through f^c and once through g^c. See text for details. . 131
C.2 Loss function (error) on test set. Same results as in main text, but for multiple hyperparameters. The qualitative results are stable under a wide range of hyperparameters. 132
C.3 *Left:* Comparison for different implementations of Dropout on the test environments. *Right:* Comparison of Information Bottleneck Actor Critic (IBAC) and Dropout, with and without SNI, *without weight decay and data augmentation.* 134
C.4 Training Performance with weight decay and data augmentation (left) and without (right) 134

D.1 *Left:* Same results as in Fig. 6.5 (middle), but with the fraction of correct data points f on the x-Axis. *Middle:* Multiroom example layout. The red agent needs to reach the green square, avoiding walls (grey) and passing through doors (blocks with coloured outline). *Right:* Sokoban example layout. The green agent needs to push (or pull) yellow boxes on the red targets, avoiding walls. 136
D.2 Individual training curves for the data used in Fig. 6.5(middle) and Fig. D.1. The bottom row shows the same data as the top row, just 'zoomed in'. 137

D.3 All individual results on *ProcGen*. Shown is the mean and standard deviation across two random seeds. 140

List of Abbreviations

A(2)C . . . . . (Advantage) Actor-Critic
AESMC . . . . Autoencoding Sequential Monte Carlo
A(G)I . . . . . Artificial General Intelligence
CNN . . . . . . Convolutional Neural Network
DVRL . . . . . Deep Variational Reinforcement Learning
ELBO . . . . . Evidence Lower Bound
IBAC . . . . . Information Bottleneck Actor Critic
ITER . . . . . Iterated Relearning
KL . . . . Kullback-Leibler Divergence
LSTM . . . . . Long Short Term Memory
MAP . . . . . Maximum A Posteriori
ML . . . . machine learning
MSOL . . . . . Multitask Soft Option Learning
NN . . . . Neural Network
PAI . . . . . . Planning as Inference
(PO)MDP . . (Partially Observable) Markov Decision Process
PPO . . . . . . Proximal Policy Optimisation
RL . . . . Reinforcement Learning
RNN . . . . . . Recurrent Neural Network
SGD . . . . . . Stochastic Gradient Descent
SL . . . . Supervised Learning
SNI . . . . . . Selective Noise Injection
VAE . . . . . . Variational Autoencoder
VIB . . . . . . Variational Information Bottleneck

1
Introduction

Contents

1.1 Generalisation in Machine Learning 1
1.2 Generalisation in Deep Reinforcement Learning 5
1.3 Overview 7

1.1 Generalisation in Machine Learning

Learning is a fascinating mechanism. Based on past experiences, it allows an organism to modify future responses to environmental cues, offering a clear evolutionary advantage [1]. It is therefore unsurprising that such a mechanism can be found in a wide variety of species, including bacteria and plants [2]. In more recent history, even machines have started to learn.

Learning is also closely connected to intelligence, a much sought after ability evoking a large amount of fascination and inquiry. Despite the attention, so far it is offering mostly more questions rather than answers [3]. There is much debate about what intelligence really is, with definitions often revolving around capabilities to *learn*, to *reason* or to *act* in some way. In practise, it is mostly measured by focussing on the last of those aspects as performance is easier to evaluate than

learning or reasoning [4–6]. Widely known measures include the IQ [7] or the g-factor, both of which capture performance across a range of tasks.

Notwithstanding our limited understanding of human intelligence, imbuing machines with it was an early goal in the history of computer science [8]. Motivated by its utility, and likely the fascination it evoked, the field of AI was created alongside the first digital computers.

Avoiding the difficult question of what intelligence is, its general (or 'strong') artificial form is typically defined by referring back to humans [9], as proposed by the famous Turing test [8] which ascribes artificial general intelligence (AGI) to agents able to fool us into thinking they might be human. But even this is hardly a practical objective, so most progress was made in terms of much *narrower* artificial intelligence (AI), by focussing on individual tasks such as classification or reinforcement learning. Early examples include the *Perceptron* [10], a binary classifier, *MENACE* [11], a reinforcement learning (RL) algorithm implemented using match boxes to play Tic-Tac-Toe and the *Nearest Neighbour* algorithm [12] to perform pattern recognition. Conflating these narrow forms of AIs with something more general has, in the past, certainly contributed to the 'AI Effect' that "intelligence is whatever machines haven't done yet" [13].

On the quest towards stronger autonomous agents, a variety of approaches have been tried, including expert systems using only human-specified knowledge bases [14]. However, in many situations, for example for visual input or in RL, the desired outcome is often much easier to specify than the required reasoning, making a learning based approach attractive: compared to describing objects or winning strategies, it is much simpler to label objects, or even just their presence in images, and declare the winner in a game.

Given these practical advantages, it is unsurprising that learning is the predominant approach towards greater automation, especially given the recent explosion in available training data and computing power. While the jury is still out on whether it will eventually allow us to understand, or even re-create human like intelligence, it is already revolutionising many aspects of our life and work, from helping the

visually impaired or optimising complex logistics and infrastructure control to breakthroughs in science, not to mention a vast array of entertainment applications.

If we were concerned with the development of *general* AI, we would need to answer how to bridge the gap from its narrow sibling. However, in this thesis we are content focussing on narrow tasks alone, working towards progress on one of the fundamental challenges associated with learning: generalisation.

The great challenge in learning is not memorisation, computers are excellent at that, but acquiring knowledge in a way that allows transfer to *new*, previously unseen situations. To illustrate the problem, imagine you are given a bag of exotic fruit which you have never seen before. Eating them, you notice that only some are delicious. Next time you are buying this fruit on the market, how will you select them? The difficulty lies in the uniqueness of each fruit: Without any additional information, it can be hard to induce which shared characteristics to pay attention to in order to predict its taste: The colour? The softness? Size? Smell? Age or place of origin? Or time of day you ate it?

Some of these examples will clearly make more sense than others. This is our *prior knowledge*, often also called *inductive bias* in the context of machine learning, that we already bring to the table and which makes any form of generalisation even possible. For us, it is rooted in our previous experiences and genetic makeup. Without this ‘common sense’, we could not know that it is not the number of atoms in the fruit that makes or breaks its taste or that there is any way to predict it at all.

So we need an inductive bias to be able to generalise. Unfortunately, providing machines with human-like ‘common sense’ has so far proven unattainable. What to use in its place has attracted an enormous amount of research and resulted in a variety of proposed techniques. These often either try to find ‘simpler’ solutions (for various definitions of simplicity), or provide additional structure to the space of allowed solutions. An important set of such structures are ‘invariances’, for example the idea that the appearance of an object should be independent of its location in an image [15]. If the chosen structure is suitable for the problem, it can greatly improve generalisation and thereby the performance of our machine because it rules

out many irrelevant hypotheses. Coming back to the exotic fruit, 'simple' might here mean that we are not looking for a combination of features, whereas 'structure' might restrict our attention to visual or tactile senses only.

Equally important to the question of *which* inductive bias we need, is *how much* we need. There is a fundamental trade-off: the more data we can use for learning, the less we need to assume a-priori. Given an entire truck of fruit we could compare them in many more ways than just focussing on e.g. their visual appearance alone. This raises an important question: Should we, as researchers, care about inductive biases and spend time and effort on improving them, or should we rather find ways to use more data in effective ways?

Undeniably, in the long term, successfully and effectively processing more data will provide superior performance as the available computing power and data increases. However, the search for and investigation of inductive biases will nevertheless be insightful and useful. For one, while the available resources are likely to increase exponentially, so might the requirements for data as tasks become more and more complex, limiting what can be achieved in the near future through scaling alone[1]. Furthermore, improving performance on current applications matters on shorter time-scales [19, p. 80], as it helps to inspire, guide and measure progress[2]: for example, the field of machine learning would not exist as it is today without the invention of convolutional neural networks (CNNs) which incorporate spatial invariances into neural networks. Data is always limited and if the imposed structure is close enough to reality, it can greatly improve results. Consequently, it is not surprising that even human intelligence is riddled with cognitive biases we employ to be able to make decisions quickly under incomplete information and limited mental bandwidth [20]. And lastly, we should not forget that *any* method allowing for generalisation will contain inductive biases. We haven't yet reached the limits of what current neural networks and training methods can achieve with more data

[1]While gains in performance for *individual* tasks appear to follow a power law [16, 17], data and compute requirements for scaling task *complexity* might scale exponentially [18], as also evidenced by the recent rise in model sizes and training costs used in state of the art results.

[2]as well as provide funding

[21–23], but we also cannot know whether the biases implicit in current training methods are the most efficient or optimal to the tasks we are trying to solve. For example, in Chapter 6 we will find evidence that the inductive biases intrinsic to training networks by stochastic gradient descent is sub-optimal for many online RL methods on tasks where generalisation is an important factor in agent performance. Ultimately, pushing the boundaries of machine learning will require both more data and compute, but also finding the right inductive biases in our training to achieve the best possible performance with the resources that are given.

1.2 Generalisation in Deep Reinforcement Learning

In this thesis we will focus on the field of deep reinforcement learning (RL): our goal is to train agents that can interact with their environment through a temporal sequence of decisions in order to achieve a goal. We will not tell our agents how to succeed, but will only provide them with an indication of which outcomes are desirable or not. Often, for example when playing a game, this is much easier to do than describing the solution. Furthermore, it also opens the door to finding "superhuman" behaviours.

RL is an exciting field of research. Its formalism allows to capture a wide array of challenging applications and raises many novel questions, for example that of credit assignment, i.e. how to best determine which of the past actions were responsible for a positive or negative outcome. It also seems unlikely that any progress towards AGI would not at least partially incorporate insights from RL, given the temporal nature of the world. But even on shorter time-scales, it holds the promise to revolutionise fields like robotics, finance, logistics and even science.

Our goal in this thesis will be to better understand and improve generalisation in deep RL, i.e. for agents utilising neural networks. Depending on the task at hand, this can improve performance (see e.g. Chapters 3 and 4), improve robustness to unseen situations (Chapter 5) or even reduce the number of environment interactions required for training (Chapter 6).

In the previous section, we have already discussed why investigating generalisation is necessary to minimise the amount of data required for training. This is especially relevant for RL since, as we will discuss, data here is often more costly to acquire than in other fields of machine learning. Furthermore, RL also poses unique challenges for generalisation, thereby requiring specialised tools and approaches.

A key difference of RL is that it is not only concerned with *learning* a solution, but also with *finding* it for the given reward structure. This has two consequences. First, the most powerful RL algorithms usually require the agent to interact with the environment in order to try out various approaches. This means that data cannot easily be re-used, as in other fields, but new data is required for each trained agent. Second, because this data has to be acquired by a not-yet optimal agent, it contains many sub-optimal, and potentially costly decisions. Examples include robots which are easily damaged or wrong ad placements forgoing potential revenue. For safety-critical settings like autonomous driving, this might even rule out such an approach entirely. In research, including this thesis, this is often avoided by utilising only simulated environments, allowing for faster and more affordable experiments. However, for many applications costly real-world data is required, making it crucial to reduce the required amount of training data. Consequently, while simulated data can help us iterate faster, it is paramount to not only focus on this simplified setting, but to find new ways of reducing sample complexity. This holds especially true, since even for simulated data we are reaching the limits of what is economically feasible through scaling up alone, especially in more complex environments [24, 25]. Better generalisation can help to train faster and with less data, bringing down costs and making RL applicable to a wider range of problems. It can also help to improve robustness, another critical requirement in many applications.

Another approach to reduce the amount of costly training data is transfer- or meta-learning: The idea is to re-use prior knowledge, for example acquired from cheaper data sources. This is a promising approach, but at its core, it is also relying on a form of generalisation: if the prior knowledge does not transfer to the target environment, it will offer limited gains.

RL also poses unique challenges for generalisation which require separate inspection. For one, the problem the RL agent is solving is one of (amortised) planning, for which a different set of inductive biases is suitable than for other tasks like natural language processing (NLP) or computer vision (CV). In Chapter 3 and Chapter 4 we will see examples of such inductive biases: belief computation for partially observable environments and hierarchical decision making for transferring knowledge between tasks. Furthermore, the typical training setup in RL introduces additional complexities not present in other fields. In Chapter 5 we will investigate how this complicates the use of stochastic regularisation methods in on-policy algorithms. Lastly, in Chapter 6 we investigate how transient non-stationarities, which are introduced by many deep RL algorithms, lead to sub-optimal generalisation.

Investigating inductive biases for RL might also ultimately provide deeper insights into the nature of intelligence itself: human information processing might have evolved to solve the same underlying, fundamental problem of fast learning and effective knowledge transfer in a large, complex and ever changing world.

1.3 Overview

We begin in Chapter 2 by providing background on some important ideas and formalisms which we will use throughout the thesis. Next, we present the main contributions of this thesis as outlined below. Individual chapters are based on research projects for which I assumed the role of lead author, conducting most or all of the included experiments and taking main responsibility for write-up and research direction. However, all projects were undertaken as team efforts and I am immensely grateful to my co-authors without whom this work would not have been possible.

The thesis is organised in two parts. In Part I we focus on finding inductive biases for deep RL in the form of novel network architectures combined with suitable training methods.

In Chapter 3, which is based on [26], we aim to improve the performance of deep RL agents in partially observable environments, i.e. in situations where not all relevant information is readily available to the agent. We take inspiration from the

field of statistical inference and embed a particle filter into the network architecture. Particle filters are a powerful technique for inferring unobserved information in time-series data of the type encountered in such environments, thereby allowing our agents to learn better policies with higher asymptotic performance.

In Chapter 4, which is based on [27], we equip the agent with the ability of explicitly hierarchical decision making. By separating task-*unspecific* 'lower-level' behaviour from task-*specific* 'higher-level' behaviour, we facilitate easier transfer of lower-level policies to novel tasks. Our main contribution in this project is an improved algorithm of how suitable lower-level behaviours can be found without manual, human specification.

The inductive biases discussed in Part I focus on the problem settings of partial observability and transfer. On the other hand, Part II focusses on regularisation and generalisation in online RL.

In Chapter 5, based on [28], we investigate challenges arising from applying stochastic regularisation to on-policy RL methods and propose an appropriate solution.

Lastly, in Chapter 6, which is based on [29], we investigate the impact of transient-nonstationarity on the training of neural networks. While online RL agents gradually improve their performance and explore the environment, the observed training data distribution is non-stationary, even in stationary environments. We find that this can negatively impact the generalisation performance of neural networks and propose a suitable method to alleviate the effect.

2
Background

Contents

2.1 Probability and information theory 10
2.1.1 Probability theory 10
2.1.2 Bayesian inference 12
2.1.3 Information Theory 14
2.2 Reinforcement learning 15
2.2.1 Markov Decision Processes 16
2.2.2 Solving a Markov Decision Process: Value-function and policy 18
2.2.3 Value function methods 18
2.2.4 Policy gradient methods 21
2.2.5 A note on notation 22
2.3 Function approximation 22
2.3.1 Deep Learning 23
2.3.2 Deep variational Bayes 24
2.3.3 Deep reinforcement learning 25

This chapter serves to introduce notation and provide background information on some of the important ideas which will be used throughout the thesis. More specific background material and references to related work will additionally be given in each chapter.

First, we introduce tools and notation to work with uncertainty in Section 2.1, present key results from reinforcement learning (RL) in Section 2.2 and lastly provide

some background on using neural networks as function approximators in Section 2.3, with a particular focus on how to use them for RL and Bayesian inference.

2.1 Probability and information theory

To start with, we introduce some key concepts and notations relating to uncertainty. We first introduce probability theory as a means to describe and work with uncertainty. Second, we introduce information theory as a means to quantify the amount of uncertainty in a probability distribution and, conversely, how much information we gain when we observe new information.

2.1.1 Probability theory

Probability theory allows us to express uncertainty over which value, also called *outcome*, a random variable X might assume. In the case of a finite number of possible values x, this uncertainty is described in the form of *probabilities* $P(X = x)$, taking on a value between 0, expressing impossibility of x, and 1, expressing certainty that $X = x$. We call P a *distribution* over X if it provides us with a probability for each possible value x, the sum of which must be 1 as outcomes are mutually exclusive. We will denote the *probability mass function* for the *event* $X = x$ by $p(x) = P(X = x)$.

If there is an uncountable infinite number of possible values of x, for example for continuous $x \in \mathbb{R}$, the probability for each individual outcome is 0. Instead, we rely on a *probability density function* $p(x)$ which we can integrate over *events* A to recover finite probability values:

$$P(X \in A) = \int_{x \in A} p(x) dx. \tag{2.1}$$

The probability for the particular event that $X \leq x$ is expressed by the *cumulative distribution function*

$$P(X \leq x) = \int_{-\infty}^{x} p(x) dx. \tag{2.2}$$

We can also relate two different events through *conditional probabilities* $P(A|B)$ denoting the probability for A given that B has occurred. Alternatively, we can view $P(A|B)$ as the *likelihood* of B, given A has occurred. While A and B can relate to the same random variable, we will mostly use conditional probabilities to reason about the probability distribution of one variable X, given a realized value of another variable Y, i.e. $P(X = x|Y = y)$. The two random variables X and Y are called independent if $P(X = x|Y = y) = P(X = x)$. Conditional densities are defined similarly.

A very useful relation is *Bayes' rule*, which allows us to 'swap' the direction of conditioning:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.3}$$

This follows directly from the product rule $P(A, B) = P(A)P(B|A) = P(B)P(A|B)$, where $P(A, B)$ denotes the *joint* probability that both A and B occur. When working with multiple random variables, one can also compute the *marginal* $P(X)$ of a joint distribution $P(X, Y)$ by integration, i.e. $p(x) = \int p(x, y)dy$, or summation for discrete y.

Lastly, we introduce the notation for computing an *expectation* $\mathbb{E}[f(X)]$ of a function f, which averages out the uncertainty over X:

$$\mathbb{E}[f(X)] = \int f(x)p(x)dx \tag{2.4}$$

If f is the identify function, we refer to it as the *expected value* or *mean* $\mu = \mathbb{E}[X]$. Sometimes we will denote the sampling distribution as subscript $\mathbb{E}_{p(x)}[X]$, but if it is clear from context, we will omit it for a less cluttered notation. If one or more random variables are *not* averaged over, this is explicitly stated as *conditional expectation* $\mathbb{E}[X|Y = y]$. The variance is defined as $\mathrm{Var}(X) = \mathbb{E}\left[(X - \mu)^2\right]$ and its squared root is the standard deviation $\sigma = \sqrt{\mathrm{Var}(X)}$.

2.1.2 Bayesian inference

Often we have access to some observed data $\mathcal{D} = \{X_1, \dots, X_n\}$ consisting of n datapoints and are interested in *inferring* some unobserved *latent* variable $\{Z_1, \dots, Z_n\}$ which influenced the data generation process. For example, in healthcare we generally do not have direct access to the true causes of a patient's ailments (here Z), but do observe the symptoms X. Similarly, as we will explore in Chapter 4, imbuing agents with an internal state requires us to infer it, since we only have access to its observed real-world actions.

Bayesian inference is one approach of doing so, by *inverting* the generative process, expressed by the model $P(X|Z)$, to compute the *posterior distribution* $P(Z|X)$. Making use of Eq. (2.3) we immediately get

$$P(Z|X) = \frac{P(X|Z)P(Z)}{P(X)}. \tag{2.5}$$

Here, $P(Z)$, denotes our *prior* belief over Z, for example, which illnesses are more likely a-priori, before examining the patient. The *likelihood*, as defined by our model, tells us how likely it is that the observed data X was in fact generated for a certain value of Z and the *marginal likelihood* $P(X)$ serves as normalisation constant and only depends on the chosen model.

While this process appears simple, it can be exceedingly difficult. On the one hand, computing $P(X)$ might be computationally infeasible, especially for high-dimensional Z. Furthermore, while knowledge of $P(X)$ allows *evaluation* of $P(Z|X)$, for many purposes we would like to *sample* values from it, which requires, for example for continuous Z, the inverse of the cumulative distribution function, which we might not be able to write down analytically.

An enormous body of literature exists to tackle those problems. Here, we only want to give a very brief introduction for two types of solution strategies whose machinery will be used in some of the following chapters.

Monte Carlo

The idea of *Monte Carlo* is to approximate a probability distribution by randomly drawn samples. Most often, this is used for *Monte Carlo integration*, i.e. to estimate an expectation such as $I = \mathbb{E}[f(X)] = \int f(x)p(x)dx$ by the estimator

$$I_N := \frac{1}{N}\sum_{n=1}^{N} f(\hat{x}_n) \quad \text{where} \quad \hat{x}_n \sim p(x) \tag{2.6}$$

Note that the estimator is *unbiased* due to the linearity of the expectation. This idea is not just used in Bayesian inference, but also more widely in machine learning (ML), for example to compute approximate gradients, or in RL, for example to estimate the value function through 'Monte Carlo rollouts'.

This approach is underpinned by the *law of large numbers*, which states that, informally, the more samples N we use, the closer the empirical average will be to the true expected value if the samples are independent and identically distributed (i.i.d.). In other words, it tells us that the Monte Carlo estimator is *consistent*.

We can also ask how quickly we are converging, i.e. how $\text{Var}(I_N)$ changes with N:

$$\text{Var}(I_N) = \text{Var}\left(\frac{1}{N}\sum_{n=1}^{N} f(X_n)\right) = \sum_{n=1}^{N} \text{Var}\left(\frac{f(X_n)}{N}\right) = \frac{\text{Var}(f(X))}{N} \tag{2.7}$$

where we used the i.i.d assumption to replace $\text{Var}(f(X_n))$ with $\text{Var}(f(X))$ and to ignore the covariance terms when moving the sum out of the variance. Consequently, as we increase N, the standard deviation shrinks at a rate of $1/\sqrt{N}$.

Variational Bayes

An alternative to approximating the posterior distribution with samples is the *variational* approach. Here, we introduce *variational* distributions, for example one for each of the n data-points $Q_n(Z_n|\phi)$ which are parametrised by ϕ. As we will see in Section 2.3.2, when using neural networks, we often learn only *one* neural network, which conditions on the data-point as additional input, i.e. $Q(Z|X, \phi)$. Our aim is to minimize some distance function between $Q(Z|X, \phi)$ and the true posterior $P(Z|X)$.

Often the Kullback-Leibler divergence (KL) is chosen as distance as it allows formulating the evidence lower bound (ELBO):

$$\begin{aligned}
\mathbb{D}_{\text{KL}}(Q(Z|X,\phi) \parallel P(Z|X)) &= \mathbb{E}_Q\left[\log Q(Z|X,\phi)\right] - \mathbb{E}_Q\left[\log p(Z|X)\right] \\
&= \mathbb{E}_Q\left[\log Q(Z|X,\phi)\right] - \mathbb{E}_Q\left[\log P(Z,X)\right] + \log P(X) \\
&= -\,\text{ELBO} + \log P(X)
\end{aligned} \tag{2.8}$$

Reordering the terms, we see that $\text{ELBO} = \log P(X) + \mathbb{D}_{\text{KL}}(Q(Z|X,\phi) \parallel P(Z|X))) \leq \log P(X)$ as $\mathbb{D}_{\text{KL}} \geq 0$, thereby motivating it's name as a lower bound on the evidence, as the marginal likelihood $P(X)$ is often called. Importantly, while we typically cannot compute or optimize $\mathbb{D}_{\text{KL}}(Q(Z|X,\phi) \parallel P(Z|X))$ directly as we don't have access to $P(Z|X)$, we often *can* optimize $\text{ELBO} = \mathbb{E}_Q\left[\log \frac{P(Z,X)}{Q(Z|X,\phi)}\right]$ under some additional assumptions, for example a particular factorization of $Q(Z|X,\phi)$ and by employing an iterative procedure. Furthermore, because $\log P(X)$ does not depend on ϕ, maximising the ELBO has the desired effect of reducing the distance between $Q(Z|X,\phi)$ and $P(Z|X)$.

So far, we have discussed the variational approach to approximate posterior distributions over latents Z for a given model $P_\theta(X,Z) = P_\theta(X|Z)P_\theta(Z)$ with fixed θ. By introducing additional 'higher-level' hyper-parameters, we could also treat θ as an unobserved latent and include it into the variational approach. Alternatively, we can optimise a single value of θ using the maxiumum a posteriori or maximum likelihood estimate and alternate between optimizing the ELBO with respect to ϕ and θ.

2.1.3 Information Theory

Many information-theoretic quantities are often used throughout ML, including this thesis. In this section, our aim is to briefly introduce the most important ones.

The core intuition behind information theory is that random events that occur rarely, contain more information. To justify this intuition, note that receiving confirmation of an event which is guaranteed to happen will not be very 'informative'. On the other hand, learning that something very unexpected happened provides much more information.

To also make information additive for independent events, it is defined as $I(x) = -\log P(x)$ for event x. When using the natural logarithm, it is measured in *nats*, whereas the use of a base-2 logarithm leads to *bits*.

The average information provided by a random event is the *entropy* $H(X) = \mathbb{E}_{P(X)}[-\log P(X)]$. It is maximal for a uniform distribution and 0 for deterministic events. When X is continuous, it is also called *differential entropy*. The *conditional entropy* is similarly defined as $H(X|Y) = \mathbb{E}_{P(Y)}[H(X|Y)] = -\sum_{x,y} p(x,y) \log p(x|y)$. Note that unlike the conditional expectation, here we also average over the conditioning variable.

A natural question to ask next is how much we learn about X by conditioning on Y. This is expressed by the *mutual information* $I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$. Note how it measures the reduction in entropy through conditioning. It can also be written as

$$I(X,Y) = \mathbb{E}_{P(X,Y)}\left[\log \frac{p(x,y)}{p(x)p(y)}\right]. \tag{2.9}$$

Lastly, we introduce the Kullback-Leibler divergence between two probability distributions

$$\mathbb{D}_{\mathrm{KL}}(P(x) \parallel Q(x)) = \sum_x p(x) \log \frac{p(x)}{q(x)}. \tag{2.10}$$

There are various interpretations of this quantity. For example, it measures the amount of additional encoding costs if Q is assumed as generating distribution instead of the true distribution P. Similarly, it also measures the additional 'surprise' in a Bayesian setting if Q is used as prior instead of P. As we have seen in Section 2.1.2 it is also often used as distance between two distributions. It is to note, however, that it is not symmetric and therefore not a metric.

2.2 Reinforcement learning

In the following we provide an overview over how the learning problem in RL is formalized, usually as Markov decision process (MDP) or partially observable Markov decision process (POMDP), and how it can be subsequently solved by learning an optimal state(-action) value function or policy. For more details, we refer to [30].

2.2.1 Markov Decision Processes

In RL our goal is optimize the behaviour of one *agent* within an *environment*. Most commonly, this is formalized as a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, P, P_0, r, \gamma)$: In a sequence of discrete time-steps t, the agent receives the current *state* of the environment, $s_t \in \mathcal{S}$, and has to choose an *action* $a_t \in \mathcal{A}$ with which to respond. Here, $\mathcal{A}$ is the space of actions available to the agent and $\mathcal{S}$ is the state-space, i.e. the set of all possible states of the environment. We represent the agent by a *policy* $\pi(A_t|S_t) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, expressing the probability to choose each action in a given state.

Upon taking an action, the environment transitions to a new state s_{t+1}. This transition can be stochastic and is expressed by the distribution $P(S_{t+1}|S_t, A_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. Importantly, because we assume the state s_t to be *Markov*, the transition only conditions on the current state and not on any earlier states $s_{<t}$.

In this work, we will always assume an *episodic* setting. Consequently, at $t = 0$, the initial state will be drawn randomly as $s_0 \sim P_0(S_0)$ where $P_0 : \mathcal{S} \rightarrow [0, 1]$ is the initial state distribution. Furthermore, after either a fixed *horizon* T or when the agent reaches a *terminal* state, the current interaction with the environment, i.e. the *episode*, ends.

So far, we have described the dynamics of the agent-environment interaction.

$$P(S_{\leq T+1}, A_{\leq T}) = P_0(S_0)\pi(A_0|S_0) \prod_{t=1}^{T} P(S_t|S_{t-1}, A_{t-1})\pi(S_t|A_t) \tag{2.11}$$

Ultimately, our goal will be to find an optimal policy

$$\pi^\star(A_t|S_t) = \arg\max_\pi \mathbb{E}_{P(\tau)}\left[R(\tau)\right] \tag{2.12}$$

maximizing the expected future discounted *return* $R(\tau) = \sum_{t=0} \gamma^t r_t(s_t, a_t, s_{t+1})$ over *trajectories* $\tau = (s_0, a_0, r_0, \ldots, s_{T+1})$.

Here, the reward $r_t : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a scalar value which the agent receives from the environment at each step together with the current state. The reward function is often specified by a human, rewarding desired outcomes with a high, and penalizing undesired outcomes with a low or negative value.

Sometimes, we care more about immediate rewards than those far into the future. This can be expressed using the *discount factor* $\gamma \in [0, 1]$ (or $\gamma \in [0, 1)$ for non-episodic settings) where smaller values assign a relatively higher weight to more immediate rewards.

For simplicity of exposition, we so far assumed $\mathcal{S}$ and $\mathcal{A}$ to be discrete. However, by replacing the probabilities with corresponding densities, the above definitions can without problem be extend to continuous state- and action spaces, which we will also encounter throughout this work.

Partial observability

One strong assumption made by the MDP is that the agent observes the *full* state every time it has to make a decision. Unfortunately, this is not the case in many important applications. To accommodate this, we can extend the formalism to partially observable Markov decision processes (POMDPs) which are represented by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, O, P_0, r, \gamma)$. This adds an observation-space $\mathcal{O}$ and observation distribution $O(s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$ to the definition of an MDP.

The agent now only has access to the *observations* $o_t \in \mathcal{O}$, which often correspond to a noisy or partially occluded version of the state s_t. Importantly, these are not Markov any more. Consequently, to take optimal actions, the agent must now condition on the entire history $(o_{\leq t}, a_{<t})$ where we use $o_{\leq t}$ to denote $(o_0, \dots, o_t)$ and $a_{<t}$ to denote $(a_0, \dots, a_{t-1})$.

Unfortunately, the dimensionality of the history grows exponentially in t. An alternative to conditioning on it is to instead perform Bayesian inference (see Section 2.1.2) to compute the *belief state* $b_t := p(S_t | o_{\leq t}, a_{<t})$ which is a sufficient statistic of the history. In other words, if we know b_t we cannot learn anything more about s_t, and thereby the optimal action, when additionally conditioning on $(o_{\leq t}, a_{<t})$. This allows us to condition our policy $\pi^\star(A_t | b_t)$ on b_t. Furthermore, the belief state can be computed recursively as

$$b_{t+1} = \frac{\int b_t O(o_{t+1} | s_{t+1}, a_t) P(s_{t+1} | s_t, a_t) \mathrm{d}s_t}{\int \int b_t O(o_{t+1} | s_{t+1}, a_t) P(s_{t+1} | s_t, a_t) \, \mathrm{d}s_t \, \mathrm{d}s_{t+1}}. \tag{2.13}$$

However, as discussed in Section 2.1.2, the integrations involved in its computation can be infeasible, thereby requiring approximation. Furthermore we now also need knowledge of the transition and observation probabilities P and O. In chapter Chapter 3 we will explore how those can be learned simultaneously with the policy.

2.2.2 Solving a Markov Decision Process: Value-function and policy

For simplicity, this section assumes an MDP, however the definitions can be extended to POMDPs by replacing the state s_t with either the history or belief state b_t.

We have already introduced the optimisation objective $J^\pi = \mathbb{E}_{P(\tau)}[R(\tau)]$ as the average expected reward under both policy- and environment stochasticity. In practise, it will be helpful to also introduce the *value function* $V^\pi(s) = \mathbb{E}_{P(\tau)}[R(\tau)|S_0 = s]$ and *state-action value function* $Q^\pi(s,a) = \mathbb{E}_{P(\tau)}[R(\tau)|S_0 = s, A_0 = a]$, which express the expected future discounted reward conditioned on starting in a certain state s and taking action a. We will also often use the *advantage* $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

To find the optimal policy $\pi^\star$, this presents two approaches: Either, we try to optimise the policy directly by following the *policy gradient* $\nabla_\pi J^\pi$, or we try to find the optimal action value function $Q^\star(s,a) = Q^{\pi^\star}(s,a)$ and use $\pi^\star(a|s) = \arg\max_a Q^\star(s,a)$ as our optimal policy.

One important restriction on the desired algorithm is that we only assume to be able to sample entire trajectories from the environment. We cannot sample individual states or transitions and neither know the distribution $P(S_{t+1}|S_t, A_t)$.

2.2.3 Value function methods

We start by discussion several 'building blocks' which can be combined in a number of ways. There are two main problems we have to tackle: i) How do we find the value functions V^π and Q^π for a given policy π and ii) how can we improve π, once we know its value function?

The thus discussed building blocks can be combined in a number of ways, giving rise to variously named algorithms, of which we will briefly discuss some of the most famous ones.

The prediction problem: Finding the value function We start with the first problem: How to find $V^\pi(s)$ and $Q^\pi(s,a)$? To evaluate $V^\pi(s) = \mathbb{E}_{P(\tau)}[R(\tau)|S_0 = s]$, we could just 'try out' the observed future return $R(\tau)$ by starting in s and following π. For $Q^\pi(s,a)$ we start in s and take a first, independently of π, and follow π thereafter. If either the policy or environment is stochastic, we would need to roll out many such trajectories and average them. However, we usually can't randomly initialise our environment in an arbitrary state s. As an alternative, we can start in states sampled from $s_0 \sim P_0(S_0)$ and make sure our policy π has a non-zero probability for every possible action, thereby making sure we eventually pass through any state reachable from our initial states.

That is the idea behind the Monte Carlo approach and the thereby estimated returns we often call *Monte Carlo returns*. Unfortunately, especially for long episodes with a stochastic policy and environment, we might require a very large number of such rollouts to obtain a reliable low-variance estimate: as we've seen in Section 2.1.2, the standard deviation of the resulting estimator only decreases with $1/\sqrt{N}$ for N such rollouts. So the question arises: Can we do better?

In fact, we can, by recognising that there exists a recursive relationship, called the *Bellman equation*, which should hold for value functions:

$$V^\pi(s) = \mathbb{E}_{\pi,P}\left[r(S,A,S') + \gamma V^\pi(S') \,\middle|\, S = s\right] \tag{2.14}$$

$$Q^\pi(s,a) = \mathbb{E}_{P}\left[r(S,A,S') + \gamma V^\pi(S') \,\middle|\, S = s, A = a\right] \tag{2.15}$$

So, instead of getting one Monte Carlo sample for $V^\pi(s)$ by summing the discounted rewards until the end of the episode, we can 'cut it short' after just one step and take what we have learned so far for $V^\pi(s')$ as an estimate for the remainder. Confusingly for statisticians, we call this process *bootstrapping* (for them it means something completely different) and the resulting method is part of the family

of *temporal difference* methods. These types of methods offer us a bias-variance trade-off: Because $V(s')$ might have already been averaged over multiple samples, it will have lower variance. But because our estimate of it might be wrong (for example because we use function approximation to represent $V(s')$, as we will do in section Section 2.3.3), we will introduce bias into our estimation. Instead of bootstrapping after just one step, we can also take n steps as many state-of-the-art algorithms do, reducing the bias, but increasing the variance compared to the one-step version.

If we *had* access to the model and could perform such an update step in expectation across all states simultaneously, it would have nice convergence properties because the update operator is a contraction.

Improving the policy Once we've determined $Q^\pi(s,a)$ improving our policy π is straightforward: We construct our new policy $\pi'(a|s) = \arg\max_a Q^\pi(s,a)$, i.e. in each state, we greedily take the action that would improve our future expected return, even if we were to 'just' follow π afterwards. If that means that now for any state $Q^\pi(s,\pi'(s)) > V^\pi(s)$, we have improved our policy. If this is not the case, then our policy is now optimal and we have for all states the optimal value functions $V^\star(s)$, $Q^\star(s,a)$

$$V^\star(s) = \max_a Q^\star(s,a) = \max_a \mathbb{E}_P \left[r(S,A,S') + \gamma V^\star(S') \,\middle|\, S=s, A=a \right] \tag{2.16}$$

which is also called the *Bellman optimality equation.*

The above ideas can be combined in a number of ways. For example, if we assume access to the true model, we can turn Eq. (2.14) into an iterative update rule across the entire state space. Then, alternating between updating $V^\pi(s)$ until convergence and improving the policy is called *policy iteration*, whereas only performing one update step on $V^\pi(s)$ before improving the policy would be called *value iteration.* On the other hand, iterating the bellman *optimality* equations using experience sampled from the environment results in the famous Q-learning algorithm.

Next, we will look at an alternative family of algorithms which tries to optimise the policy directly, called policy gradient algorithms.

2.2.4 Policy gradient methods

The idea of policy gradient methods is to estimate the gradient $\nabla_\theta J^\pi$ with respect to the parameters θ that now parametrise the policy $\pi_\theta(a|s)$. Typically, we cannot compute this gradient directly as it would require differentiation with respect to the environment dynamics, which are unknown to us. Instead, we make use of a cool trick with many names, but typically referred to as the REINFORCE estimator [31] within the RL community. Writing out the gradient, we can see that

$$\begin{aligned}\nabla J^\pi &= \nabla \mathbb{E}_{P(\tau)}\left[R(\tau)\right] \\ &= \sum_t \mathbb{E}_{P(\tau\setminus A_t)}\left[\int_{a_t} \nabla_\theta \pi_\theta(a_t|S_t) R(\tau) da_t\right] \\ &= \sum_t \mathbb{E}_{P(\tau\setminus A_t)}\left[\int_{a_t} \pi_\theta(a_t|S_t) \frac{\nabla_\theta \pi_\theta(a_t|S_t)}{\pi_\theta(a_t|S_t)} R(\tau) da_t\right] \\ &= \sum_t \mathbb{E}_{P(\tau\setminus A_t)}\left[\int_{a_t} \pi_\theta(a_t|S_t) \nabla_\theta \log \pi_\theta(a_t|S_t) R(\tau) da_t\right] \\ &= \sum_t \mathbb{E}_{P(\tau)}\left[\nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau)\right]\end{aligned} \tag{2.17}$$

where we denote by $\mathbb{E}_{P(\tau\setminus A_t)}[\cdot]$ the expectation over all variables except A_t. The second line follows from the product rule for derivatives, and we use $\frac{\nabla_x f(x)}{f(x)} = \nabla_x \log f(x)$ in the fourth line.

A number of variance reduction techniques can be applied to Eq. (2.17). The *policy gradient theorem* [32] tells us that in each state, we only need to take future rewards into account, i.e. we can replace $R(\tau)$ with $R_t(\tau_{>t})$ or its expectation $Q^\pi(s_t, a_t)$. Furthermore, the use of a *baseline*, typically $V^\pi(s_t)$ does not introduce bias and results in the following estimator

$$\nabla J^\pi = \sum_t \mathbb{E}_{P(\tau)}\left[\nabla_\theta \log \pi_\theta(A_t|S_t) A^\pi(S_t, A_t)\right] \tag{2.18}$$

with the previously introduced advantage $A^\pi(s_t, a_t)$. Interestingly, the TD-error used to update the action-value function is also an unbiased estimate for the advantage, e.g. for the one-step case $A^\pi(s_t, a_t) = \mathbb{E}_T\left[r(S_t, A_t, S_{t+1}) + \gamma V(S_{t+1}) - V(S_t) \mid S_t = s_t, A_t = a_t\right]$

This family of policy gradients algorithms is often referred to as *actor-critic* approaches as they estimate both a policy, as well as a value function, which is referred to as critic.

2.2.5 A note on notation

When describing algorithmic contributions we will often drop the distinction between upper and lower case notation for random variables X and their values x as this would likely lead to confusion. For example, while the value function $V(s)$ is defined as an expectation *conditioned* on a particular starting state s, we will often want to average over all possible values of S to define a loss function for $V(S)$. Furthermore, while we generally define loss functions as expectation over the trajectory distribution $p(\tau) = p(S_{\leq T}, A_{<T})$, these will in practise always be approximated by a collected batch of data $\mathcal{D}_\tau = \{s^{(i)}_{\leq T}, a^{(i)}_{<T}\}_i$. As these distinctions are irrelevant for the contributions of this thesis, we will mostly denote all variables and their values in lower case notation. This will also allow us to use the common notation of A_t or A as the shorthand notation for the advantage $A^\pi(s_t, a_t)$ and a_t as the corresponding action. We will keep the distinction were appropriate, for example for definitions of generative models or information-theoretic quantities.

2.3 Function approximation

So far, we have treated states s and actions a as discrete, implying that $|\mathcal{S}|$ and $|\mathcal{A}|$ are small enough such that we can visit all (relevant) possible combinations in $\mathcal{S} \times \mathcal{A}$ often enough in the environment to learn, and store, quantities like action-probabilities or value functions on this joint domain. In many environments, for example when states are visual inputs or of continuous value, this is not the case. As a result, we employ *function approximation* using parametric models to approximate such quantities of interest. This allows us to *generalise* between states, relieving us from having to visit every possible state during training while still promising optimal behaviour, at least most of the time.

In this work, we will exclusively use deep neural networks (DNNs) for function approximation. While many alternatives have been explored, the use of deep neural networks has shown exceptional promise in allowing RL to scale up to many relevant problem domains.

In Section 2.3.1 we will provide some background on working with neural networks. Next, we will introduce how they can be applied within a Bayesian framework (Section 2.3.2) and lastly how they can be used in RL (Section 2.3.3).

2.3.1 Deep Learning

A DNN consists of a sequence of differentiable functions, often called 'layers' if they contain trainable parameters. The most widely used types of layers, i.e. fully connected and convolutional layers, use parametrised affine transformations. Alternating such layers with activation functions such as sigmoids or ReLUs [33, 34], allows the DNN to also approximate non-linear relationships.

In principle, for sufficient 'width' (dimensionality of intermediate representations) or 'depth' (number of such layers) such DNNs can approximate any smooth function arbitrarily well. In practise, incorporating additional structure, often called *inductive bias*, into the network architecture can lead to improved performance as it allows the combination of the high flexibility of DNNs with task-specific prior information.

In general, any differentiable function can be used and a large body of research exists to find ones which empirically provide better results on certain tasks. On a high level, Chapters 3 to 5 follow this idea to develop architectures more suitable for POMDPs (Chapter 3), skill transfer (Chapter 4) and generalisation in RL (Chapter 5).

DNNs are usually trained using stochastic gradient descent (SGD) or a modification thereof: On a subset of all available data points (the *batch*), the gradient of the network's parameters w.r.t. a scalar loss function is computed and the parameters are adjusted accordingly. The exact size and direction of each parameter's 'step' can be computed in a variety of ways, often utilising additional statistics of previous adjustments [e.g. 35]. A key insight for the efficient computation of the parameters' gradients was the development of the *backpropagation* algorithm [36], allowing the computational complexity to scale linearly with the number of parameters.

In the following we will briefly describe important ideas of how DNNs have been applied to two areas: Bayesian inference in latent variable models and RL.

2.3.2 Deep variational Bayes

In Section 2.1.2 we introduced the idea of computing an approximate posterior distribution $Q(Z|X, \phi)$ for data X in a model $P_\theta(Z)P_\theta(X|Z)$ using variational inference.

As it turns out, this idea is extremely useful when we want to train DNNs with *stochastic* latent variables. Such stochastic latents might be desirable for a variety of reasons, for example because we want to incorporate an inductive bias into our model (as we do in Chapter 3) or for regularisation (as we explore in Chapter 5).

As in variational inference, our goal is to maximise the ELBO both w.r.t ϕ and θ. Two key ideas [37] are important to do so efficiently with neural networks: the *reparameterisation trick* and the idea of *amortised inference*.

In classical variational inference, we usually optimise a separate approximate posterior $Q_n(Z|\phi)$ for each data-point n. However, this will not generalise to new data points. Instead, in deep learning, we represent Q through a neural network which takes the data-point X as additional input. We call this amortised inference, as $Q(Z|X, \phi)$ now provides us with a posterior distribution, even for new data points, without requiring any additional inference procedures.

The ELBO is an expectation over Q. Consequently, we require gradients w.r.t this expectation over Q to optimise ϕ. As we have seen in Section 2.2.4, this is possible using the log-derivative trick, which, however, results in a high-variance gradient estimator.

Instead, Kingma and Welling [37] found that for certain probability distributions, most notably the normal distribution, we can instead write $\mathbb{E}_{P(Z|X,\phi)}[f(z)] = \mathbb{E}_{P(\epsilon)}[f(g(x, \epsilon, \phi))]$ where $z = g(x, \epsilon, \phi)$ is a differentiable function. Conveniently, this now allows us to express the gradient of the expectation as the expectation of a gradient $\nabla_\phi \mathbb{E}_{P(Z|X,\phi)}[f(z)] = \mathbb{E}_{P(\epsilon)}\left[\nabla_\phi f(g(x, \epsilon, \phi))\right]$, resulting in a gradient estimator with much lower variance.

The most widely known such architecture is the variational autoencoder (VAE) [37] which makes the additional assumption that the prior over Z is a multivariate normal distribution with zero mean and diagonal unit variance $P(Z) =$

$\mathcal{N}(0, I)$. Both the *encoder* (also called *proposal distribution*) $Q(Z|X, \phi)$ and *decoder* $P(X|Z, \theta)$ are neural networks, optimised jointly by maximising the ELBO

$$\text{ELBO}(\theta, \phi, x) = \mathbb{E}_{Q(Z|x,\phi)}\left[\log \frac{p(x|Z,\theta)p(Z)}{q(Z|x,\phi)}\right] \tag{2.19}$$

over some dataset $x \in \mathcal{D}$.

2.3.3 Deep reinforcement learning

In Section 2.2 we provided some background on approaches to solve the RL problem of finding an optimal policy. Here, we discuss how neural networks can be trained to approximate the policy $\pi_\theta(a|s)$ and value function $V_\theta(s)$. We focus on policy gradient methods, as those will be used throughout this work. The main goal of the algorithmic modifications presented here is to stabilise training, since using DNNs as function approximators in RL usually violates assumptions required for guaranteed convergence. In practise, such modifications have been found to improve stability as well as asymptotic performance and are very widely used.

The advantage $A(s, a)$ is an important quality to estimate as it often is used both as TD-error estimate to train the value function, as well as in the actor-critic gradient estimation Eq. (2.18).

Instead of the one-step approximation introduced in Section 2.2.4, in practise we often use a batched n-step version. Starting at timestep t, we collect data from n_s steps in the environment. Instead of using an n-step target estimator for each of those n_s values $V_\theta(s_{t+i})$, $i = 0, \ldots, n_s - 1$, we use $V_\theta(s_{t+n_s})$ as bootstrap value for all, resulting in varying unroll lengths depending on the location i in the batch:

$$A(s_{t+i}, a_{t+i}) := \left(\sum_{j=0}^{n_s-i-1} \gamma^j r_{t+i+j}\right) + \gamma^{n_s-i} V_\theta(s_{t+n_s}) - V_\theta(s_{t+i}) \tag{2.20}$$

Further performance gains can sometimes be achieved by also using the intermediate value functions $V_\theta(s_{t+i})$ as bootstrap values and mixing all resulting estimators [38]. In practise, this estimation is performed over data from n_e environments which are often simulated in parallel. In this work, we are following this setup since it is

widely used and allows for faster data collection and less correlated samples in each batch. It is, however, important to note that consequently additional challenges have to be overcome in applying our results to applications in which these parallel executions of multiple environments are not possible.

To optimise the policy, we will in most of this work use PPO which optimises the following surrogate loss [39]:

$$L_{\text{PPO}} = -\mathbb{E}_{\mathcal{D}_\tau}\left[\min(c_t(\theta)A_t, \text{clip}(c_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right] \tag{2.21}$$

with $c_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi^r(a_t|s_t)}$ and $A_t = A(s_t, a_t)$. Here, $\mathcal{D}_\tau$ is a batch of data collected using the *rollout policy* $\pi^r(a_t|s_t)$. While we use our current policy π_θ to collect data, in practise we perform multiple gradient updates on the same dataset $\mathcal{D}_\tau$, thereby moving π_θ away from π^r, resulting in the required *importance weight* $c_t(\theta)$. The hyper-parameter ϵ is a small positive constant. The above objective is an efficient approximation to trust region methods [40], optimising a pessimistic lower bound of the objective function on the collected data. It corresponds to estimating the gradient w.r.t the policy conservatively, since moving π_θ further away from π^r_θ, such that $c_t(\theta)$ moves outside a chosen range $[1-\epsilon, 1+\epsilon]$, is only taken into account if it decreases performance.

The overall minimisation objective is then

$$L_t(\theta) = L_{\text{PPO}} + \lambda_V \mathbb{E}_{\mathcal{D}_\tau}\left[A^2\right] - \lambda_H \mathbb{E}_{\mathcal{D}_\tau}\left[H[\pi_\theta]\right] \tag{2.22}$$

where $H[\cdot]$ denotes an entropy bonus to encourage exploration and prevent the policy from becoming too deterministic prematurely.

Part I

Inductive Biases for Reinforcement Learning

Abstract

In environments with large or continuous state spaces, good generalisation is a requirement for well performing agents as not every possible state can be experienced during training. And while deep neural networks are highly flexible and can be applied to an extremely wide range of problems to provide the required generalisation, one can often improve their performance by incorporating more task-specific structure into the network. In this part we explore such structures which are focussed on specific, yet widely applicable problem domains within RL.

First, in Chapter 3 we focus on the setting of partially observable environments (POMDPs), in which the agent cannot observe all relevant information at each moment in time. Typically, RNNs and more recently Transformers [41] are used in such situations to aggregate the past to try to infer the required information. However, depending on the environment, the thereby required computation might be highly complex, namely similar to performing statistical inference. To aid learning such complex computation, we embed a suitable structure, a particle filter [42], into the architecture of the network and show how this is able to improve performance.

Second, in Chapter 4 we turn our attention to hierarchical RL. Separating 'higher-level' (for example *what* to do), from 'lower-level' (e.g. *how* to do it) decision making promises to facilitate easier reuse of such policies by allowing novel recombinations of the individual steps. However, while for humans it may often seem natural to define clear boundaries between individual lower-level sub-tasks, how to learn these optimally remains an open questions. In this chapter, we propose to extract suitable sub-task boundaries by aiming to find lower-level policies which are easily re-usable across multiple tasks. We also propose a novel algorithm which overcomes several difficulties in optimising this objective.

3

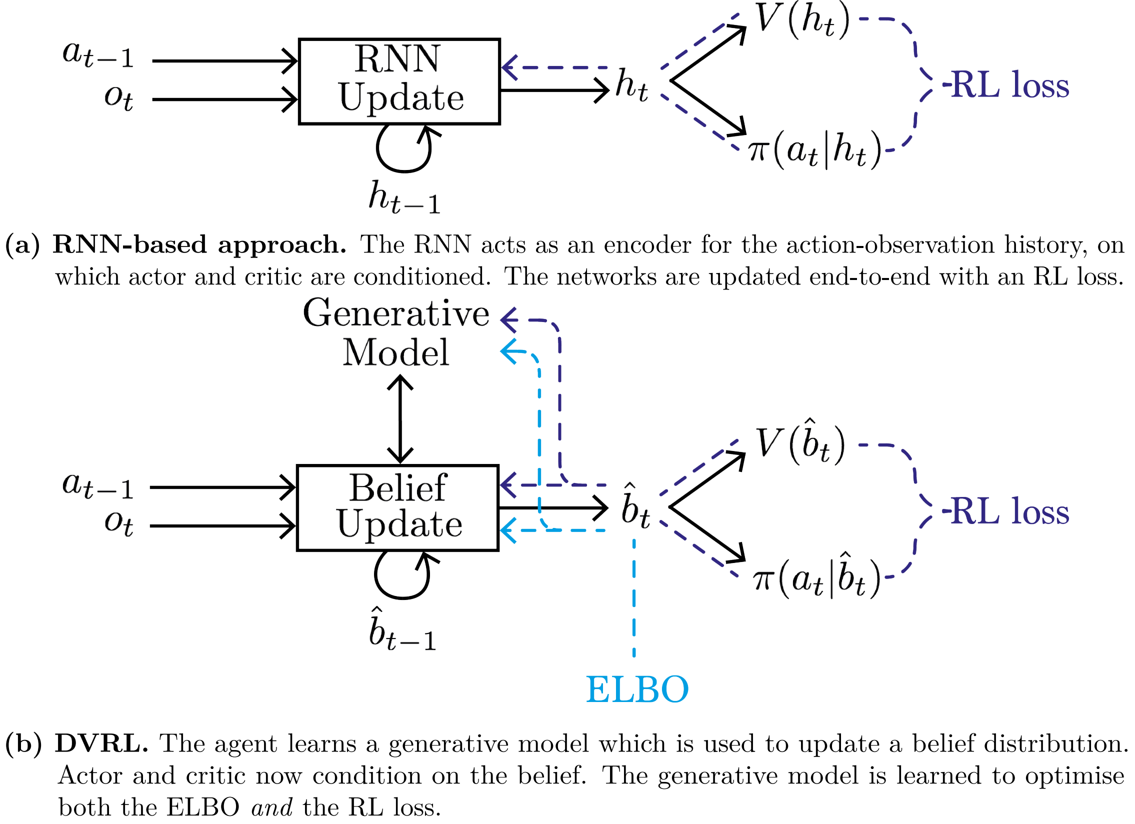
Deep variational reinforcement learning for Partially Observable Markov Decisions Processes

Contents

3.1 Introduction **29**
3.2 Background **32**
3.2.1 Variational Autoencoders for time series 32
3.2.2 Advantage Actor Critic 33
3.3 Method **33**
3.3.1 Baseline architecture 34
3.3.2 Extending the latent state 35
3.3.3 Recurrent latent state update 37
3.3.4 Loss function 38
3.4 Related work **39**
3.5 Experiments **41**
3.5.1 Mountain hike 41
3.5.2 Atari 43
3.5.3 Ablation studies 45
3.6 Conclusion **46**

3.1 Introduction

Many deep RL methods assume that the state of the environment is fully observable at every time step [40, 43, 44]. However, this assumption often does not hold in



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.

(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 3.1: Comparison of RNN and DVRL encoders.

reality, as occlusions and noisy sensors may limit the agent's perceptual abilities. Such problems can be formalised as partially observable Markov decision process (POMDP) [45, 46]. Because we usually do not have access to the true generative model of our environment, there is a need for reinforcement learning methods that can tackle POMDPs when only a stream of observations is given, without any prior knowledge of the latent state space or the transition and observation functions.

POMDPs are notoriously hard to solve: since the current observation does in general not carry all relevant information for choosing an action, information must be aggregated over time and typically, the entire history must be taken into account.

This history can be encoded either by remembering features of the past [47] or by performing inference to determine the distribution over possible latent states [46]. However, the computation of this *belief state* requires knowledge of the model.

Most previous work on deep learning in POMDPs relies on training a RNN to

summarise the past. Examples are the deep recurrent Q-network (DRQN) [48] and the action-specific deep recurrent Q-network (ADRQN) [49]. Because these approaches are completely model-free, they place a heavy burden on the RNN. Since performing inference implicitly requires a known or learned model, they are likely to summarise the history either by only remembering features of the past or by computing simple heuristics instead of actual belief states. This is often suboptimal in complex tasks. Generalisation is also often easier over beliefs than over trajectories since distinct histories can lead to similar or identical beliefs.

The premise of this work is that deep policy learning for POMDPs can be improved by taking less of a black box approach than DRQN and ADRQN. While we do not want to assume prior knowledge of the transition and observation functions or the latent state representation, we want to allow the agent to learn models of them and infer the belief state using these learned models.

In this chapter, we propose deep variational reinforcement learning (DVRL), which implements this approach by providing a helpful inductive bias to the agent. In particular, we develop an algorithm that can learn an internal generative model and use it to perform approximate inference to update the belief state. Crucially, the generative model is not only learned based on an ELBO objective, but also by how well it enables maximisation of the expected return. This ensures that, unlike in an unsupervised application of VAEs, the latent state representation and the inference performed on it are suitable for the ultimate control task. Specifically, we develop an approximation to the ELBO based on autoencoding sequential Monte Carlo (AESMC) [50], allowing joint optimisation with the n-step policy gradient update. Uncertainty in the belief state is captured by a particle ensemble. A high-level overview of our approach in comparison to previous RNN-based methods is shown in Figure 3.1.

We evaluate our approach on Mountain Hike and several *flickering* Atari games. On Mountain Hike (a low dimensional, continuous environment), we can show that DVRL is better than an RNN based approach at inferring the required information from past observations for optimal action selection in a simple setting. Our results on flickering Atari show that this advantage extends to complex environments with

high dimensional observation spaces. Here, partial observability is introduced by i) using only a single frame as input at each time step and ii) returning a blank screen instead of the true frame with probability 0.5.

3.2 Background

In Chapter 2 we already introduced POMDPs and the VAE. Here, we provide some additional background on how the VAE formalism can be extended to time-series data, as encountered in RL when we aim to encode entire trajectories like in POMDPs.

3.2.1 Variational Autoencoders for time series

For sequential data, we assume that a series of latent states $Z_{\leq T}$ gives rise to a series of observations $O_{\leq T}$. We consider a family of generative models parameterised by θ that consists of the initial distribution $p_\theta(Z_0)$, transition distribution $p_\theta(Z_t \mid Z_{t-1})$ and observation distribution $p_\theta(O_t \mid Z_t)$. Given a family of encoder distributions $q_\phi(Z_t \mid Z_{t-1}, O_t)$, we can also estimate the gradient of the ELBO term in the same manner as in (2.19), noting that:

$$p_\theta(Z_{\leq T}, O_{\leq T}) = p_\theta(Z_0) \prod_{t=1}^{T} p_\theta(Z_t \mid Z_{t-1}) p_\theta(O_t \mid Z_t), \tag{3.1}$$

$$q_\phi(Z_{\leq T} \mid O_{\leq T}) = p_\theta(Z_0) \prod_{t=1}^{T} q_\phi(Z_t \mid Z_{t-1}, O_t), \tag{3.2}$$

where we slightly abuse notation for q_ϕ by ignoring the fact that we sample from the model $p_\theta(S_0)$ for $t = 0$. Le et al. [50], Maddison et al. [51] and Naesseth et al. [52] introduce a new ELBO objective based on sequential Monte Carlo (SMC) [53] that allows faster learning in time series:

$$\text{ELBO}_{\text{SMC}}(\theta, \phi, o_{\leq T}) = \mathbb{E}\left[\sum_{t=1}^{T} \log\left(\frac{1}{K}\sum_{k=1}^{K} w_t^k\right)\right], \tag{3.3}$$

where K is the number of particles and w_t^k is the weight of particle k at time t. Each particle is a tuple containing a weight w_t^k and a value z_t^k which is obtained as follows. Let z_0^k be samples from $p_\theta(Z_0)$ for $k = 1, \ldots, K$. For $t = 1, \ldots, T$, the

weights w_t^k are obtained by resampling the particle set $(z_{t-1}^k)_{k=1}^K$ proportionally to the previous weights and computing

$$w_t^k = \frac{p_\theta(z_t^k \mid z_{t-1}^{u_{t-1}^k}) p_\theta(o_t \mid z_t^k)}{q_\phi(z_t^k \mid z_{t-1}^{u_{t-1}^k}, o_t)}, \tag{3.4}$$

where z_t^k corresponds to a value sampled from $q_\phi(Z_t^k \mid z_{t-1}^{u_{t-1}^k}, o_t)$ and $z_{t-1}^{u_{t-1}^k}$ corresponds to the resampled particle with the ancestor index $u_0^k = k$ and $u_{t-1}^k \sim \text{Discrete}((w_{t-1}^k / \sum_{j=1}^K w_{t-1}^j)_{k=1}^K)$ for $t = 2, \ldots, T$.

3.2.2 Advantage Actor Critic

In Section 2.3.3 we have already introduced the deep policy gradient method PPO. In this chapter, we will be using the similar, but slightly simpler, method A2C [54, 55] in order to learn the agent's policy $\pi_\rho(a_t|s_t)$. While in recent years PPO has become one of the most widely used training algorithms, A2C was, at the time, more established. To avoid confusion with the learned generative model and inference network, we use ρ to denote the policy parameters in this chapter. A2C also relies on the advantge estimation Eq. (2.20), but replaces the proxy loss Eq. (2.21) with the simpler one

$$L_{\text{A2C}}(\rho) = \mathbb{E}_{\mathcal{D}_\tau} \left[\log \pi_\rho(a_t|s_t) A_t \right]. \tag{3.5}$$

The losses for the value-function and entropy remain unchanged and we again make use of multiple parallel environments, i.e. multiple streams of experience for the agent..

3.3 Method

Fundamentally, there are two approaches to aggregating the history in the presence of partial observability: remembering features of the past or maintaining beliefs.

In most previous work, including ADRQN [49], the current history $(a_{\leq t}, o_{<t})$ is encoded by an RNN, which leads to the recurrent update equation for the latent state h_t:

$$h_t = \text{RNNUpdate}_\phi(h_{t-1}, a_{t-1}, o_t) \;. \tag{3.6}$$

Since this approach is model-free and does not make use of any generative model of the environment, it is unlikely (albeit possible, see e.g. [56]) to approximate belief update steps, instead relying on memory or simple heuristics.

Inspired by the premise that a good way to solve many POMDPs involves i) estimating the transition and observation model of the environment, ii) performing inference under this model, and iii) choosing an action based on the inferred belief state, we propose deep variational reinforcement learning (DVRL). It extends the RNN-based approach to explicitly support belief inference. Training everything end-to-end shapes the learned model to be useful for the RL task at hand, and not only for predicting observations. For belief inference, we rely on particle filtering which has itself proven to be one of the most powerful methods for latent state inference, both under known system dynamics [53], as well as when these are jointly learned using neural networks [50–52].

We first explain our baseline architecture and training method in Section 3.3.1. For a fair comparison, we modify the original architecture of Zhu, Li, and Poupart [49] in several ways. We find that our new baseline outperforms their reported results in the majority of cases.

In Sections 3.3.2 and 3.3.3, we explain our new latent belief state $\hat{b}_t$ and the recurrent update function

$$\hat{b}_t = \mathrm{BeliefUpdate}_{\theta,\phi}(\hat{b}_{t-1}, a_{t-1}, o_t) \tag{3.7}$$

which replaces Equation (3.6). Lastly, in Section 3.3.4, we describe our modified loss function, which allows learning the model jointly with the policy.

3.3.1 Baseline architecture

We will compare DVRL to an RNN based encoder as shown in Figure 3.1. While previous work often used Q-learning to train the policy [48, 49, 57, 58], we use n-step A2C (see Section 3.2.2). This avoids drawing entire trajectories from a replay buffer and allows continuous actions.

Furthermore, since A2C interleaves unrolled trajectories and performs a parameter update only every n_s steps, it makes it feasible to maintain an approximately correct latent state. A small bias is introduced by not recomputing the latent state after each gradient update step.

We also modify the implementation of backpropagation-throught-time (BPTT) for n-step A2C in the case of policies with latent states. Instead of back-propagating gradients only through the computation graph of the current update involving n_s steps, we set the size of the computation graph independently to involve up to n_g steps. This leads to an average BPTT-length of $\frac{1}{L}\sum_{i=1}^{L} in_s$ for $L = \frac{n_g}{n_s}$ where n_g is a multiple of n_s[1]. This decouples the bias-variance tradeoff of choosing n_s from the bias-runtime tradeoff of choosing n_g. Our experiments show that choosing $n_g > n_s$ greatly improves the agent's performance.

3.3.2 Extending the latent state

For DVRL, we extend the latent state to be a set of K particles, capturing the uncertainty in the belief state [59, 60]. Each particle consists of the triplet (h_t^k, z_t^k, w_t^k) [61]. The value h_t^k of particle k is the latent state of an RNN; z_t^k is an additional stochastic latent state that allows us to learn stochastic transition models; and w_t^k assigns each particle an importance weight. Our belief state $\hat{b}_t$ is thus an approximation of the posterior distribution in our learned model

$$
\begin{aligned}
&p_\theta(H_{\leq T}, Z_{\leq T}, O_{\leq T} \mid a_{<T}) = \\
&p_\theta(H_0) \prod_{t=1}^{T} \Big(p_\theta(Z_t|H_{t-1}, a_{t-1}) p_\theta(O_t|H_{t-1}, Z_t, a_{t-1}) \delta_{\psi_\theta^{\text{RNN}}(H_{t-1}, Z_t, a_{t-1}, O_t)}(H_t) \Big),
\end{aligned} \tag{3.8}
$$

with stochastic transition model $p_\theta(Z_t|H_{t-1}, a_{t-1})$, decoder $p_\theta(O_t|H_{t-1}, Z_t, a_{t-1})$, and deterministic transition function $H_t = \psi_\theta^{\text{RNN}}(H_{t-1}, Z_t, O_t, a_{t-1})$ which is denoted using the Dirac delta distribution δ and for which we use an RNN. The model is trained to jointly optimise the ELBO and the expected return.

[1] This is implemented in PyTorch using the (now deprecated) `retain_graph=True` flag in the `backward()` function: The computation graph is only reset every n_g/n_s updates and otherwise extended.

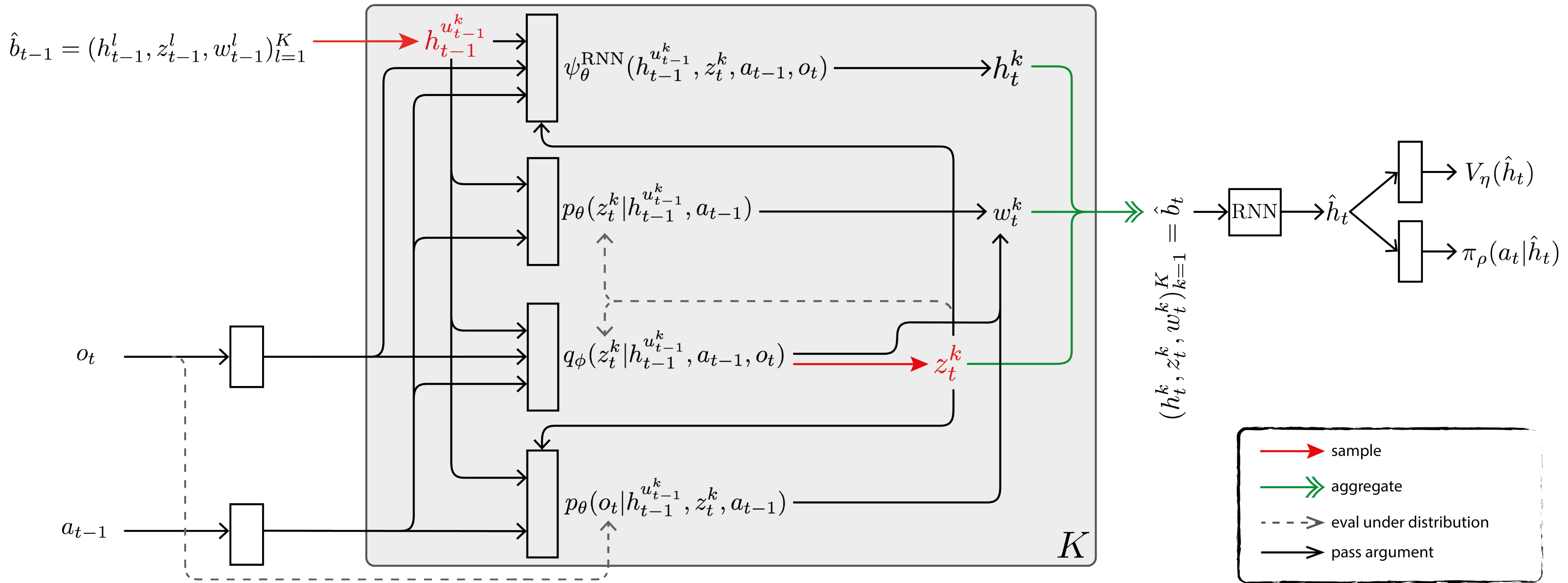


Figure 3.2: Overview of DVRL. We do the following K times to compute our new belief $\hat{b}_t$: Sample an ancestor index u^k_{t-1} based on the previous weights $w^{1:K}_{t-1}$ (Eq. 3.9). Pick the ancestor particle value $h^{u^k_{t-1}}_{t-1}$ and use it to sample a new stochastic latent state z^k_t from the encoder q_ϕ (Eq. 3.10). Compute h^k_t (Eq. 3.11) and w^k_t (Eq. 3.12). Aggregate all K values into the new belief $\hat{b}_t$ and summarise them into a vector representation $\hat{h}_t$ using a second RNN. Actor and critic can now condition on $\hat{h}_t$ and $\hat{b}_t$ is used as input for the next timestep. Red arrows denote random sampling, green arrows the aggregation of K values. Black solid arrows denote the passing of a value as argument to a function and black dashed ones the evaluation of a value under a distribution. Boxes indicate neural networks. Distributions are normal or Bernoulli distributions whose parameters are outputs of the neural network.

3.3.3 Recurrent latent state update

To update the latent state, we proceed as follows:

$$u_{t-1}^k \sim \text{Discrete}\left(\frac{w_{t-1}^k}{\sum_{j=1}^K w_{t-1}^j}\right) \tag{3.9}$$

$$z_t^k \sim q_\phi(Z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t) \tag{3.10}$$

$$h_t^k = \psi_\theta^{\text{RNN}}(h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1}, o_t) \tag{3.11}$$

$$w_t^k = \frac{p_\theta(z_t^k \mid h_{t-1}^{u_{t-1}^k}, a_{t-1}) p_\theta(o_t \mid h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1})}{q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)} \,. \tag{3.12}$$

First, we resample particles based on their weight by drawing ancestor indices u_{t-1}^k. This improves model learning [50, 51] and allows us to train the model jointly with the n-step loss (see Section 3.3.4).

For $k = 1, \ldots, K$, new values for z_t^k are sampled from the encoder $q_\phi(Z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)$ which conditions on the resampled ancestor values $h_{t-1}^{u_{t-1}^k}$ as well as the last actions a_{t-1} and current observation o_t. Latent variables z_t are sampled using the reparameterization trick. The values z_t^k, together with $h_{t-1}^{u_{t-1}^k}, a_{t-1}$ and o_t, are then passed to the transition function ψ_θ^{RNN} to compute h_t^k.

The weights w_t^k measure how likely each new latent state value (z_t^k, h_t^k) is under the model and how well it explains the current observation.

To condition the policy π_ρ and value function V_θ on the belief $\hat{b}_t = (z_t^k, h_t^k, w_t^k)_{k=1}^K$, we need to summarise the set of particles into a single vector representation $\hat{h}_t$. One option would be to compute a weighted average over K policies and value functions that each take in a single particle value (z_t^k, h_t^k) – this however would ignore the uncertainty in the latent state after the next action [62]. Instead, we use a (second) RNN that sequentially takes in each tuple (z_t^k, h_t^k, w_t^k) and outputs $\hat{h}_t$ as its last latent state. While we found this approach to work well in practise, many other alternatives are feasible, including simple concatenation or the use of a Deep Set architecture [63].

Additional encoders are used for a_t, o_t and z_t; see Appendix A for details. Figure 3.2 summarises the entire update step.

3.3.4 Loss function

To encourage learning a model, we include the term

$$L^{\text{ELBO}}(\theta, \phi) = \mathbb{E}_{\mathcal{D}_\tau}\left[\log\left(\frac{1}{K}\sum_{k=1}^{K} w_{t+i}^k\right)\right] \tag{3.13}$$

in each gradient update every n_s steps. This leads to the overall loss:

$$\mathcal{L}^{\text{DVRL}}(\rho, \theta, \phi) = L_{\text{PPO}}(\rho, \theta, \phi) + \lambda^H \mathbb{E}_{\mathcal{D}_\tau}\left[H[\pi_\rho(\cdot|\hat{b}_{\theta,\phi})]\right] \tag{3.14}$$

$$+\lambda^V \mathbb{E}_{\mathcal{D}_\tau}\left[A_\rho^2(\hat{b}_{\theta,\phi})\right] + \lambda^E L^{\text{ELBO}}(\theta, \phi)\ . \tag{3.15}$$

The losses now also depend on the encoder parameters ϕ and model parameters θ, since the policy and value function now condition on the latent states instead of s_t. By introducing a n-step approximation of L^{ELBO} instead of optimising it over entire trajectories, we can learn θ and ϕ to jointly optimise the ELBO and the RL loss.

If we assume that observations and actions are drawn from the stationary state distribution induced by the policy π_ρ, then $-L^{\text{ELBO}}$ over n_s trajectory steps is a stochastic approximation to the action-conditioned ELBO:

$$\frac{1}{T}\mathbb{E}_{p(\tau)}\,\text{ELBO}_{\text{SMC}}(o_{\leq T}|a_{<T}) = \frac{1}{T}\mathbb{E}_{p(\tau)}\mathbb{E}\left[\sum_{t=1}^{T}\log\left(\frac{1}{K}\sum_{k=1}^{K} w_t^k\right)\middle| a_{\leq T}\right]\ , \tag{3.16}$$

which is a conditional extension of Equation (3.3), similar to the extension of VAEs by Sohn, Lee, and Yan [64]. To make Equation (3.16) tractable, we approximate the expectation over $p(\tau)$ by using sampled trajectories from n_e environments. Furthermore, because we assume a stationary state distribution, we can take the sum $\sum_{t=1}^{T}$ outside of both expectations. This allows us to perform a stochastic gradient update that is based on only n_s summands instead of all T, leading to Equation (3.13) which includes an additional minus sign to account for its minimisation.

The importance of the resampling step (3.9) in allowing this approximation becomes clear if we compare (3.16) with the alternative ELBO for the importance weighted autoencoder (IWAE) [53, 65] that does not include resampling:

$$\text{ELBO}_{\text{IWAE}}(o_{\leq T}|a_{<T}) = \mathbb{E}\left[\log\left(\frac{1}{K}\sum_{k=1}^{K}\prod_{t=1}^{T} w_t^k\right)\middle| a_{\leq T}\right]\ . \tag{3.17}$$

Here, the product over time and summation over particles are swapped. Because this loss is not additive over time anymore, we cannot approximate it with shorter parts of the trajectory. This prevents joint optimisation with the RL loss. While too frequent multinomial resampling could increase the variance of the loss function estimator, more advanced resampling strategies like residual, stratified or systematic resampling provide effective tools at minimizing this effect. [66].

3.4 Related work

Most existing POMDP literature focusses on *planning* algorithms, where the transition and observation functions, as well as a representation of the latent state space, are known [67–72]. In most realistic domains however, these are not known a priori.

There are several approaches that utilise RNNs in POMDPs [73–76], including multi-agent settings [57], learning text-based fantasy games [58] or applied to Atari [48, 49]. As discussed in Section 3.3, our algorithm extends those approaches by enabling the policy to explicitly reason about a model and the belief state.

More recently, after the work in this chapter was published, transformers have emerged as another powerful architecture acting on state-action histories [24, 77].

Another more specialised approach called QMDP-Net [78] learns a value iteration network (VIN) [79] end-to-end and uses it as a transition model for planning. However, a VIN makes strong assumptions about the transition function and in QMDP-Net the belief update must be performed analytically.

The idea to learn a particle filter based policy that is trained using policy gradients was previously proposed by Coquelin, Deguest, and Munos [80]. However, they assume a known model and rely on finite differences for gradient estimation. While this approach is unbiased, it is not feasible for most tasks in which deep RL methods would be applied, due to the typically unknown environmental dynamics and the limited scaling of finite difference methods for larger neural networks.

Instead of optimising an ELBO to learn a maximum-likelihood approximation for the latent representation and corresponding transition and observation model, previous work also tried to learn those dynamics using spectral methods [81], a

Bayesian approach [82, 83], or nonparametrically [84]. However, these approaches do not scale to large or continuous state and observation spaces. For continuous states, actions, and observations with Gaussian noise, a Gaussian process model can be learned [85]. An alternative to learning an (approximate) transition and observation model is to learn a model over trajectories [86]. However, this is again only possible for small, discrete observation spaces.

Due to the complexity of the learning in POMDPs, previous work already found benefits to using auxiliary losses. Unlike the losses proposed by Lample and Chaplot [87], we do not require additional information from the environment. The *UNREAL* agent [88] is, similarly to our work, motivated by the idea to improve the latent representation by utilising all the information already obtained from the environment. While their work focuses on finding unsupervised auxiliary losses that provide good training signals, our goal is to use the auxiliary loss to better align the network computations with the task at hand by incorporating prior knowledge as an inductive bias.

Furthermore, predictions can not only be used as auxiliary losses, but can also be used as a basis for representing states in partially observable environments: For *Predictive State Representations*, instead of summarising the past, each state is a set of predictions about observable outcomes of tests one might probe the environment with [89, 90]. More recent work also combines this approach with learning RNNs [91, 92].

There is some evidence from recent experiments on the dopamine system in mice [93] showing that their response to ambiguous information is consistent with a theory operating on belief states.

Lastly, since publication of this work, several improvements for particle based history aggregation in RL have been proposed. Ma et al. [94] replace the generative decoder with a discriminator, thereby alleviating the generative model from having to learn unnecessary details in the environment. On the other hand, Wang et al. [95] not only use particles for filtering of the past, but also for planning into the future.

3.5 Experiments

We evaluate DVRL on Mountain Hike and on flickering Atari. We show that DVRL deals better with noisy or partially occluded observations and that this scales to high dimensional and continuous observation spaces like images and complex tasks. We also perform a series of ablation studies, showing the importance of using many particles, including the ELBO training objective in the loss function, and jointly optimising the ELBO and RL losses.

More details about the environments and model architectures can be found in Appendix A together with additional results and visualisations. All plots and reported results are smoothed over time and parallel executed environments. We average over five random seeds, with shaded areas indicating the standard deviation. All RNNs are gated recurrent units (GRUs) [96]. DVRL is trained end-to-end using $\mathcal{L}^{\text{DVRL}}$ while the RNN baseline is trained using A2C.

3.5.1 Mountain hike

In this task, the agent has to navigate along a mountain ridge, but only receives noisy measurements of its current location. Specifically, we have $\mathcal{S} = \mathcal{O} = \mathcal{A} = \mathbb{R}^2$ where $s_t = [x_t, y_t]^T \in \mathcal{S}$ and $o_t = [\hat{x}_t, \hat{y}_t]^T \in \mathcal{O}$ are true and observed coordinates respectively and $a_t = [\Delta x_t, \Delta y_t]^T \in \mathcal{A}$ is the desired step. Transitions are given by $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$ with $\epsilon_{s,t} \sim \mathcal{N}(\cdot|0, 0.25 \cdot I)$ and $\tilde{a}_t$ is the vector a_t with length capped to $\|\tilde{a}_t\| \leq 0.5$. Observations are noisy with $o_t = s_t + \epsilon_{o,t}$ with $\epsilon_{o,t} \sim \mathcal{N}(\cdot|0, \sigma_o \cdot I)$ and $\sigma_o \in \{0, 1.5, 3\}$. The reward at each timestep is $R_t = r(x_t, y_t) - 0.01\|a_t\|$ where $r(x_t, y_t)$ is shown in Figure 3.3. The starting position is sampled from $s_0 \sim \mathcal{N}(\cdot|[-8.5, -8.5]^T, I)$ and each episode ends after 75 steps.

DVRL used 30 particles and we set $n_g = 25$ for both RNN and DVRL. The latent state h for the RNN-encoder architecture was of dimension 256 and 128 for both z and h for DVRL. Lastly, $\lambda^E = 1$ and $n_s = 5$ were used, together with RMSProp with a learning rate of 10^{-4} for both approaches.

Care was taken to design the environment without local minima or challenging exploration problems. Consequently, the main difficulty in Mountain Hike is to

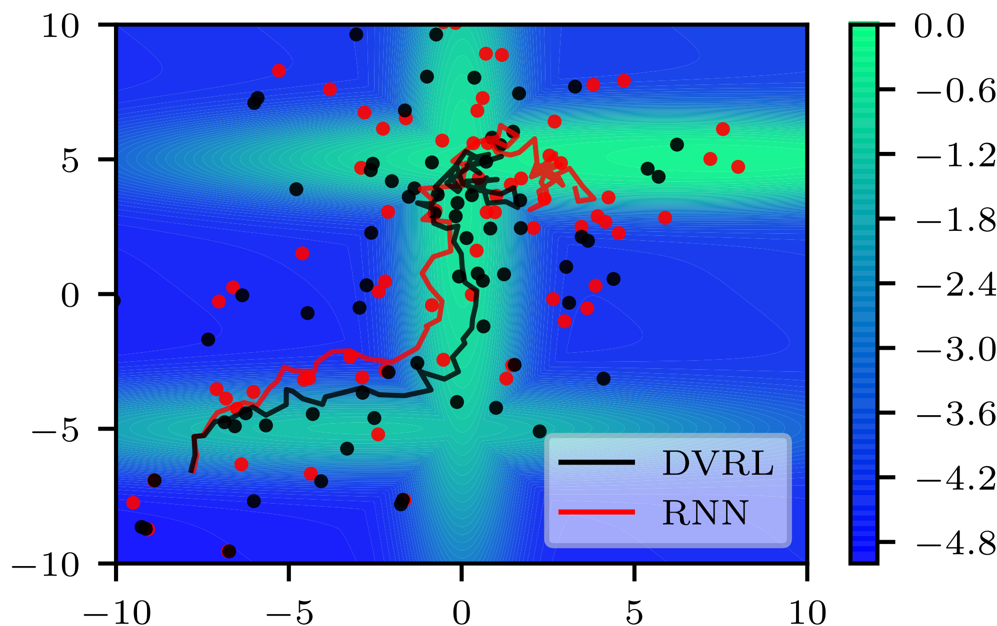


Figure 3.3: Mountain Hike is a continuous control task with observation noise $\sigma_o = 3$. Background colour indicates rewards. Red line: trajectory for RNN based encoder. Black line: trajectory for DVRL encoder. Dots: received observations. Both runs share the same noise values $\epsilon_{i,t}$. DVRL achieves higher returns (see Fig. 3.4) by better estimating its current location and remaining on the high reward mountain ridge.

correctly estimate the current position and the achieved return reflects the capability of the network to do so. In particular, this design also contributes to the observed low variance of agent performance across random seeds. DVRL outperforms RNN based policies, especially for higher levels of observation noise σ_o (Figure 3.4). In Figure 3.3 we compare the different trajectories for RNN and DVRL encoders for the same noise, i.e. $\epsilon_{s,t}^{\text{RNN}} = \epsilon_{s,t}^{\text{DVRL}}$ and $\epsilon_{o,t}^{\text{RNN}} = \epsilon_{o,t}^{\text{DVRL}}$ for all t and $\sigma_o = 3$. DVRL is better able to follow the mountain ridge, indicating that its inference based history aggregation is superior to a largely memory/heuristics based one. Note that while DVRL initially requires more time to learn, it quickly surpasses the performance of the RNN baseline. This is not unexpected as it not only has to learn how to solve the task, but also has to infer an additional dynamics model describing the environment. Fluctuations in performance during this initial training phase are likely due to the stochastic nature of SGD.

The example in Figure 3.3 is representative but selected for clarity: The shown trajectories have $\Delta J(\sigma_o = 3) = 20.7$ compared to an average value of $\Delta \bar{J}(\sigma_o = 3) = 11.43$ (see Figure 3.4).

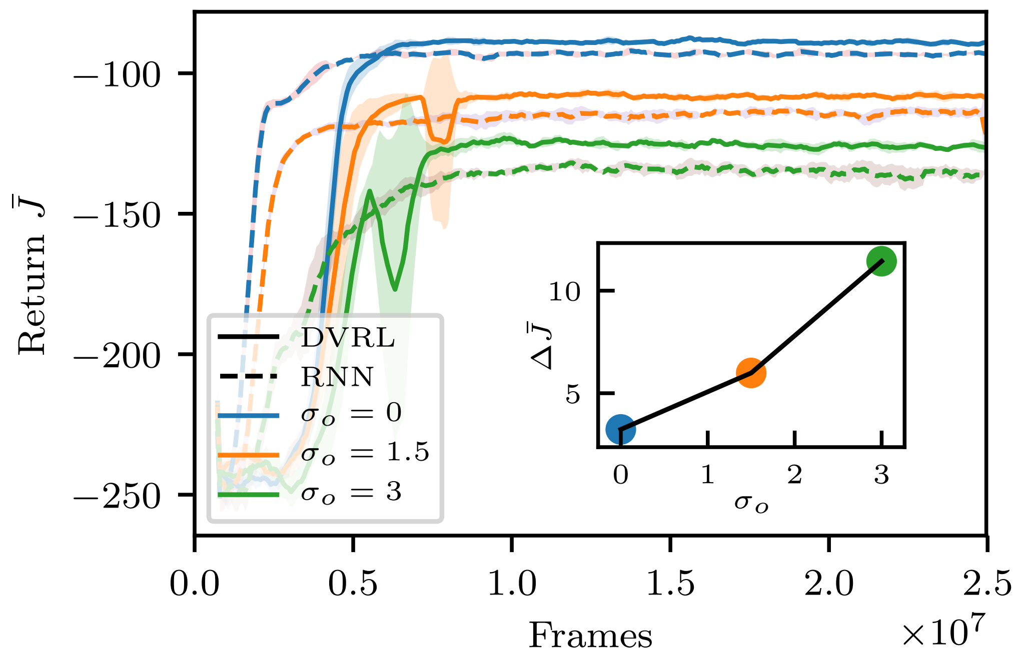


Figure 3.4: Returns achieved in Mountain Hike. Solid lines: DVRL. Dashed lines: RNN. Colour: Noise levels. *Inset:* Difference in performance between RNN and DVRL for same level of noise: $\Delta \bar{J}(\sigma_o) = \bar{J}(\text{DVRL}, \sigma_o) - \bar{J}(\text{RNN}, \sigma_o)$. DVRL achieves slighly higher returns for the fully observable case and, crucially, its performance deteriorates more slowly for increasing observation noise, showing the advantage of DVRL's inference computations in encoding the history in the presence of observation noise.

3.5.2 Atari

We chose flickering Atari as evaluation benchmark, since it was previously used to evaluate the performance of ADRQN [49] and DRQN [48]. Atari environments [97] provide a wide set of challenging tasks with high dimensional observation spaces. We test our algorithm on the same subset of games on which DRQN and ADRQN were evaluated.

Partial observability is introduced by *flickering*, i.e., by a probability of 0.5 of returning a blank screen instead of the actual observation. Furthermore, only one

frame is used as the observation. This is in line with previous work [48]. We use a frameskip of four[2] and for the stochastic Atari environments there is a 0.25 chance of repeating the current action for a second time at each transition.

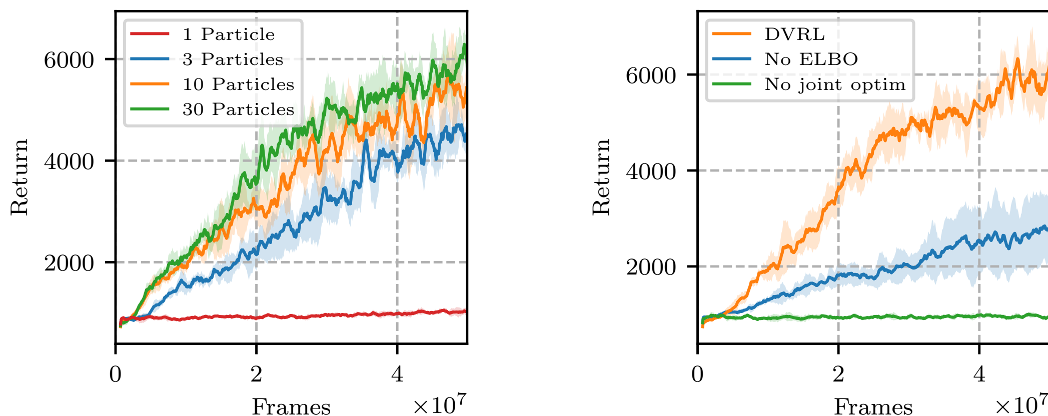
DVRL used 15 particles and we set $n_g = 50$ for both agents. The dimension of h was 256 for both architectures, as was the dimension of z. Larger latent states decreased the performance for the RNN encoder. Lastly, $\lambda^E = 0.1$ and $n_s = 5$ was used with a learning rate of 10^{-4} for RNN and $2 \cdot 10^{-4}$ for DVRL, selected out of a set of 6 different rates based on the results on ChopperCommand.

Table 3.1 shows the results for the more challenging stochastic, flickering environments. Results for the deterministic environments, including returns reported for DRQN and ADRQN, can be found in Appendix A. DVRL significantly outperforms the RNN-based policy on five out of ten games and narrowly underperforms significantly on only one. This shows that DVRL is viable for high dimensional observation spaces with complex environmental models.

Table 3.1: Returns on stochastic and flickering Atari environments, averaged over 5 random seeds. Bold numbers indicate statistical significance at the 5% level as determined by a paird t-test. We did not observe multi-modality in the distribution of outcomes and therefore believe it is reasonable to approximate it as Gaussian. Out of ten games, DVRL significantly outperforms the baseline on five games and underperforms narrowly on only one game. Comparisons against DRQN and ADRQN on deterministic Atari environments are in Appendix A.

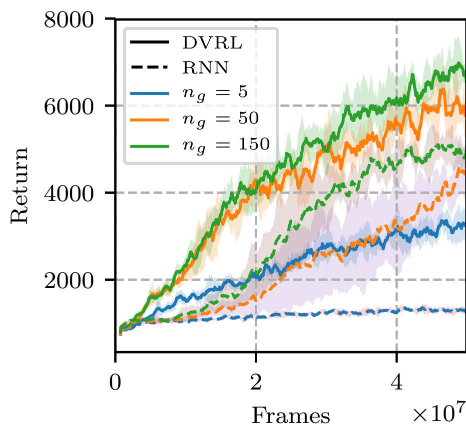
Env	DVRL($\pm std$)	RNN($\pm std$)
Pong	**18.17**(±2.67)	6.33(±3.03)
Chopper	**6602**(±449)	5150(±488)
MsPacman	2221(±199)	2312(±358)
Centipede	4240(±116)	4395(±224)
BeamRider	1663(±183)	1801(±65)
Frostbite	**297**(±7.85)	254(±0.45)
Bowling	29.53(±0.23)	**30.04**(±0.18)
IceHockey	**−4.88**(±0.17)	−7.10(±0.60)
DDunk	**−5.95**(±1.25)	−15.88(±0.34)
Asteroids	1539(±73)	1545(±51)

[2] A frameskip of one is used for Asteroids due to known rendering issues with this environment



(a) Influence of the particle number on performance for DVRL. Only using one particle is not sufficient to encode enough information in the latent state.

(b) Performance of the full DVRL algorithm compared to setting $\lambda^E = 0$ ("No ELBO") or not backpropagating the policy gradients through the encoder ("No joint optim").



(c) Influence of the maximum backpropagation length n_g on performance. Note that RNN suffers most from very short lengths. This is consistent with our conjecture that RNN relies mostly on memory, not inference.

Figure 3.5: Ablation studies on flickering ChopperCommand (Atari).

3.5.3 Ablation studies

Using more than one particle is important to accurately approximate the belief distribution over the latent state (z, h). Consequently, we expect that higher particle numbers provide better information to the policy, leading to higher returns. Figure 3.5a shows that this is indeed the case. This is an important result for our architecture, as it also implies that the resampling step is necessary, as detailed in

Section 3.3.4. Without resampling, we cannot approximate the ELBO on only n_s observations.

Secondly, Figure 3.5b shows that the inclusion of L^{ELBO} to encourage model learning is required for good performance. Furthermore, not backpropagating the policy gradients through the encoder and only learning it based on the ELBO ("No joint optim") also deteriorates performance.

Lastly, we investigate the influence of the backpropagation length n_g on both the RNN and DVRL based policies. While increasing n_g universally helps, the key insight here is that a short length $n_g = 5$ (for an average BPTT-length of 2 timesteps) has a stronger negative impact on RNN than on DVRL. This is consistent with our notion that RNN is mainly performing memory based reasoning, for which longer backpropagation-through-time is required: The belief update (2.13) in DVRL is a one-step update from b_t to b_{t+1}, without the need to condition on past actions and observations. The proposal distribution can benefit from extended backpropagation lengths, but this is not necessary. Consequently, this result supports our notion that DVRL relies more on inference computations to update the latent state.

3.6 Conclusion

In this chapter we proposed DVRL, a method for solving POMDPs given only a stream of observations, without knowledge of the latent state space or the transition and observation functions operating in that space. Our method leverages a new ELBO-based auxiliary loss and incorporates an inductive bias into the structure of the policy network, taking advantage of our prior knowledge that performing inference in POMDPs can be an effective way of constructing state representations for optimal solutions.

We compared DVRL to an RNN-based architecture and found that we consistently outperform it on a diverse set of tasks, including a number of Atari games modified to have partial observability and stochastic transitions.

We also performed several ablation studies showing the necessity of using an ensemble of particles and of joint optimisation of the ELBO and RL objective. Furthermore, the results support our claim that the latent state in DVRL approximates a belief distribution in a learned model.

Access to a belief distribution opens up several interesting research directions. Investigating the role of better generalisation capabilities and the more powerful latent state representation on the policy performance of DVRL can give rise to further improvements. DVRL is also likely to benefit from more powerful model architectures and a disentangled latent state. Furthermore, uncertainty in the belief state and access to a learned model can be used for curiosity driven exploration in environments with sparse rewards.

Embedding an inductive bias into the network architecture allowed us to improve the asymptotic performance of our agents. However, it did not help to improve the learning speed as it required the agent to learn an additional environmental model (see e.g. Fig. 3.4). One promising approach to significantly reduce sample complexity in training is multitask- or transfer-learning in which policies can be re-used across multiple environments. In the next chapter we will explore how the inclusion of an explicit hierarchy can help greatly in learning such transferable policies.

4
Multitask soft option learning

Contents

4.1 Introduction **48**
4.2 Background **51**
4.2.1 Planning as inference 51
4.2.2 Options 52
4.2.3 Multi-task learning 53
4.3 Method **53**
4.3.1 Hierarchical posterior policies 55
4.3.2 Hierarchical prior policy 55
4.3.3 Objective 56
4.3.4 Soft vs. classical options 57
4.3.5 Local optima option learning 58
4.4 Related work **59**
4.5 Experiments **61**
4.5.1 Moving bandits 63
4.5.2 Taxi 65
4.5.3 Out-of-distribution tasks 66
4.6 Discussion **67**

4.1 Introduction

A key challenge in Deep Reinforcement Learning is to scale current approaches to complex tasks without requiring a prohibitive number of environmental interactions. One promising approach is to construct or learn efficient exploration priors to focus

on more relevant parts of the state-action space, reducing the number of required interactions. This includes, for example, reward shaping [98], curriculum learning [99], meta-learning [100] and transfer learning [101].

In particular, transfer learning does not require human designed rewards or curricula, instead allowing the network to learn what and how to transfer knowledge between tasks. One promising way to capture such knowledge is to decompose policies into a hierarchy of sub-policies (or skills) that can be reused and combined in novel ways to solve new tasks [102]. This idea of Hierarchical RL (HRL) is also supported by findings that humans appear to employ a hierarchical mental structure when solving tasks [103]. In such a hierarchical policy, lower-level, *temporally extended* skills yield directed behavior over multiple time steps. This has two advantages: i) it allows efficient exploration, as the target states of skills can be reached without having to explore much of the state space in between, and ii) directed behavior also reduces the variance of the future reward, which accelerates convergence of estimates thereof. On the other hand, while a hierarchical approach can significantly speed up exploration and training, it can also severely limit the expressiveness of the final policy and lead to suboptimal performance when the temporally extended skills are not able to express the required policy for the task at hand.

Many methods exist for learning such hierarchical skills, [e.g. 102, 104, 105]. The key challenge is to learn skills which are diverse, and relevant for future tasks. One widely used approach is to rely on additional human-designed input, often in the form of manually specified subgoals [106, 107] or a fixed temporal extension of learned skills [108]. While this can lead to impressive results, it is only applicable in situations where relevant subgoals or temporal extension can be easily identified a priori.

In this chapter we propose Multitask Soft Option Learning (MSOL), an algorithm to learn hierarchical skills from a given distribution of tasks without any additional human specified knowledge. MSOL trains simultaneously on multiple tasks from this distribution and autonomously extracts sub-policies which are reusable across them. As discussed above, this allows for more efficient exploration and faster

learning on new tasks from this distribution, assuming there exist such reusable components. In many situations this indeed is the case. For example, locomotion usually relies on a set of repeated movements like standing, walking (stepping) or turning with only their order and length changed between tasks. Similarly, many other tasks like object manipulation equally consist of many 'typical' behaviours like grasping, moving, pushing, etc. Importantly, in contrast to many previous approaches to hierarchical learning, our method also allows for small deviations between applications of such sub-policies, as for example required when grasping different objects or moving to slightly different locations.

Unlike prior work [108], our proposed *soft option* framework avoids several pitfalls of learning options from multiple tasks, which arise when skills are jointly optimized with a higher-level policy that determines when each skill is used. Generally, as each skill must be used for similar purposes across all tasks, to learn consistent behavior, a complex training schedules is required to assure a nearly converged higher-level policy before skills can be updated [108]. However, once a skill has converged it can be hard to change its behavior without hurting the performance of higher-level policies that rely it. Training is therefore prone to end up in local optima: even if changing a skill on *one* task could increase the return, it would likely lead to lower returns on *other* tasks in which it is currently used. This is particularly an issue when multiple skills have learned similar behavior, preventing the learning of a diverse set of skills.

MSOL alleviates both difficulties. The core idea is to learn a "prototypical" – or *prior* – behavior for each skill, while allowing the actually executed skill on each task – the *posterior* – to deviate from it if the specific task rewards require it. Penalizing deviations between the prior and task-specific posteriors gives rise to skills that are consistent across tasks, and can be elegantly formulated in the Planning as Inference (PAI) framework [109]. This distinction between prior and task-specific posterior obviates the need for complex training schedules: individual posteriors can change independently of each other and discover new skills without direct interference in other tasks. Nevertheless, the penalization term encourages

skills to be similar across tasks *and* rewards higher-level policies for preferring such more specialised skills. We discuss in more detail in Section 4.3.5 how this helps to prevent the aforementioned local optima.

In addition to these optimisation pitfalls, the idea of soft options also alleviates the restrictiveness of hierarchical policies. New tasks can make use of learned skills, by initializing their posterior skills from the priors, but are not restricted by them. The penalization term between prior and posterior acts here as learned shaping reward, guiding the exploration on new tasks towards previously relevant behavior, without requiring the new policy to exactly match previous behavior. In difference to prior work, MSOL can thus even learn tasks that are not solvable with the learned skills alone. Finally, we show how the *soft option* framework gives rise to a natural solution to the challenging task of learning option-termination policies. Similar to prior related work, we do assume access to a distribution of similar tasks for training, from which we can sample individual tasks freely without restrictions.

Our experiments demonstrate that MSOL outperforms previous hierarchical and transfer-learning algorithms during transfer tasks in a multitask setting. Unlike prior work, MSOL only modifies the regularized reward and loss function. and does not require specialized architectures, or artificial restrictions on the expressiveness of either the higher-level or intra-option policies.

4.2 Background

In this chapter, we will use the PAI framework, introduced below, to learn hierarchical policies in a multi-task setting. Such policies are often formalised using the *option* framework, which we will also introduce in this section.

4.2.1 Planning as inference

Planning as inference (PAI) [109, 110] frames RL as a probabilistic inference problem. The agent learns a distribution[1] $q_\phi(a|s)$ over actions a given states s, i.e., a policy,

[1] In this chapter, RL will be expressed as variational inference over a generative model. Nevertheless, we will follow our convention of not distinguishing upper and lower case letters for random variables, even when defining generative models, as this would lead to confusion about

parameterized by ϕ, which induces a distribution over trajectories τ of length T, i.e., $\tau = (s_0, a_0, s_1, \ldots, a_T, s_{T+1})$:

$$q_\phi(\tau) = \rho(s_0) \prod_{t=0}^{T} q_\phi(a_t|s_t)\, P(s_{t+1}|s_t, a_t)\,. \tag{4.1}$$

This is very similar to the generative model presented in Section 2.2. However, it can also be seen as a structured variational approximation of the optimal trajectory distribution [see e.g. 109, for more details]. Note that the true initial state probability $\rho(S_0)$ and transition probability $P(s_{t+1}|s_t, a_t)$ are used in the variational posterior, as we can only control the policy, not the environment.

A significant advantage of this formulation is that it is straightforward to incorporate information both from prior knowledge, in the form of a prior policy distribution, and the task at hand through a likelihood function that is defined in terms of the achieved reward. The prior policy $p(a_t|s_t)$ can be specified by hand or, as in our case, learned (see Section 4.3). To incorporate the reward, we introduce a binary *optimality variable* $\mathcal{O}_t$ [109], whose likelihood is highest along the optimal trajectory that maximizes return: $p(\mathcal{O}_t = 1|s_t, a_t) = \exp\Big(r(s_t, a_t)/\beta\Big)$, where for $\beta \to 0$ we recover the original RL problem. The constraint $r \in (-\infty, 0]$ can be relaxed without changing the inference procedure [109]. For brevity, we denote $\mathcal{O}_t = 1$ as $\mathcal{O}_t \equiv (\mathcal{O}_t = 1)$. In this formalism, not only a single trajectory is seen as optimal, but rather our goal is to find a policy under which the trajectory distribution matches the distribution $p(\tau \mid \mathcal{O}_{1:T})$. PAI aims to find such a policy through variational inference, i.e. by finding a variational posterior (Eq. (4.1)) which approximates $p(\tau \mid \mathcal{O}_{1:T})$ by minimizing the Kullback-Leibler (KL) divergence:

$$\begin{aligned} \mathcal{L}(\phi) &= \mathbb{D}_{\text{KL}}\Big(q_\phi(\tau) \,\|\, p(\tau \mid \mathcal{O}_{1:T})\Big)\,, \text{ where} \\ p(\tau, \mathcal{O}_{1:T}) &= \rho(s_1) \prod_{t=1}^{T} p(a_t|s_t) P(s_{t+1}|s_t, a_t) p(\mathcal{O}_t|s_t, a_t). \end{aligned} \tag{4.2}$$

4.2.2 Options

Options [102] are skills that generalize primitive actions and consist of three components: i) an intra-option policy $p(a_t|s_t, z_t)$ that selects primitive actions according

actions a and the advantge A.

to the currently active option z_t, ii) a probability $p(b_t|s_t, z_{t-1})$ of terminating the *previously* active option z_{t-1}, and iii) an initiation set $\mathcal{I} \subseteq \mathcal{S}$, which we simply assume to be $\mathcal{S}$. By construction, the higher-level (or master-) policy $p(z_t|z_{t-1}, s_t, b_t)$ can only select a new option z_t if the previous option z_{t-1} has terminated. Note that z_t used in this chapter is distinct from the stochastic latent state which it denoted in the previous one. However, in both cases it describes an unobserved latent variable.

4.2.3 Multi-task learning

In a multi-task setting, we have a set of different tasks $i \in \mathcal{T}$, drawn from a task distribution with probability $\xi(i)$. All tasks share state space $\mathcal{S}$ and action space $\mathcal{A}$, but each task has its own initial-state distribution ρ_i, transition probability $P_i(s_{t+1}|s_t, a_t)$, and reward function r_i. Our goal is to learn n tasks concurrently, distilling common information that can be leveraged to learn faster on new tasks from $\mathcal{T}$. In this setting, the prior policy $p_\theta(a_t|s_t)$ can be learned jointly with the task-specific posterior policies $q_{\phi_i}(a_t|s_t)$ [101]. To do so, we simply extend Eq. (4.2) to

$$\mathcal{L}(\{\phi_i\}, \theta) = \mathbb{E}_{i\sim\xi}\left[\mathbb{D}_{\mathrm{KL}}\Big(q_{\phi_i}(\tau) \,\|\, p_\theta(\tau, \mathcal{O}_{1:T})\Big)\right] = -\frac{1}{\beta}\,\mathbb{E}_{i\sim\xi,\tau\sim q}\left[\sum_{t=1}^{T} R^{\mathrm{reg}}_{i,t}\right], \qquad (4.3)$$

where $R^{\mathrm{reg}}_{i,t} := r_i(s_t, a_t) - \beta \ln \frac{q_{\phi_i}(a_t|s_t)}{p_\theta(a_t|s_t)}$ is a regularised reward. Minimizing the loss in Eq. (4.3) is equivalent to maximizing the regularized reward $R^{\mathrm{reg}}_{i,t}$. Moreover, minimizing the term $\mathbb{E}_{\tau\sim q}\Big[\ln \frac{q_{\phi_i}(a_t|s_t)}{p_\theta(a_t|s_t)}\Big]$ implicitly minimizes the expected KL-divergence $\mathbb{E}_{s_t\sim q}\Big[\mathbb{D}_{\mathrm{KL}}[q_{\phi_i}(\cdot|s_t)\|p_\theta(\cdot|s_t)]\Big]$. In practise (see Appendix B.2.1) we will also make use of a discount factor $\gamma \in [0, 1]$. For details on how γ arises in the PAI framework we refer to Levine [109]. Note that the parameters θ of the prior are shared between all tasks, while the posterior parameters ϕ_i as specific to task i.

4.3 Method

Our aim in this chapter is to learn a reusable set of options that allow for faster training on new tasks from a given distribution. To differentiate ourselves from classical 'hard' options, which, once learned, do not change during new tasks, we

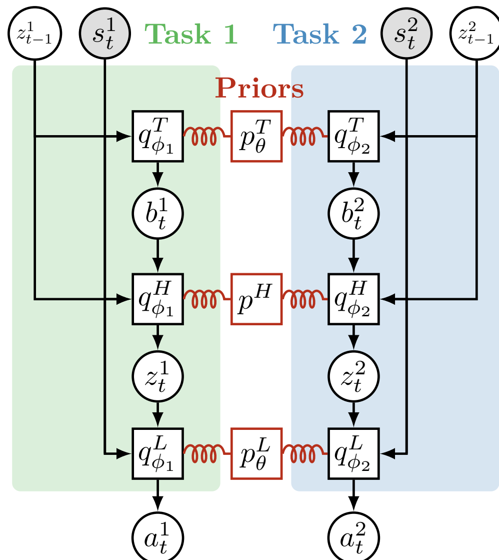


Figure 4.1: Two hierarchical posterior policies (left and right) with common priors (middle). For each task i, the policy conditions on the current state s_t^i and the last selected option z_{t-1}^i. It samples, in order, whether to terminate the last option (b_t^i), which option to execute next (z_t^i) and what primitive action (a_t^i) to execute in the environment.

call our novel approach *soft-options*. Each soft-option consists of an option *prior*, denoted by p_θ, which is shared across all tasks, and a task-specific option *posterior*, denoted by q_{ϕ_i} for task i. Unlike most previous work, e.g. [108], we learn both intra-option and termination policies. The priors of both the intra-option policy p_θ^L and the termination policy p_θ^T capture how an option typically behaves and remain fixed once they are fully learned. At the beginning of training on a new task, they are used to initialize the task-specific posterior distributions $q_{\phi_i}^L$ and $q_{\phi_i}^T$. During training, the posterior is then regularized against the prior to prevent inadvertent unlearning. However, if maximizing the reward on certain tasks is not achievable with the prior policy, the posterior is free to deviate from it. We can thus speed up training using options, while remaining flexible enough to solve more tasks. Additionally, this soft option framework also allows for learning good *priors* in a multitask setting while avoiding complex training schedules and local optima (see Section 4.3.5). In this work, we also learn the higher-level posterior $q_{\phi_i}^H$ within the framework of PAI, but assume a fixed, uniform prior distribution p^H, i.e. we assume there is no shared higher-level structure between tasks. Figure 4.1 shows

an overview over this architecture which we explain further below.

4.3.1 Hierarchical posterior policies

To express options in the PAI framework, we introduce two additional variables at each time step t: *option selections* z_t, representing the currently selected option, and decisions b_t to *terminate* them and allow the higher-level (master) policy to choose a new option. The agent's behavior depends on the currently selected option z_t, by drawing actions a_t from the *intra-option posterior policy* $q^L_{\phi_i}(a_t|s_t, z_t)$. The selection z_t itself is drawn from a *master policy* $q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t) = (1 - b_t)\,\delta(z_t - z_{t-1}) + b_t\, q^H_{\phi_i}(z_t|s_t)$, which conditions on $b_t \in \{0, 1\}$, drawn by the *termination posterior policy* $q^T_{\phi_i}(b_t|s_t, z_{t-1})$. The master policy either continues with the previous z_{t-1} or draws a new option, where we set $b_1 = 1$ at the beginning of each episode. We slightly abuse notation by referring by $\delta(z_t - z_{t-1})$ to the Kronecker delta $\delta_{z_t,z_{t-1}}$ for discrete and the Dirac delta distribution for continuous z_t. The joint posterior policy is

$$q_{\phi_i}(a_t, z_t, b_t|s_t, z_{t-1}) = q^T_{\phi_i}(b_t|s_t, z_{t-1})\, q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)\, q^L_{\phi_i}(a_t|s_t, z_t)\,. \tag{4.4}$$

While z_t can be a continuous variable, we consider only $z_t \in \{1 \dots m\}$, where m is the number of available options. The induced distribution $q_{\phi_i}(\tau)$ over trajectories of task i, $\tau = (s_1, b_1, z_1, a_1, s_2, \dots, s_T, b_T, z_T, a_T, s_{T+1})$, is then

$$q_{\phi_i}(\tau) = \rho_i(s_1) \prod_{t=1}^{T} q_{\phi_i}(a_t, z_t, b_t|s_t, z_{t-1})\, P_i(s_{t+1}|s_t, a_t). \tag{4.5}$$

4.3.2 Hierarchical prior policy

Our framework transfers knowledge between tasks by a shared prior $p_\theta(a_t, z_t, b_t|s_t, z_{t-1})$ over all joint policies (4.4):

$$p_\theta(a_t, z_t, b_t|s_t, z_{t-1}) = p^T_\theta(b_t|s_t, z_{t-1})\, p^H(z_t|z_{t-1}, b_t)\, p^L_\theta(a_t|s_t, z_t). \tag{4.6}$$

By choosing p^T_θ, p^H, and p^L_θ correctly, we can learn useful temporally extended options. The parameterized priors $p^T_\theta(b_t|s_t, z_{t-1})$ and $p^L_\theta(a_t|s_t, z_t)$ are structurally equivalent to the posterior policies $q^T_{\phi_i}$ and $q^L_{\phi_i}$ so that they can be used as initialization for the latter on new tasks. Optimizing the regularized return (see

next section) w.r.t. θ distills the common behavior into the prior policy and softly enforces similarity across posterior distributions of each option amongst all tasks i.

The prior $p^H(z_t|z_{t-1}, b_t) = (1 - b_t)\,\delta(z_t - z_{t-1}) + b_t\frac{1}{m}$ selects the previous option z_{t-1} if $b_t = 0$, and otherwise draws options uniformly to ensure exploration. Because the posterior master policy is different on each task, there is no need to distill common behavior into a joint prior.

4.3.3 Objective

We extend the multitask objective in (4.3) by substituting $p_\theta(\tau, \mathcal{O}_{1:T})$ and $p_{\phi_i}(\tau)$ with those induced by our hierarchical posterior policy in (4.4) and the corresponding prior. The resulting objective has the same form but with a new regularized reward that is maximized:

$$R_{i,t}^{\text{reg}} = r_i(s_t, a_t) - \underbrace{\beta \ln \frac{q_{\phi_i}^H(z_t|s_t,z_{t-1},b_t)}{p^H(z_t|z_{t-1},b_t)}}_{①} - \underbrace{\beta \ln \frac{q_{\phi_i}^L(a_t|s_t,z_t)}{p_\theta^L(a_t|s_t,z_t)}}_{②} - \underbrace{\beta \ln \frac{q_{\phi_i}^T(b_t|s_t,z_{t-1})}{p_\theta^T(b_t|s_t,z_{t-1})}}_{③}\,. \tag{4.7}$$

As we maximize $\mathbb{E}_q[R_{i,t}^{\text{reg}}]$, this corresponds to maximizing the expectation over

$$r_i(s_t,a_t) - \beta\Big[\mathbb{D}_{\text{KL}}(q_{\phi_i}^H\|p^H) + \mathbb{D}_{\text{KL}}(q_{\phi_i}^L\|p_\theta^L) + \mathbb{D}_{\text{KL}}(q_{\phi_i}^T\|p_\theta^T)\Big], \tag{4.8}$$

along the on-policy trajectories drawn from $q_{\phi_i}(\tau)$. In the following, we will discuss the effects of all three regularisation terms on the optimisation.

Term ① of the regularisation encourages exploration in the space of options since we chose a uniform prior for p^H when the previous option was terminated. It can *also* be seen as a form of deliberation cost [111] as it is only nonzero whenever we terminate an option and the master policy needs to select another to execute: if the option is not terminated, we have $z_t = z_{t-1}$ with probability 1 for both prior and posterior by construction and $\mathbb{D}_{\text{KL}}(q_{\phi_i}^H\|p^H) = 0$.

Because Eq. (4.7) is optimized across *all* tasks i, term ② updates the prior towards the 'average' posterior. It also regularizes each posterior towards this prior. This enforces similarity between option posteriors across tasks. Importantly, it also encourages the *master* policy to pick the most *specialized* option that still maximizes the return, i.e the option for which the posteriors $q_{\phi_i}^L$ are most similar

across tasks as this will minimize term ②. Consequently, if multiple options have learned the desired behavior, the master policy will only pick the most specialized option consistently. As discussed in Section 4.3.5, this allows us to escape the local optima that hard options face in multitask learning, while still having fully specialized options after training.

Lastly, we can use ③ to also encourage temporal abstraction of options. To do so, *during option learning*, we fix the termination prior p^T to a Bernoulli distribution $p^T(b) = (1-\alpha)^b \alpha^{1-b}$. Choosing a large α encourages prolonged execution of one option, but allows switching whenever necessary. This is similar to deliberation costs [111] but with a more flexible cost model.

We can still distill a termination prior p^T_θ which can be used on future tasks. Instead of learning p^T_θ by minimizing the KL against the posterior termination policies, we can get more decisive terminations by minimizing

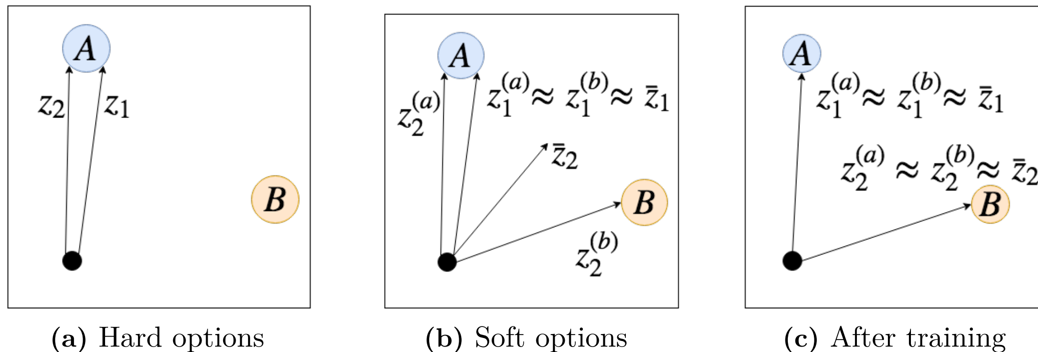
$$\min_\theta \sum_{i=1}^n \mathbb{E}_{\tau\sim q^i}\Big[\mathbb{D}_{\mathrm{KL}}\Big(\hat{q}_{\phi_i}(\cdot|s_t,z_{t-1}) \| \, p^T_\theta(\cdot|s_t,z_{t-1})\Big)\Big], \tag{4.9}$$

and $\hat{q}_{\phi_i}(b=1|s_t,z_{t-1}) = \sum_{z_t \neq z_{t-1}} q^H_{\phi_i}(z_t|s_t,z_{t-1},b_t=1)$ i.e., the learned termination prior distills the probability that the tasks' master policies would change the active option if they had the opportunity. Details on how we optimized the MSOL objective are given in Appendix B.2.

4.3.4 Soft vs. classical options

Assume we are faced with a new task and are given some prior knowledge in the form of a set of skills that we can use. Using the skills' policies and termination probabilities as prior policies p^T and p^L in the soft option framework, we can interpret β as a temperature parameter determining how closely we are required to follow them. For $\beta \to \infty$ we recover the classical "hard" option case and our posterior option policies are restricted to the prior.[2] On the other hand, for $\beta = 0$ the priors only initialise the otherwise unconstrained policy on new tasks. Without the additional regularisation, this can cause the policy to quickly unlearn

[2]However, in this limiting case optimisation using the regularised reward is not possible.



(a) Hard options **(b)** Soft options **(c)** After training

Figure 4.2: Hierarchical learning of two concurrent tasks (a and b) using two options (z_1 and z_2) to reach two relevant targets (A and B). a) Local optimum when simply sharing options across tasks. b) Escaping the local optimum by using prior ($\bar{z}_i$) and posterior ($z_i^{(j)}$) policies. c) Learned options after training. Details are given in the text in Section 4.3.5.

useful behaviours., Only for $0 < \beta < \infty$ MSOL can keep prior information to guide long-term exploration but can also explore policies "close" to them.

4.3.5 Local optima option learning

In this section our aim is to provide an intuitive explanation of why learning hard options in a multitask setting can lead to local optima and how soft options can overcome this. In this local optimum, multiple options have learned the same behavior and are unable to change it, even if doing so would ultimately lead to a higher reward. We use the Moving Bandits experiment schematically depicted in Fig. 4.2 as an example. The agent (black dot) observes two target locations A and B but does not know which one is the correct one that has to be reached in order to generate a reward. The state- and action-spaces are continuous, requiring multiple actions to reach either A or B from the starting position. Consequently, having access to two options, one for each location, can accelerate learning. Experimental results comparing MSOL against a recently proposed 'hard option' method (MLSH, [108]) are discussed in Section 4.5.1.

Let us denote the options we are learning as z_1 and z_2 and further assume that due to random initialization or late discovery of target B, both skills currently reach A. In this situation, the master policies on tasks in which the correct goal is A are indifferent between using z_1 and z_2 and will consequently use *both* with

equal probability since we typically apply at least some entropy regularisation during training to avoid premature convergence.

In the case of hard options, changing one skill, e.g. z_2, towards B in order to solve tasks in which B is the correct target, decreases the performance on all tasks that currently use z_2 to reach target A, because for hard options the skills are shared exactly across tasks. Averaged across all tasks, this would at first *decrease* the overall average return, preventing any option from changing away from A, leaving B unreachable and training stuck in a local optimum.

To "free up" z_2 and learn a new skill reaching B, all master policies need to refrain from using z_2 to reach A and instead use the equally useful skill z_1 exclusively. Importantly, using soft options makes this possible. In Figures 4.2b and 4.2c we depict this schematically. The key difference is that in MSOL we have separate *task-specific posteriors* $z_i^{(a)}$ and $z_i^{(b)}$ for tasks a and b and soft options $i \in \{1, 2\}$ (for simplicity, we assume that the correct target is A for task a and B for task b). This allows us, in a first step, to solve *all* tasks (Fig. 4.2b): despite master policies on tasks a still using posterior $z_2^{(a)}$ to reach A, the other posterior $z_2^{(b)}$ can learn to reach B. However, this now makes option z_2 less specialized across tasks, i.e. the prior $\bar{z}_2$ does not agree with either posterior $z_2^{(a)/(b)}$. Consequently, for tasks a, the master policies will now strictly prefer option z_1 to reach A, allowing option z_2 to specialize on only reaching B, leading to the situation shown in Fig. 4.2c in which both options specialize to reach different targets.

4.4 Related work

Most hierarchical approaches rely on proxy rewards to train the lower level components and their terminations. Some of them aim to reach pre-specified subgoals [102], which are often found by analyzing the structure of the MDP [112], previously learned policies [113] or predictability [114]. Those methods typically require knowledge, or a sufficient approximation, of the transition model, both of which are often infeasible.

Recently, several authors have proposed unsupervised training objectives for learning diverse skills based on their distinctiveness [105]. However, those approaches

don't learn termination functions and cannot guarantee that the required behavior on the downstream task is included in the set of learned skills. Hausman et al. [115] also incorporate reward information, but do not learn termination policies and are therefore restricted to learning multiple solutions to the provided task instead of learning a *decomposition* of the task solutions which can be re-composed to solve new tasks.

A third usage of proxy rewards is by training lower level policies to move towards goals defined by the higher levels. When those goals are set in the original state space [107], this approach has difficulty scaling to high dimensional state spaces like images. Setting the goals in a learned embedding space [106] can be difficult to train, though. In both cases, the temporal extension of the learned skills are set manually. On the other hand, Goyal et al. [116] also learn a hierarchical agent, but not to transfer skills, but to find decisions states based on how much information is encoded in the latent layer.

HiREPS [117] also take an inference motivated approach to learning options. In particular Daniel et al. [118] propose a similarly structured hierarchical policy, albeit in a single task setting. However, they do not utilize learned prior *and* posterior distributions, but instead use expectation maximization to iteratively infer a hierarchical policy to explain the current reward-weighted trajectory distribution.

Several previous works try to overcome the restrictive nature of options that can lead to sub-optimal solutions by allowing the higher-level actions to modulate the behavior of the lower-level policies Heess et al. [119] and Haarnoja et al. [120]. However, this significantly increases the required complexity of the higher-level policy and therefore the learning time.

The multitask- and transfer-learning setup used in this chapter is inspired by Thrun and Schwartz [121] who suggests extracting options by using commonalities between solutions to multiple tasks. Prior multitask approaches often rely on additional human supervision like policy sketches [122] or desirable sub-goals [113] in order to learn skills which transfer well between tasks. In contrast, our work aims at finding good termination states without such supervision. Tirumala et

al. [123] investigate the use of different priors for the higher-level policy while we are focussing on learning transferrable option priors. Closest to our work is MLSH [108] which, however, shares the lower-level policies across all tasks without distinguishing between prior and posterior and does not learn termination policies. As discussed, this leads to local minima and insufficient diversity in the learned options. Similarly to us, Fox, Moshkovitz, and Tishby [124] differentiate between prior and posterior policies on multiple tasks and utilize a KL-divergence between them for training. However, they do not consider termination probabilities and instead only choose one option per task.

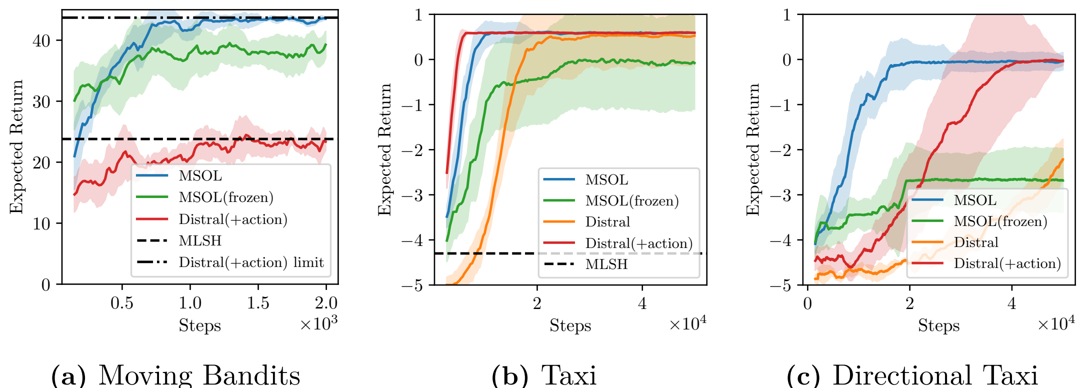
Our approach is closely related to DISTRAL [101] with which we share the multitask learning of prior and posterior policies. However, DISTRAL has no hierarchical structure and applies the same prior distribution over primitive actions, independent of the task. As a necessary hierarchical heuristic, the authors propose to also condition on the last primitive action taken. This works well when the last action is indicative of future behavior; however, in Section 4.5 we show several failure cases where a *learned* hierarchy is needed.

Compared to recent meta-learning approaches also focussing on fine-tuning on new tasks (e.g. [125, 126]), the primary advantage of MSOL lies in better exploration, which has been shown to be often challenging for meta-learning [127].

4.5 Experiments

We conduct a series of experiments to show: i) MSOL trains successfully without complex training schedules like in MLSH [108], ii) MSOL can learn useful termination policies, iii) when learning hierarchies in a multitask setting, unlike other methods, MSOL successfully overcomes the local minimum of insufficient option diversity, as described in Section 4.3.5, iv) using soft options yields fast transfer learning while still reaching optimal performance, even on new, out-of-distribution tasks.

All architectural details and hyper-parameters can be found in Appendix B. For all experiments, we first train the exploration priors and options on n tasks from the available task distribution $\mathcal{T}$ (training phase is plotted in Appendix



(a) Moving Bandits **(b)** Taxi **(c)** Directional Taxi

Figure 4.3: Performance of applying the learned options and exploration priors to new tasks. Each line is the median over 5 random seeds (2 for MLSH) and shaded areas indicated standard deviations. Performance during the training phase is shown in Fig. B.1. *Moving Bandits* (a) is a simple environment capturing the effects described in Section 4.3.5. The results show that MLSH, which uses hard options, struggles with local minima during the learning phase, whereas MSOL is able to learn a diverse set of options. *Taxi* (b) and *Directional Taxi* (c) additionally require good termination policies, which MLSH cannot learn as it uses a fixed option duration. See Fig. 4.4 for a visualization of the options and terminations learned by MSOL. DISTRAL(+action) is a strong non-hierarchical baseline which uses the last action as option-heuristic, but suffers when that action is not very informative, for example in (c).

B.4). Subsequently, we test how quickly we can learn new tasks from $\mathcal{T}$ (or another distribution $\mathcal{T}'$).

We compare the following algorithms: MSOL is our proposed method that utilizes soft options both during option learning and transfer. MSOL(frozen) uses the soft options framework during learning to find more diverse skills, but does not allow fine-tuning the posterior sub-policies after transfer. DISTRAL [101] is a strong non-hierarchical transfer learning algorithm that also utilizes prior and posterior distributions. DISTRAL(+action) utilizes the last action as option-heuristic, that is, as additional input to the policy and prior, which works well in some tasks but fails when the last action is not sufficiently informative. Conditioning on an *informative* last action allows the DISTRAL prior to learn temporally correlated exploration strategies. MLSH [108] is a multitask option learning algorithm like MSOL, but utilizes ‘hard’ options for both learning and transfer, i.e., sub-policies that are shared exactly across tasks. It relies on fixed option durations and requires a complex training

schedule between master and intra-option policies to stabilize training. We use the author's MLSH implementation. We also compare against OC [104], which takes the task-id as additional input in order to apply it to multiple tasks.

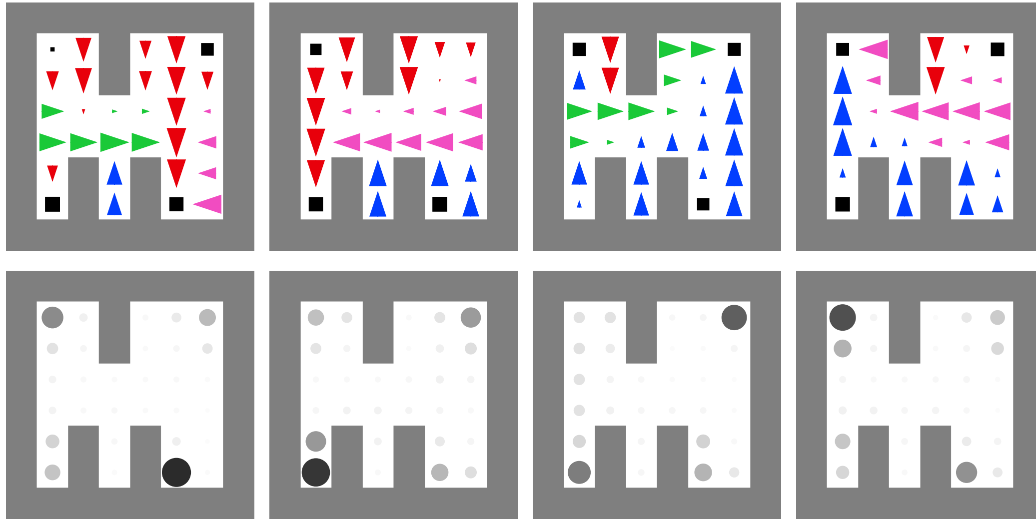
Note that, during test time, MLSH and MSOL(frozen) can be fairly compared as each uses one fixed policy per skill. On the other hand, DISTRAL, DISTRAL(+action) and MSOL use adaptive posterior policies for each task and are consequently more expressive.

4.5.1 Moving bandits

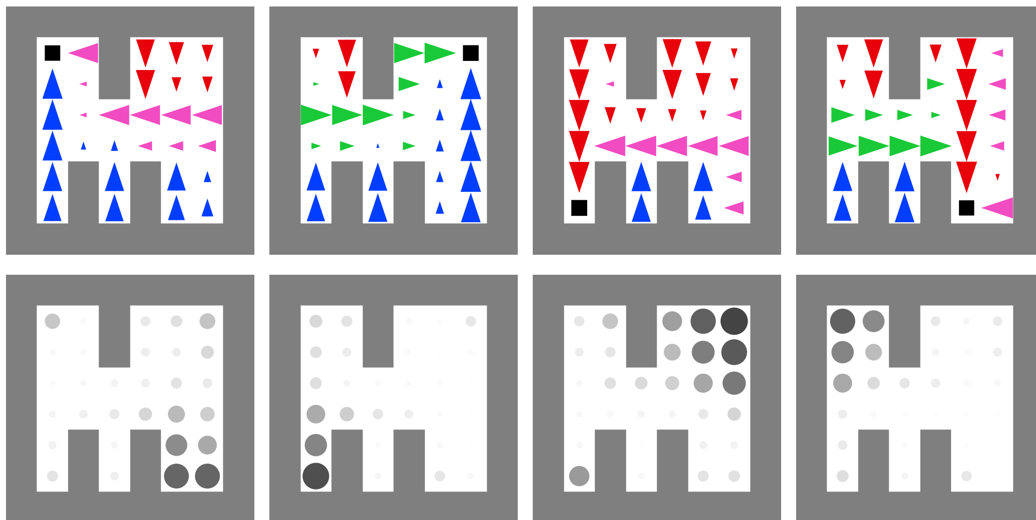
We start with the 2D Moving Bandits environment proposed and implemented by Frans et al. [108], which is similar to the example in Section 4.3.5. There are two randomly sampled, distinguishable, marked positions in the environment. In each episode, the agent receives a reward of 1 for each time step it is sufficiently close to the correct one of both positions, and 0 otherwise. Which location is rewarded is not signaled in the observation. The agent can take actions that move it in one of the four cardinal directions. Each episode lasts 50 steps.

We compare against MLSH and DISTRAL to highlight challenges that arise in multitask training. We allow MLSH and MSOL to learn two options. During transfer, optimal performance can only be achieved with diverse options that have successfully learned to reach *different* marked locations. In Fig. 4.3a we can see that MSOL is able to do so but the hard options learned by MLSH both learned to reach the *same* goal location, resulting in only approximately half the optimal return during transfer. This is exactly the situation outlined in Section 4.3.5 in which learning hard options can lead to local optima.

DISTRAL, even with the last action provided as additional input, is not able to quickly utilize the prior knowledge. The last action only conveys meaningful information when taking the goal locations into account: DISTRAL agents need to *infer* the intention based on the last action and the relative goal positions. While this is possible, in practice the agent was not able to do so, even with a much larger network. Much longer training ultimately allows DISTRAL to perform as



(a) Options before pickup.



(b) Options after pickup.

Figure 4.4: Options learned with MSOL on the taxi domain, one option per column, before (top) and after pickup (bottom). The light gray area indicates walls. The colored arrows indicate the direction of the most likely action and their size indicates its probability. A square indicates the pickup/dropoff action. On the other hand, the intensity and size of the grey circles indicate the option-termination probability.

well as MSOL, denoted by "DISTRAL(+action) limit". This is not surprising since its posterior is flexible and will therefore eventually be able to learn any task. However, it is not able to learn transferrable prior knowledge which allows *fast* training on the new task. Lastly, MSOL(frozen) also outperforms DISTRAL(+action)

and MLSH, but performs worse that MSOL. This highlights the utility of making options soft, i.e. adaptable, during transfer to new tasks. It also shows that the advantage of MSOL over the other methods lies not only in its flexibility during transfer, but also during the original learning phase.

4.5.2 Taxi

Next, we use a slightly modified version of the original Taxi domain [128] to show learning of termination functions as well as transfer- and generalization capabilities. To solve the task, the agent must pick up a passenger on one of four possible locations by moving to their location and executing a special 'pickup/drop-off' action. Then, the passenger must be dropped off at one of the other three locations, again using the same action executed at the corresponding location. The domain has a discrete state space with 30 locations arranged on a grid and a flag indicating whether the passenger was already picked up. The observation is a one-hot encoding of the discrete state, excluding passenger- and goal location. This introduces an information-asymmetry between the task-specific master policy, and the shared options, allowing them to generalize well [129]. Walls (see Fig. 4.4) limit the movement of the agent and invalid actions.

We investigate two versions of Taxi. In the original, just called *Taxi*, the action space consists of one no-op, one 'pickup/drop-off' action and four actions to move in all cardinal directions. In *Directional Taxi*, we extend this setup: the agent faces in one of the cardinal directions and the available movements are to move forward or rotate either clockwise or counter-clockwise. In both environments the set of tasks $\mathcal{T}$ are the 12 different combinations of pickup/drop-off locations. Episodes last at most 50 steps and there is a reward of 2 for delivering the passenger to its goal and a penalty of -0.1 for each time step. During training, the agent is initialized to any valid state. During testing, the agent is always initialized without the passenger on board.

We allow four learnable options in MLSH and MSOL. This necessitates the options to be diverse, i.e., one option to reach each of the four pickup/drop-off locations. Importantly, it also requires the options to learn to terminate when a passenger is

picked up. As one can see in Fig. 4.3b, MLSH struggles both with option-diversity and due to its fixed option duration: because the starting position is random, the duration until the option needs to terminate is different between episodes and cannot be captured by one hyperparameter. Furthermore, even without correct terminations, one could still learn to solve (at least) four out of the twelve tasks, leading to an average reward of approximately -3.2[3]. However, MLSH is not able to learn diverse enough policies, resulting in worse performance.

DISTRAL(+action) performs well in the original *Taxi* environment, as seen in Fig. 4.3b. This is expected since here the last action, moving in a compass direction, is a good indicator for the agent's intention, effectively acting as an optimal "option" and inducing temporally extended exploration. However, in the directional case shown in Fig. 4.3c, actions rarely indicate intentions, which makes it much harder for DISTRAL(+action) to use prior knowledge. By contrast, MSOL performs well in both taxi environments. In the directional case, learned MSOL options capture temporally correlated behavior much better than the last action in DISTRAL.

Fig. 4.4 demonstrates that the options learned by MSOL learn movement and termination policies that make intuitive sense. Note that the same soft option represents different behavior depending on whether it already picked up the passenger, as this behavior does not need to terminate the current option on three of the 12 tasks.

4.5.3 Out-of-distribution tasks

In this section, we show how learning soft options can help with transfer to unseen tasks. These experiments were contributed by Jinke He. In Fig. 4.5a we show learning on four tasks from $\mathcal{T}$ using options that were trained on the remaining eight, comparing against advantage actor-critic (A2C) [54] and Option Critic (OC) [104]. Note that in OC, there is no information-asymmetry: the same networks are shared across all tasks and provided with a task-id as additional input, including to the option-policies. This prevents OC from generalizing well

[3] The optimal policy for a task achieves approximate a return of 0.5 on average whereas the worst possible return is -5.

to unseen tasks. On the other hand, withholding the task-information would be similar to MLSH, which we already showed to struggle with local minima. The strong performance of MSOL shows that information-asymmetric options help to generalize to previously unseen tasks.

We also investigate the utility of flexible soft options under a shift of the task distribution: in Figs. 4.5b and 4.5c we show learning performance on twelve *modified* tasks in which the pickup/dropoff locations where moved by one cell while the options were trained with the original locations. While the results in Fig. 4.5b use a smaller grid, Fig. 4.5c shows the results for a larger grid in which exploration is more difficult. As expected, hard options are not able to solve this task for either grid-size. Moreover, while combining hard options with primitive actions allows the tasks to be solved eventually, it performs worse than training a new, flat policy from scratch. The finding that access to misspecified, hard options can actually *hurt* exploration is consistent with previous literature [130]. On the other hand, MSOL is able to quickly learn on this new task by adapting the previously learned options.

Note that on the small grid in which exploration is easy, our hierarchical method performs similar to a flat policy. On the larger grid exploration becomes more challenging and MSOL learns significantly faster, highlighting how transfer learning can improve exploration. More results can be found in Appendix B.4.2.

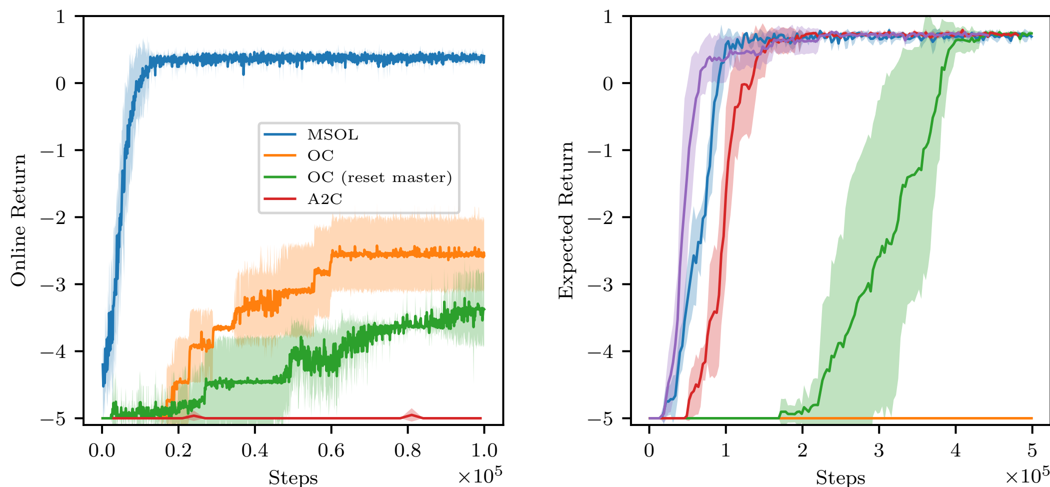
4.6 Discussion

Multitask Soft Option Learning (MSOL) proposes reformulating options using the perspective of prior and posterior distributions. This offers several key advantages. First, during transfer, it allows us to distinguish between fixed, and therefore knowledge-preserving option *priors*, and flexible option *posteriors* that can adjust to the reward structure of the task at hand. This effects a similar speed-up in learning as the original options framework, while avoiding sub-optimal performance when the available options are not perfectly aligned to the task. Second, utilizing this 'soft' version of options in a multitask learning setup increases optimisation stability and removes the need for complex training schedules. Furthermore, this

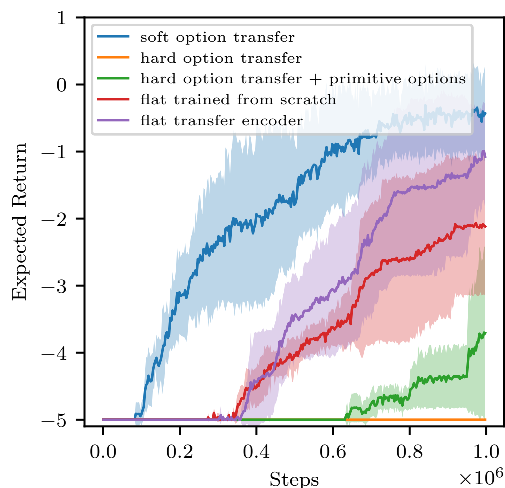
framework naturally allows master policies to coordinate across tasks and avoid local minima of insufficient option diversity. It also allows for autonomously learning option-termination policies, a very challenging task which is often avoided by fixing option durations manually.

Lastly, using this formulation also allows inclusion of prior information in a principled manner without imposing too rigid a structure on the resulting hierarchy. We utilize this advantage to explicitly incorporate the bias that good options should be temporally extended. In future research, other types of information can be explored. As an example, one could investigate sets of tasks which would benefit from a learned master prior, like walking on different types of terrain.

In this part of the thesis, we have presented two contributions, DVRL and MSOL, which improve performance by imposing additional structure on the agent architecture and training algorithm. In fact, in this chapter, we have even used two separate inductive biases: A *hierarchical* policy, and a prior/posterior split inducing (soft) *regularisation* on the policy. In the next part of this thesis, we will explore regularisation and generalisation more specifically, first by exploring how many existing, stochastic regularisation methods can be applied to RL to improve generalisation and second by investigating another source of poor generalisation in RL, namely transient non-stationarities induced by training algorithms.



(a) Taxi: Generalization **(b)** Taxi (small): Adaptation



(c) Taxi (large): Adaptation

Figure 4.5: We compare MSOL against Option Critic (OC), hard options and flat policies trained from scratch or with a pre-trained encoder. For a fair comparison, the soft option prior is identical to the hard option in these experiments. *Left:* Since the options in OC are not task-agnostic, they fail to generalize to previously unseen tasks. *Middle and right:* Transfer performance of options to environments in which the pickup and dropoff locations where shifted, making the options misspecified. Only soft options provide utility over flat policies in this setting. The middle figure shows results on a small grid in which exploration is simple, whereas the right figure shows that transfer learning can accelerate exploration especially on larger tasks.

Part II

Generalisation in Reinforcement Learning

Abstract

In the previous chapters, we focussed on improving architectures and training methods in RL by imposing a suitable structure on the agent. In contrast, in this part, we will focus on *regularisation* as a means to improve performance in domains which require a large degree of generalisation.

Many environments used in deep RL research only exhibit a limited amount of stochasticity [e.g. 97, 131], thereby effectively only requiring the agent to generalise in narrow 'corridors' of states within which optimal agents remain. Recently, on the other hand, several procedurally generated environments have been proposed [e.g. 132, 133] in order to introduce a significantly larger amount of variation in the encountered states. Improving performance on these types of environments will greatly help to improve performance in many real-world applications with equally high variability. Furthermore, separating designated *train* and *test* levels allows us to explicitly measure the generalisation capabilities of our agents, facilitating progress towards more robust policies.

In Chapter 5 we first explore stochastic regularisation in online RL methods such as PPO. These types of regularisation, like the Variational Information Bottleneck [134] or Dropout [135], rely on the injection of stochasticity into the network during training and have proven to be highly effective in supervised learning. We explore how this noise injection can interfere with the data collection in online RL methods and propose a solution to improve performance.

In Chapter 6 we investigate generalisation in deep RL on a more fundamental level by exploring, through a range of experiments, how transient non-stationarity in the training data can interfere with the stochastic gradient training of neural networks and can bias them towards solutions with worse generalisation properties. Many state of the art RL algorithms introduce these types of non-stationarity into the training since they are using a continuously changing policy for data collection. We propose a novel framework to reduce the non-stationarity experienced by the trained policy, which thereby allows for improved generalisation.

5

Generalization in Reinforcement Learning with Selective Noise Injection

Contents

5.1 Introduction **72**
5.2 Background **74**
5.2.1 Regularislarisation Techniques in Supervised Learning . 74
5.3 The Problem of Using Stochastic Regularisation in RL 76
5.4 Method **77**
5.4.1 Selective Noise Injection 77
5.4.2 Information Bottleneck Actor Critic 79
5.5 Experiments **81**
5.5.1 Learning Features in the Low-Data Regime 81
5.5.2 Multiroom 82
5.5.3 Coinrun 84
5.6 Related Work **86**
5.7 Conclusion **88**

5.1 Introduction

In the previous chapters, we have seen that deep RL can successfully be applied to a range of challenging tasks, including Atari [see also 44, 97, 136] or continuous control [39, 137]. Recently, even more complex domains with long-ranged temporal dependencies have been solved [24, 138]. In those settings, the challenge is to

be able to successfully explore and learn policies complex enough to solve the training tasks. Consequently, the focus of these works was to improve the learning performance of agents in the training environment and less attention was being paid to generalisation to (unseen) testing environments.

However, being able to generalise is a key requirement for the broad application of autonomous agents. Spurred by several recent works showing that most RL agents overfit to the training environment [139–143], multiple benchmarks to evaluate the generalisation capabilities of agents were proposed, typically by procedurally generating or modifying levels in video games [144–149]. How to learn generalisable policies in these environments remains an open question, but early results have shown the use of regularisation techniques (like weight decay, dropout and batch normalisation) established in the supervised learning paradigm can also be useful for RL agents [145]. This chapter builds on these results, but highlights two important differences between supervised learning and RL which need to be taken into account when regularising agents.

First, because in RL the *training data depends on the model* and, consequently, the regularisation method, stochastic regularisation techniques like Dropout or BatchNorm can have adverse effects. For example, injecting stochasticity into the policy can lead to prematurely ending episodes, preventing the agent from observing future rewards. Furthermore, stochastic regularisation can destabilize training through the learned *critic* and off-policy importance weights. To mitigate those adverse effects and effectively apply stochastic regularisation techniques to RL, we propose SNI. It selectively applies stochasticity only when it serves regularisation and otherwise computes the output of the regularised networks deterministically. We focus our evaluation on Dropout and the Variational Information Bottleneck (VIB), but the proposed method is applicable to most forms of stochastic regularisation.

A second difference between RL and supervised learning is the *non-stationarity of the data-distribution* in RL. Despite many RL algorithms utilizing millions or even billions of observations, the diversity of states encountered early on in training can be small, making it difficult to learn general features. While it remains an

open question as to why deep neural networks generalise despite being able to perfectly memorize the training data [21, 150], it has been shown that the optimal point on the worst-case generalisation bound requires the model to rely on a more compressed set of features the fewer data-points we have [151, 152]. Therefore, to bias our agent towards more general features even early on in training, we adapt the Information Bottleneck (IB) principle to an actor-critic agent, which we call IBAC. In contrast to other regularisation techniques, IBAC directly incentivizes the compression of input features, resulting in features that are more robust under a shifting data-distribution and that enable better generalisation to held-out test environments. In this chapter, we will be focussing on applying regularisation to the agent and will explore the effects of non-stationarity in more depth in Chapter 6.

We evaluate our proposed techniques using PPO, an off-policy actor-critic algorithm, on two challenging generalisation tasks, *Multiroom* [133] and *Coinrun* [145]. We show the benefits of both IBAC and SNI individually as well as in combination, with the resulting IBAC-SNI significantly outperforming the previous state of the art results.

5.2 Background

Although any RL method with an off-policy correction term could be used with our proposed method of SNI, PPO (see Section 2.3.3) has shown strong performance and enables direct comparison with prior work [145].

In the following, we discuss regularisation techniques from supervised learning which can be used to mitigate overfitting.

5.2.1 Regularislarisation Techniques in Supervised Learning

In supervised learning, classifiers are often regularised using a variety of techniques to prevent overfitting. Here, we briefly present several major approaches which we either utilize as baseline or extend to RL in Section 5.4.

Weight decay, also called L2 regularisation, reduces the magnitude of the weights θ by adding an additional loss term $\lambda_w \frac{1}{2}\|\theta\|_2^2$. With a gradient update of the form $\theta \leftarrow \theta - \alpha\nabla_\theta(L(\theta) + \frac{\lambda_w}{2}\|\theta\|_2^2)$, this decays the weights in addition to optimizing $L(\theta)$, i.e. we have $\theta \leftarrow (1 - \alpha\lambda_w)\theta - \alpha\nabla_\theta L(\theta)$.

Data augmentation refers to changing or distorting the available input data to improve generalisation. In this work, we use a modified version of cutout [153], proposed by [145], in which a random number of rectangular areas in the input image is filled by random colors.

Batch Normalization [154, 155] normalizes activations of specified layers by estimating their mean and variance using the current mini-batch. Estimating the batch statistics introduces noise which has been shown to help improve generalisation [156] in supervised learning.

Another widely used regularisation technique for deep neural networks is *Dropout* [135]. Here, during training, individual activations are randomly zeroed out with a fixed probability p_d. This serves to prevent co-adaptation of neurons and can be applied to any layer inside the network. One common choice, which we are following in our architecture, is to apply it to the last hidden layer.

Lastly, we will briefly describe the Variational Information Bottleneck (VIB) [134], a deep variational approximation to the Information Bottleneck (IB) [157]. While not typically used for regularisation in deep supervised learning, we demonstrate in Section 5.5 that our adaptation IBAC shows strong performance in RL. Given a data distribution $p(X, Y)$, the learned model $p_\theta(Y|X)$ is regularised by inserting a stochastic latent variable Z and minimizing the mutual information between the input X and Z, $I(X, Z)$, while maximizing the predictive power of the latent variable, i.e. $I(Z, Y)$. The VIB objective function is:

$$L_{\text{VIB}} = \mathbb{E}_{p(X,Y),p_\theta(Z|X)}\Big[- \log q_\theta(Y|Z) + \beta D_{KL}[p_\theta(Z|X)\|q(Z)]\Big] \tag{5.1}$$

where $p_\theta(Z|X)$ is the encoder, $q_\theta(Y|Z)$ the decoder, $q(Z)$ the approximated latent marginal often fixed to a normal distribution $\mathcal{N}(0, I)$ and β is a hyperparameter. For a normal distributed $p_\theta(Z|X)$, Eq. (5.1) can be optimized by gradient decent using the reparameterization trick [37].

5.3 The Problem of Using Stochastic Regularisation in RL

We now take a closer look at a prototypical objective for training actor-critic methods and highlight important differences to supervised learning. Based on those observations, we propose an explanation for the finding that some stochastic optimisation methods are less effective [145] or can even be detrimental to performance when combined with other regularisation techniques (see Appendix C.4).

Intuitively, the problem arises because in online RL, unlike in supervised learning, the current policy is not only updated but also used to collect new data. Consequently, employing regularisation techniques like dropout can help improve generalisation on the one hand, but also deteriorate the quality of the collected data on the other hand, for example through terminating the episode early by means of sub-optimal actions. In Section 5.4 we will propose a heuristic approach which balances these two considerations, as well as the arising off-policy correction term when a different policy than the current one is used for data collection.

In supervised learning, the optimisation objective takes a form similar to $\max_\theta \mathbb{E}_{\mathcal{D}} [\log p_\theta(y|x)]$, where we highlight the model $p_\theta(y|x)$ to be updated in blue, $\mathcal{D}$ is the available data and θ the parameters to be learned. On the other hand, in RL the objective for the actor is to maximize $J(\pi_\theta) = \mathbb{E}_{\pi_\theta(a|s)} \left[\sum_t \gamma^t R(s_t, a_t)\right]$, where, for convenience, we drop the initial-state- and transition-distribution P_0 and P from the notation of the expectation. Because now the learned distribution, $\pi_\theta(a|s)$, is part of data-generation, computing the gradients, as done in policy gradient methods, requires the log-derivative trick. For the class of deep off-policy actor-critic methods we are experimentally evaluating in this chapter, one also typically uses the policy gradient theorem [32] and an estimated *critic* $V_\theta(s)$ as baseline and for bootstrapping to reduce the gradient variance. Consequently, the gradient estimation becomes:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi^r_\theta(a_t|s_t)} \left[\sum_t^T \frac{\pi_\theta(a_t|s_t)}{\pi^r_\theta(a_t|s_t)} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)) \right] \quad (5.2)$$

where we utilize a *rollout policy* π^r_θ to collect trajectories. It can deviate from π_θ but should be similar to keep the off-policy correction term $\pi_\theta / \pi^r_\theta$ low variance. In

Eq. (5.2), only the term $\pi_\theta(a_t|s_t)$ is being updated and we highlight in orange all the additional influences of the learned policy and critic on the gradient.

Denoting by the superscript $\perp$ that $V_\theta^\perp$ is assumed constant, we can write the optimisation objective for the critic as

$$L_{\text{AC}}^V = \min_\theta \mathbb{E}_{\pi_\theta^r(a_t|s_t)}\left[\left(\gamma V_\theta^\perp(s_{t+1}) + r_t - V_\theta(s_t)\right)^2\right], \tag{5.3}$$

such that V_θ is consequently approximating the expected future return of π_θ^r. From Eqs. (5.2) and (5.3) we can see that the injection of noise into the computation of π_θ^r and V_θ can degrade performance in several ways: i) During rollouts using the rollout policy π_θ^r, it can lead to undesirable actions, potentially ending episodes prematurely, and thereby deteriorating the quality of the observed data; ii) It leads to a higher variance of the off-policy correction term π_θ/π_θ^r because the injected noise can be different for π_θ and π_θ^r, increasing gradient variance; iii) It increases variance in the gradient updates of both the policy and the critic through variance in the computation of V_θ.

5.4 Method

To utilize the strength of noise-injecting regularisation techniques in RL, we introduce Selective Noise Injection (SNI) in the following section. Its goal is to allow us to make use of such techniques while mitigating the adverse effects the added stochasticity can have on the RL gradient computation. Then, in Section 5.4.2, we propose Information Bottleneck Actor Critic (IBAC) as a new regularisation method and detail how SNI applies to IBAC, resulting in IBAC-SNI.

5.4.1 Selective Noise Injection

We have identified three sources of negative effects due to noise which we need to mitigate: In the rollout policy π_θ^r, in the critic V_θ and in the off-policy correction term π_θ/π_θ^r. We first introduce a short notation for Eq. (5.2) as $\nabla_\theta J(\pi_\theta) = \mathcal{G}_{\text{AC}}(\pi_\theta^r, \pi_\theta, V_\theta)$.

To apply SNI to a regularisation technique relying on noise-injection, we need to be able to *temporarily suspend* the noise and compute the output of the model

deterministically. This is possible for most techniques[1]: For example, in Dropout, we can freeze one particular dropout mask, in VIB we can pass in the mode instead of sampling from the posterior distribution and in Batch Normalization we can either utilize the moving average instead of the batch statistics or freeze and re-use one statistic multiple times. Formally, we denote by $\bar{\pi}_\theta$ the version of a component π_θ, with the injected regularisation noise suspended. Note that this does not mean that $\bar{\pi}_\theta$ is deterministic, for example when the network approximates the parameters of a distribution.

Then, for SNI we modify the policy gradient loss as follows: i) We use $\bar{V}_\theta$ as critic instead of V_θ in both Eqs. (5.2) and (5.3), eliminating unnecessary noise through the critic; ii) We use $\bar{\pi}^r$ as rollout policy instead of π^r. For some regularisation techniques this will reduce the probability of undesirable actions; iii) We compute the policy gradient as a *mixture* between gradients for π_θ and $\bar{\pi}_\theta$ as follows:

$$\mathcal{G}^{\text{SNI}}_{\text{AC}}(\pi^r_\theta, \pi_\theta, V_\theta) = \lambda \mathcal{G}_{\text{AC}}(\bar{\pi}^r_\theta, \bar{\pi}_\theta, \bar{V}_\theta) + (1-\lambda)\mathcal{G}_{\text{AC}}(\bar{\pi}^r_\theta, \pi_\theta, \bar{V}_\theta) \tag{5.4}$$

The first term guarantees a lower variance of the off-policy importance weight, which is especially important early on in training when the network has not yet learned to compensate for the injected noise. The second term uses the noise-injected policy for updates, thereby taking advantage of its regularising effects while still reducing unnecessary variance through the use of $\bar{\pi}^r$ and $\bar{V}_\theta$. Note that sharing the rollout policy $\bar{\pi}^r$ between both terms allows us to use the same collected data. Furthermore most computations are shared between both terms or can be parallelized. Another way to see SNI is to acknowledge that stochastic regularisation in RL can have a negative impact on the performance of the policy and Eq. (5.4) allows 'selectively' introducing such additional noise in a way which mitigates its adverse effects while still allowing some of the beneficial regularising effect improving generalisation.

[1] In this work, we will focus on VIB and Dropout as those show the most promising results without SNI (see Section 5.5) and will leave its application to other regularisation techniques for future work.

5.4.2 Information Bottleneck Actor Critic

Early on in training an RL agent, we are often faced with little variation in the training data. Observed states are distributed only around the initial states s_0, making spurious correlations in the low amount of data more likely. Furthermore, because neither the policy nor the critic have sufficiently converged yet, we have a high variance in the target values of our loss function.

This combination makes it harder and less likely for the network to learn desirable features that are robust under a shifting data-distribution during training and regularise well to held-out test MDPs. To counteract this reduced signal-to-noise ratio, our goal is to explicitly bias the learning towards finding more *compressed* features which are shown to have a tighter worst-case generalisation bound [152]. While a higher compression does not guarantee robustness under a *shifting* data-distribution, we believe this to be a reasonable assumption in the majority of MDPs, for example because they rely on a consistent underlying transition mechanism like physical laws.

To incentivize more compressed features, we use an approach similar to the VIB [134], which minimizes the mutual information $I(S,Z)$ between the state S and its latent representation Z while maximizing $I(Z,A)$, the predictive power of Z on actions A[2]. To do so, we re-interpret the policy gradient update as maximization of the log-marginal likelihood of $\pi_\theta(A|S)$ under the data distribution $p(S,A) := -\frac{\rho^\pi(S)\pi_\theta(A|S)A^\pi(S,A)}{\mathcal{Z}}$ with discounted state distribution $\rho^\pi(S)$, advantage function $A^\pi(S,A)$ and normalisation constant $\mathcal{Z}$. Taking the gradient of this objective while assuming $p(S,A)$ to be fixed, recovers the policy gradient:

$$\nabla_\theta \mathcal{Z}\, \mathbb{E}_{p(S,A)}[\log \pi_\theta(A|S)] = \int \rho^\pi(s)\pi_\theta(a|s)\nabla_\theta \log \pi_\theta(a|s) A^\pi(s,a)\, \mathrm{d}s\, \mathrm{d}a. \tag{5.5}$$

Now, following the same steps as [134], we introduce a stochastic latent variable Z and minimize $\beta I(S,Z)$ while maximizing $I(Z,A)$ under $p(S,A)$, resulting in

[2] In this section, as we will be dealing with information-theoretic quantities we will denote by A the random variable of the policy action, and denote the advantage explictly by A^π.

the new objective:

$$L_{\text{IB}} = \mathbb{E}_{p(S,A),p_\theta(Z|S)}\Big[-\log q_\theta(A|Z) + \beta D_{KL}[p_\theta(Z|S)\|q(Z)]\Big] \tag{5.6}$$

We take the gradient and use the reparameterization trick [37] to write the encoder $p_\theta(Z|S)$ as deterministic function $f_\theta(S,\epsilon)$ with $\epsilon \sim p(\epsilon)$:

$$\begin{aligned}\nabla_\theta L_{\text{IB}} &= -\mathbb{E}_{\rho^\pi(S)\pi_\theta(A|S)p(\epsilon)}\left[\nabla_\theta \log q_\theta(A|f_\theta(S,\epsilon))A^\pi(S,A)\right] + \nabla_\theta \beta D_{KL}[p_\theta(Z|S)\|q(Z)] \\ &= \nabla_\theta(L^{\text{IB}}_{\text{AC}} + \beta L^{\text{KL}}),\end{aligned} \tag{5.7}$$

resulting in a modified policy gradient objective and an additional regularisation term L^{KL}.

Policy gradient algorithms heuristically add an entropy bonus $H[\pi_\theta(A|S)]$ to prevent the policy distribution from collapsing. However, this term also influences the distributions over Z. In practice, we are only interested in preventing $q_\theta(A|Z)$ (not $\pi_\theta(A|S) = \mathbb{E}_Z[q_\theta(A|Z)]$) from collapsing because our rollout policy $\bar{\pi}_\theta$ will not rely on stochasticity in Z. Additionally, $p_\theta(Z|S)$ is already entropy-regularised by the IB loss term[3]. Consequently, we adapt the heuristic entropy bonus to

$$H^{\text{IB}}[\pi_\theta(A|S)] := \int p_\theta(s,z) H[q_\theta(A|z)]\,\mathrm{d}s\,\mathrm{d}z, \tag{5.8}$$

resulting in the overall loss function of the proposed Information Bottleneck Actor Critic (IBAC)

$$L^{\text{IBAC}}_t(\theta) = L^{\text{IB}}_{\text{AC}} + \lambda_V L^V_{\text{AC}} - \lambda_H H^{\text{IB}}[\pi_\theta] + \beta L^{\text{KL}} \tag{5.9}$$

with the hyperparameters λ_V, λ_H and β balancing the loss terms.

While IBAC incentivizes more compressed features, it also introduces stochasticity. Consequently, combining it with SNI improves performance, as we demonstrate in Sections 5.5.2 and 5.5.3. To compute the noise-suspended policy $\bar{\pi}_\theta$ and critic $\bar{V}_\theta$, we use the mode $Z = \mu_\theta(S)$ as input to $q_\theta(A|Z)$ and $V_\theta(Z)$, where $\mu_\theta(S)$ is the mode of $p_\theta(Z|S)$ and $V_\theta(Z)$ now conditions on Z instead of S, also using the compressed

[3] We have $D_{\text{KL}}[p_\theta(Z|s)\|r(Z)] = \mathbb{E}_{p_\theta(Z|s)}[\log p_\theta(Z|s) - \log r(Z)] = -H[p_\theta(Z|s)] - \mathbb{E}_{p_\theta(Z|s)}[\log r(Z)]$

features. Note that for SNI with $\lambda = 1$, i.e. with only the term $\mathcal{G}_{\text{AC}}(\bar{\pi}^r_\theta, \bar{\pi}_\theta, \bar{V}_\theta)$, this effectively recovers a L2 penalty on the activations since the variance of Z will then always be ignored and the KL-divergence between two Gaussians minimizes the squared difference of their means.

5.5 Experiments

In the following, we present a series of experiments to show that the IB finds more general features in the low-data regime and that this translates to improved generalisation in RL for IBAC agents, especially when combined with SNI. We evaluate our proposed regularisation techniques on two environments, one grid-world with challenging generalisation requirements [133] in which most previous approaches are unable to find the solution and on the recently proposed *Coinrun* benchmark [145]. We show that IBAC-SNI outperforms previous state of the art on both environments by a large margin. Details about the used hyperparameters and network architectures can be found in Appendix C, code to reproduce the results can be found at `https://github.com/microsoft/IBAC-SNI/`.

5.5.1 Learning Features in the Low-Data Regime

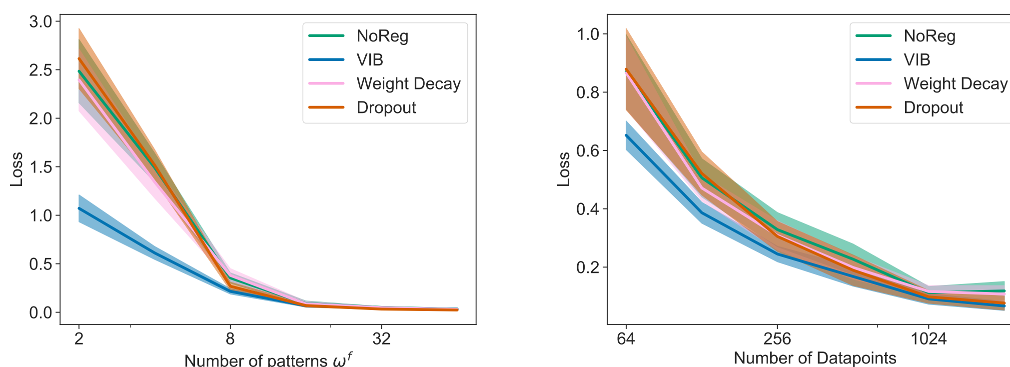


Figure 5.1: We show the loss on the test-data (lower is better). *Left:* Higher ω^f result in a larger difference in generality between features f^c and g^c, making it easier to fit to the more general g^c. *Right:* Learning g^c with fewer datapoints is more challenging, but needed early in training RL agents.

First we start in the supervised setting and show on a synthetic dataset that the VIB is particularly strong at finding more general features in the low-data regime and in the presence of multiple signals with varying degrees of generality. Our motivation is that the low-data regime is commonly encountered in RL early on in training and many environments allow the agent to base its decision on a variety of features in the state, of which we would like to find the most general ones.

We generate the training dataset $\mathcal{D}_{\text{train}} = \{(c_i, x_i)\}_{i=1}^N$ with observations $x_i \in \mathbb{R}^{d_x}$ and classes $c_i \in \{1, \dots, n_c\}$. Each data point i is generated by first drawing the class $c_i \sim Cat(n_c)$ from a uniform categorical distribution and generating the vector x_i by embedding the information about c_i in *two* different ways g^c and f^c (see Appendix C.2 for details). Importantly, only g^c is shared between the training and test set. This allows us to measure the model's relative reliance on g^c and f^c by measuring the test performance (all models perfectly fit the training data). We allow f^c to encode the information about c_i in ω^f different ways. Consequently, the higher ω^f, the less general f^c is.

In Fig. 5.1 we measure how the test performance of fully trained classification models varies for different regularisation techniques when we i) vary the generality of f^c and ii) vary the number of data-points in the training set. We find that most techniques perform comparably with the exception of the VIB which is able to find more general features both in the low-data regime and in the presence of multiple features with only small differences in generality. In the next section, we show that this translates to faster training and performance gains in RL for our proposed algorithm IBAC.

5.5.2 Multiroom

In this section, we show how IBAC can help learning in RL tasks which require generalisation. For this task, we do not distinguish between training and testing, but for each episode, we randomly generate a new layout. As the number of possible layouts is very large, learning can only be successful if the agent learns general features that are transferable between episodes.

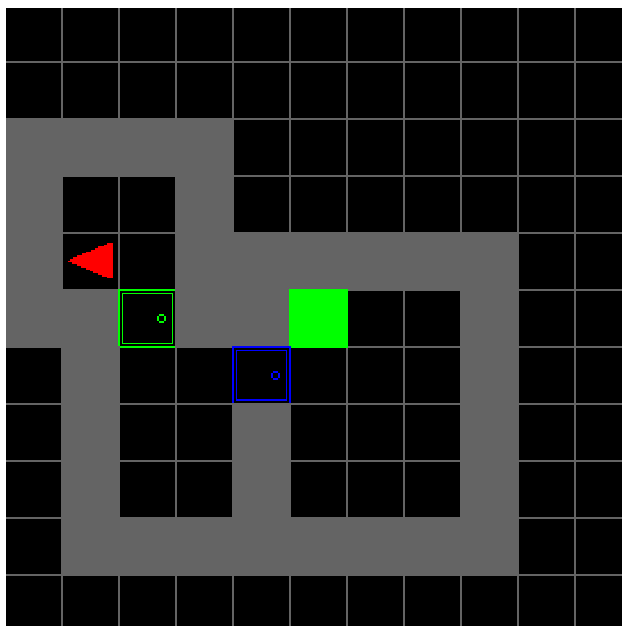


Figure 5.2: Typical layout of the Multiroom environment. The red triangle denotes the agent and its direction, the green full square is the goal, colored boxes are doors and grey squares are walls.

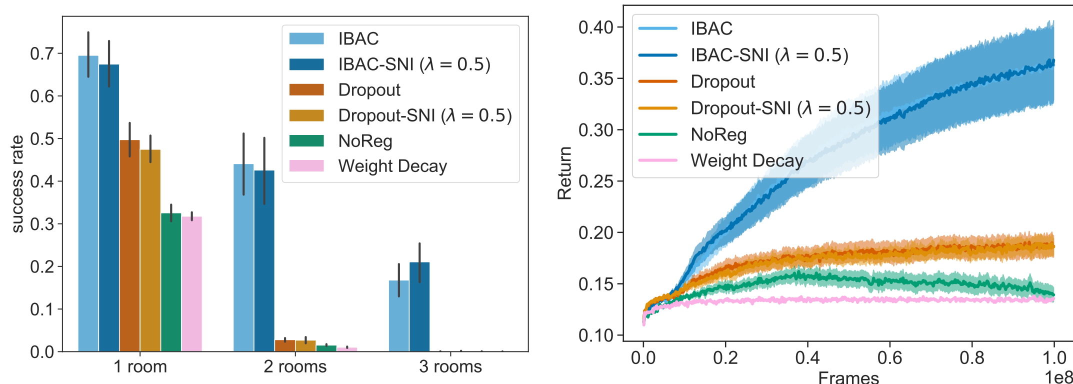


Figure 5.3: *Left:* Probability of finding the goal depending on level size for models trained on all levels. Shown are mean and standard error across 30 different seeds. *Right:* Mean and standard error over of the return of the same models averaged across all room sizes.

This experiment is based on [133]. The aim of the agent is to traverse a sequence of rooms to reach the goal (green square in Fig. 5.3) as quickly as possible. It takes discrete actions to rotate 90° in either direction, move forward and toggle doors to be open or closed. The observation received by the agent includes the full grid, one pixel per square, with object type and object status (like direction) encoded in the 3 color channels. Crucially, for each episode, the layout is generated randomly by placing a

random number of rooms $n_r \in \{1, 2, 3\}$ in a sequence connected by one door each.

The results in Fig. 5.3 report the online performance of the rollout policy, averaged across multiple parallel environments and episodes. They show that IBAC agents are much better at successfully learning to solve this task, especially for layouts with more rooms. While all other fully trained agents can solve less than 3% of the layouts with two rooms and none of the ones with three, IBAC-SNI still succeeds in an impressive 43% and 21% of those layouts. The difficulty of this seemingly simple task arises from its generalisation requirements: Since the layout is randomly generated in each episode, each state is observed very rarely, especially for multi-room layouts, requiring generalisation to allow learning. While in the 1 room layout the reduced policy stochasticity of the SNI agent slightly reduces performance, it improves performance for more complex layouts in which higher noise becomes detrimental. In the next section we will see that this also holds for the much more complex *Coinrun* environment in which SNI significantly improves the IBAC performance: While sub-optimal actions of the rollout policy in Multiroom only have limited adverse effect, they can prematurely terminate episodes in *Coinrun* and thereby severely degrade data quality.

5.5.3 Coinrun

On the previous environment, we were able to show that IBAC and SNI help agents to find more general features and to do so faster. Next, we show that this can lead to a higher final performance on previously unseen test environments. We evaluate our proposed regularisation techniques on *Coinrun* [145], a recently proposed generalisation benchmark with high-dimensional observations and a large variety in levels. Several regularisation techniques were previously evaluated there, making it an ideal evaluation environment for IBAC and SNI. We follow the setting proposed in [145], using the same 500 levels for training and evaluate on randomly drawn, new levels of only the highest difficulty.

As [145] have shown, combining multiple regularisation techniques can improve performance, with their best- performing agent utilizing data augmentation, weight

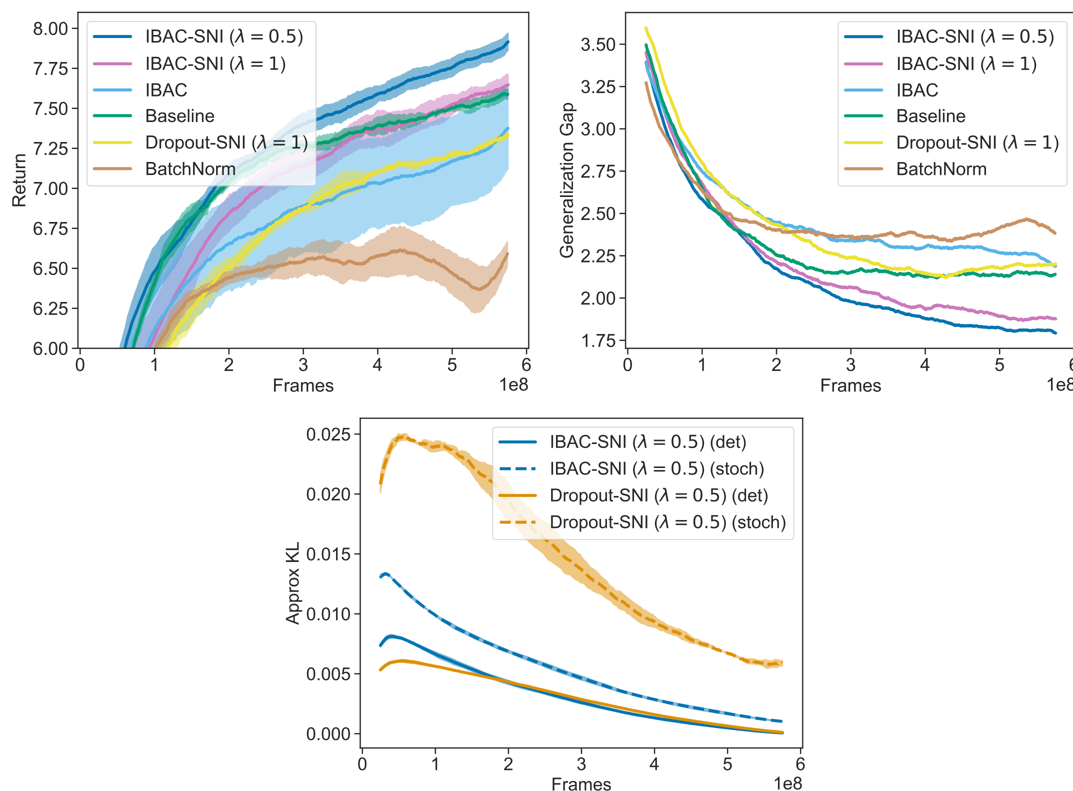


Figure 5.4: *Left:* Performance of various agents on the test environments. We note that 'BatchNorm' corresponds to the best performing agent in [145]. Furthermore, 'Dropout-SNI ($\lambda = 1$)' is similar to the Dropout implementation used in [145] but was previously not evaluated with weight decay and data augmentation. *Middle:* Difference between test performance and train performance (see Fig. C.4). Without standard deviation for readability. *Right:* Averaged approximate KL-Divergence between rollout policy and updated policy, used as proxy for the variance of the importance weight. Mean and standard deviation are across three random seeds.

decay and batch normalisation. As our goal is to push the state of the art on this environment and to accurately compare against their results, Fig. 5.4 uses weight decay and data-augmentation on all experiments. Consequently, 'Baseline' in Fig. 5.4 refers to *only* using weight decay and data-augmentation whereas the other experiments use Dropout, Batch Normalization or IBAC *in addition* to weight decay and data-augmentation. Results without those baseline techniques can be found in Appendix C.4.

First, we find that almost all previously proposed regularisation techniques *decrease* performance compared to the baseline, see Fig. 5.4 (left), with batch

normalisation performing worst, possibly due to its unusual interaction with weight decay [158]. Note that this combination with batch normalisation was the highest performing agent in [145]. We conjecture that regularisation techniques relying on stochasticity can introduce additional instability into the training update, possibly deteriorating performance, especially if their regularising effect is not sufficiently different from what weight decay and data-augmentation already achieve. This result applies to both batch normalisation and Dropout, with and without SNI, although SNI mitigates the adverse effects. Consequently, we can already improve on the state of the art by only relying on those two non-stochastic techniques. Furthermore, we find that IBAC in combination with SNI is able to significantly outperform our new state of the art baseline. We also find that for IBAC, $\lambda = 0.5$ achieves better performance than $\lambda = 1$, justifying using both terms in Eq. (5.4).

As a proxy for the variance of the off-policy correction term ${}^{\pi_\theta^r}/_{\pi_\theta}$, we show in Fig. 5.4 (right) the estimated, averaged KL-divergence between the rollout policy and the update policy for both terms, $\mathcal{G}_{\text{AC}}(\bar{\pi}_\theta^r, \bar{\pi}_\theta, \bar{V}_\theta)$, denoted by '(det)' and $\mathcal{G}_{\text{AC}}(\bar{\pi}_\theta^r, \pi_\theta, \bar{V}_\theta)$, denoted by '(stoch)'. Because PPO uses data-points multiple times it is non-zero even for the deterministic term. First, we can see that using the deterministic version reduces the KL-Divergence, explaining the positive influence of $\mathcal{G}_{\text{AC}}(\bar{\pi}_\theta^r, \bar{\pi}_\theta, \bar{V}_\theta)$. Second, we see that the KL-Divergence of the stochastic part is much higher for Dropout than for IBAC, offering an explanation of why for Dropout relying on purely the deterministic part ($\lambda = 1$) outperforms an equal mixing $\lambda = 0.5$ (see Fig. C.3).

5.6 Related Work

Generalisation in RL can take a variety of forms, each necessitating different types of regularisation. To position this work, we distinguished two types that, whilst not mutually exclusive, we believe to be conceptually distinct and found useful to isolate when studying approaches to improve generalisation.

The first type, *robustness to uncertainty* refers to settings in which the unobserved MDP m influences the transition dynamics or reward structure. Consequently the

current state s might not contain enough information to act optimally in the current MDP and we need to find the action which is optimal under the uncertainty about m. This setting often arises in robotics and control where exact physical characteristics are unknown and domain shifts can occur [159]. Consequently, *domain randomisation*, the injection of randomness into the environment, is often purposefully applied during training to allow for sim-to-real transfer [160, 161]. Noise can be injected into the states of the environment [162] or the parameters of the transition distribution like friction coefficients or mass values [163–165]. The noise injected into the dynamics can also be manipulated adversarially [166–168]. As the goal is to prevent overfitting to specific MDPs, it also has been found that using smaller [169] or simpler [143] networks can help. We can also aim to learn an adaptive policy by treating the environment as POMDP [137, 165] (similar to viewing the learning problem in the framework of Bayesian RL [170]) or as a meta-learning problem [100, 171–175].

On the other hand, we distinguish *feature robustness*, which applies to environments with high-dimensional observations (like images) in which generalisation to previously unseen states can be improved by learning to extract better features, as is the focus of this chapter. Recently, a range of benchmarks, typically utilizing procedurally generated levels, have been proposed to evaluate this type of generalisation [145–149, 176–179].

Improving generalisation in those settings can rely on generating more diverse observation data [145, 161, 180], or strong, often relational, inductive biases applied to the architecture [181–183]. Contrary to the results in continuous control domains, here deeper networks have been found to be more successful [145, 184]. Furthermore, this setting is more similar to that of supervised learning, so established regularisation techniques like weight decay, dropout or batch-normalisation have also successfully been applied, especially in settings with a limited number of training environments [145]. This is the work most closely related to ours. We build on those results and improve upon them by taking into account the specific ways

in which RL is *different* from the supervised setting. They also do not consider the VIB as a regularisation technique.

Combining RL and VIB has been recently explored for learning goal-conditioned policies [185] and meta-RL [125]. Both of these previous works [125, 185] also differ from the IBAC architecture we propose by conditioning action selection on both the encoded and raw state observation. These studies complement the contribution made here by providing evidence that the VIB can be used with a wider range of RL algorithms including demonstrated benefits when used with Soft Actor-Critic for continuous control in MuJoCo [125] and on-policy A2C in MiniGrid and MiniPacMan [185].

5.7 Conclusion

In this chapter we highlighted two important differences between supervised learning and RL: First, the training data is generated using the learned model. Consequently, using stochastic regularisation methods can induce adverse effects and reduce the quality of the data. We conjecture that this explains the observed lower performance of Batch Normalization and Dropout. Second, in RL, we often encounter a noisy, low-data regime early on in training, complicating the extraction of general features.

We argued that these differences should inform the choice of regularisation techniques used in RL. To mitigate the adverse effects of stochastic regularisation, we proposed Selective Noise Injection (SNI) which only selectively injects noise into the model, preventing reduced data quality and higher gradient variance through a noisy critic. On the other hand, to learn more compressed and general features in the noisy low-data regime, we proposed Information Bottleneck Actor Critic (IBAC), which utilizes an variational information bottleneck as part of the agent.

We experimentally demonstrated that the VIB is able to extract better features in the low-data regime and that this translates to better generalisation of IBAC in RL. Furthermore, on complex environments, SNI is key to good performance, allowing the combined algorithm, IBAC-SNI, to achieve state of the art on challenging generalisation benchmarks. We believe the results presented here can inform

a range of future works, both to improve existing algorithms and to find new regularisation techniques adapted to RL.

In the next chapter, we present one such algorithm, ITER, which is motivated by a closer inspection of the transient non-stationarity and the low-data regime early on in RL training.

6 The Impact of Non-stationarity on Generalisation in Deep Reinforcement Learning

Contents

6.1 Introduction **90**
6.2 Background **92**
6.3 The Impact of Non-stationarity on Generalisation . . **93**
6.4 Iterated Relearning **95**
6.4.1 Distillation Loss 97
6.4.2 Combining Training and Distillation 98
6.5 Experiments **99**
6.5.1 Experimental Results on Multiroom 100
6.5.2 Experimental Results on Sokoban 101
6.5.3 Experimental Results on ProcGen 101
6.5.4 Supervised Learning on CIFAR-10 103
6.6 Related Work **106**
6.7 Conclusion **107**

6.1 Introduction

In the previous chapter we discussed how widely used regularisation methods from Supervised Learning (SL) can be applied to RL to improve generalisation. We have also already introduced the idea that only observing few states early on in training

might lead to overfitting. In this chapter we will expand on this idea and investigate more closely the *transient non-stationarity* that is introduced by continuously updating the policy and over time exploring more and more of the environment as the agent improves. In deep RL, this non-stationarity is often not addressed explicitly. Typically, a single neural network model is initialised and continually updated during training. Conventional wisdom about catastrophic forgetting [186] implies that old updates from a different data-distribution will simply be forgotten. However, we provide evidence for an alternative hypothesis: networks exhibit a memory effect in their learned representations which can harm generalisation permanently if the data-distribution changed over the course of training.

To build intuition, we first study this phenomenon in a supervised setting on the CIFAR-10 dataset. We artificially introduce transient non-stationarity into the training data and investigate how this affects the asymptotic performance under the final, stationary data in the later epochs of training. Interestingly, we find that while asymptotic training performance is nearly unaffected, test performance degrades considerably, even after the data-distribution has converged. In other words, we find that latent representations in deep networks learned under certain types of non-stationary data can be inadequate for good generalisation and might not be improved by later training on stationary data.

Such transient non-stationarity is typical in RL. Consequently, we argue that this observed degradation of generalisation might contribute to the inferior generalisation properties recently attributed to many RL agents evaluated on held out test environments [141–143]. Furthermore, in contrast to Supervised Learning (SL), simply re-training the agent from scratch once the data-distribution has changed is infeasible in RL as current state of the art algorithms require data close to the on-policy distribution, even for off-policy algorithms like Q-learning [187].

To improve generalisation of RL agents despite this restriction, we introduce Iterated Relearning (ITER) in this chapter. In this paradigm for deep RL training, the agent's policy and value are periodically distilled into a freshly initialised student, which subsequently replaces the teacher for further optimisation. While

this occasional distillation step simply aims to re-learn and replace the current policy and value outputs for the training data, it allows the student to learn a better latent representation with improved performance for unseen inputs because it eliminates non-stationarity during distillation. We propose a practical implementation of ITER which performs the distillation in parallel to the training process without requiring additional training data. While this introduces a small amount of non-stationarity into the distillation step, it greatly improves sample efficiency without noticeably impacting performance.

Experimentally, we evaluate ITER on the *Multiroom* and *Sokoban* environments, as well as several environments from the recently proposed *ProcGen* benchmark and find that it improves generalisation. This provides further support to our hypothesis and indicates that the non-stationarity inherent to many RL algorithms, even when training on stationary environments, should not be ignored when aiming to learn robust agents. Lastly, to further support this claim and provide more insight into possible causes of the discovered effect, we perform additional ablation studies on the CIFAR-10 dataset.

6.2 Background

In addition to the MDP setup described in Section 2.2, we define the unnormalised discounted state distribution induced by a policy π as

$$d_\pi(s) = \sum_{t=0}^{\infty} \gamma^t \Pr\left(S_t = s | S_0 \sim p_0, A_t \sim \pi(\cdot|S_t), S_{t+1} \sim P(S_t, A_t)\right). \tag{6.1}$$

In ITER, we learn a sequence of policies and value functions, which we denote with $\pi^{(k)}(a|s)$ and $V^{(k)}(s)$ at the kth iteration ($k \in \{0, 1, 2, \dots\}$), parameterized by θ_k.

We briefly discuss some forms of non-stationarity which can arise in RL, even when the environment is stationary. For simplicity, we focus the exposition on actor-critic methods which use samples from interaction with the environment to estimate the policy gradient given by $g = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) A^\pi(s, a, s') | s, a, s' \sim d_\pi(s)\pi(a|s)P(s'|s, a)]$. As discussed, we also use neural networks to approximate the

baseline $V^\pi_\phi(s)$ and for bootstrapping from the future value $V^\pi_\phi(s')$ when estimating the advantage $A^\pi(s,a,s') = r(s,a) + \gamma V^\pi(s') - V^\pi(s)$.

There are at least three main types of non-stationarity in deep RL. First, we update the policy π_θ, which leads to changes in the state distribution $d_{\pi_\theta}(s)$. Early on in training, a random policy π_θ only explores states close to initial states s_0. As π_θ improves, new states further from s_0 are encountered. Second, changes to the policy also change the true value function $V^\pi(s)$ which $V^\pi_\phi(s)$ is approximating. Lastly, due to the use of bootstrap targets in temporal difference learning, the learned value $V^\pi_\phi(s)$ is not regressed directly towards $V^\pi(s)$. Instead V^π_ϕ fits a gradually evolving target sequence even under a fixed policy π, thereby also changing the policy gradient estimator g.

6.3 The Impact of Non-stationarity on Generalisation

In this section we investigate how asymptotic performance is affected by changes to the data-distribution during training. In particular, we assume an initial, transient phase of non-stationarity, followed by an extended phase of training on a stationary data-distribution. This is similar to the situation in RL where the data-distribution is affected by a policy which converges over time. We show that this transient non-stationarity has a permanent effect on the learned representation and negatively impacts generalisation. As interventions in RL training can lead to confounding factors due to off-policy data or changed exploration behaviour, we utilise Supervised Learning (SL) here to provide initial evidence in a more controlled setup. We use the CIFAR-10 dataset for image classification [188] and artificially inject non-stationarity.

Our goal is to provide qualitative results on the impact of non-stationarity, not to obtain optimal performance. We use a ResNet18 [189] architecture, similar to those used by [190] and [132]. Parameters are updated using SGD with momentum and, following standard practice in RL, we use a constant learning rate and do not

use batch normalisation. Weight decay is used for regularisation. Hyper-parameters and more details can be found in Appendix D.2.

We train for a total of 2500 epochs. While the last 1500 epochs are trained on the full, unaltered dataset, we modify the training data in three different ways during the first 1000 epochs. Test data is left unmodified throughout training. While each modification is loosely motivated by the RL setting, our goal is not to mimic it exactly (which would be infeasible), nor to disentangle the contributions of different types of non-stationarity. Instead, we aim to show that these effects reliably occur in the presence to various types of non-stationarity, and provide intuitions that can be brought into the RL setting in Section 6.4.

For the first modification, called `Dataset Size`, we initially train only on a small fraction of the full dataset and gradually add more data points after each epoch, at a constant rate, until the full dataset is available after epoch 1000. During the non-stationary phase, data points are reused multiple times to ensure the same number of network updates are made in each epoch. For the modification `Wrong Labels` we replace all training labels by randomly drawn incorrect ones. After each epoch, a constant number of these labels are replaced by their correct values. Lastly, `Noisy Labels` is similar to `Wrong Labels`, but the incorrect labels are sampled uniformly at the start of each epoch. For both, all training labels are correct after epoch 1000. While `Dataset Size` is inspired by the changing state distribution seen by an evolving policy, `Wrong Labels` and `Noisy Labels` are motivated by the consistent bias or fluctuating errors a learned critic can introduce in the policy gradient estimate.

The results are shown in Fig. 6.1. While the final training accuracy (left) is almost unaffected (see Table D.1 in the appendix for exact results), all three non-stationary modifications significantly reduce the test accuracy (right). The plateau in accuracy shows that this effect persists even after the models are further trained using the full dataset with correct labels: non-stationarity early in training has a permanent effect on the learned representations and quality of generalisation. These results indicate that the non-stationarity introduced by the gradual convergence

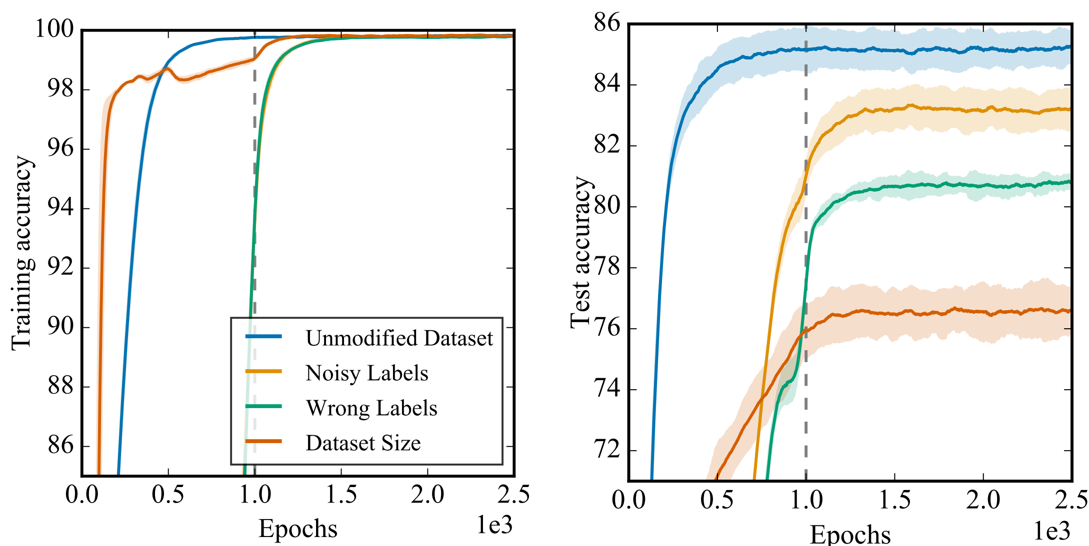


Figure 6.1: Accuracy on CIFAR-10 when the training data is non-stationary over the first 1000 epochs (dashed line). The remaining epochs are trained on the full, unaltered training data. Testing is performed on unaltered data throughout. While final training performance (left) is almost unaffected, test accuracy (right) is significantly reduced by initial, transient non-stationarity.

of the policy in RL might similarly deteriorate the generalisation of the agent. To overcome this, we propose ITER in the next section. The key insight enabling ITER is that the observed negative effect is restricted to the test data, whereas the predictions on the training data are unaffected and of high quality.

6.4 Iterated Relearning

In Section 6.3, we have seen evidence that the non-stationarity which is present in many deep RL algorithms might lead to impaired generalisation on held-out test environments. To mitigate this and improve generalisation to previously unseen states, we propose Iterated Relearning (ITER): instead of updating a single agent model throughout the entire training process, ITER learns a sequence of models, each of which is exposed to less non-stationarity during its training. As we will show in Section 6.5, this improves generalisation. ITER can be applied on top of a wide range of base RL training methods. For simplicity, we focus in the following exposition on actor-critic methods and use PPO [39] for the experimental evaluation.

The underlying insight behind ITER is that at any point during RL training the latent representation of our current agent network might be significantly damaged by non-stationarity, but its outputs on the training data are comparatively unaffected (see Fig. 6.1). Consequently, ITER aims to periodically replace the current agent network, the 'teacher', by a 'student' network which was freshly initialised and trained to mimic the teacher on the current training data. Because this re-learning and replacement step can be performed on stationary data, it allows us to re-learn a policy that matches performance on the training data but generalises better to novel test environments.

ITER begins with an initial policy $\pi^{(0)}$ and value function $V^{(0)}$ and then repeats the following steps, starting with iteration $k = 0$.

1. Use the base RL algorithm to train $\pi^{(k)}$ and $V^{(k)}$.
2. Initialise new *student* networks for $\pi^{(k+1)}$ and $V^{(k+1)}$. We refer to the current policy $\pi^{(k)}$ and value function $V^{(k)}$ as the *teacher*.
3. Distill the teacher into the student. This phase is discussed in more detail in Section 6.4.1.
4. The student replaces the teacher: $\pi^{(k)}$ and $V^{(k)}$ can be discarded.
5. Increment k and return to step 1. Repeat as many times as needed.

This results in alternating RL training with distillation into a freshly initialised student. The RL training phases continue to introduce non-stationarity until the models converge, so we want to iterate the process, repeating steps 1-4. How often we do so depends on the environment and can be chosen as a hyper-parameter. In practise we found the results to be quite robust to this choice and recommend, as general rule, to iterate as often as possible within the limits outlined in Section 6.4.2. There, we introduce a practical implementation of ITER which re-uses data between steps 1 and 3 in order to not require additional samples from the environment.

6.4.1 Distillation Loss

Our goal during the distillation phase (step 3) is to learn a new student policy $\pi^{(k+1)}$ and value function $V^{(k+1)}$ that imitate the current teacher $(\pi^{(k)}, V^{(k)})$. If the student and teacher share the same network architecture, the student could of course imitate the teacher exactly by copying its parameters. However, since the teacher was trained under non-stationarity, its generalisation performance is likely degraded (see Section 6.3). Consequently, we want to instead train a freshly initialised student network to mimic the teacher's outputs for the available data, but learn a better internal representation by training on a stationary data distribution collected by the teacher $\pi^{(k)}$, i.e., $s, a \sim d_{\pi^{(k)}}(s)\pi^{(k)}(a|s)$.

The student, parameterised by θ_{k+1}, is trained using a linear combination of four loss terms:

$$\mathcal{L}(\theta_{k+1}) = \alpha_\pi \mathcal{L}_\pi + \alpha_{\text{V}} \mathcal{L}_{\text{V}} + \mathcal{L}_{\text{PG}} + \lambda_{\text{TD}} \mathcal{L}_{\text{TD}} \tag{6.2}$$

where λ_{TD} is a fixed hyper-parameter and we linearly anneal α_π and α_{V} from some fixed initial value to zero over the course of each distillation phase.

$\mathcal{L}_\pi$ and $\mathcal{L}_V$ are supervised losses minimising the disagreement between outputs of the student and the teacher:

$$\begin{aligned} \mathcal{L}_\pi(\theta_{k+1}) &= \mathbb{E}_{s \sim d_{\pi^{(k)}}} \left[D_{\text{KL}} \left[\pi^{(k)}(\cdot|s) \,\middle\|\, \pi^{(k+1)}(\cdot|s) \right] \right], \\ \mathcal{L}_{\text{V}}(\theta_{k+1}) &= \mathbb{E}_{s \sim d_{\pi^{(k)}}} \left[\left(V^{(k)}(s) - V^{(k+1)}(s) \right)^2 \right]. \end{aligned} \tag{6.3}$$

The additional terms $\mathcal{L}_{\text{PG}}$ and $\mathcal{L}_{\text{TD}}$ are off-policy RL objectives for updating the actor and critic:

$$\begin{aligned} \mathcal{L}_{\text{PG}}(\theta_{k+1}) &= -\mathbb{E}_{s \sim d_{\pi^{(k)}}, a \sim \pi^{(k)}, s' \sim T(s,a)} \left[\log \pi^{(k+1)}(a|s) \perp \left(\tfrac{\pi^{(k+1)}(a|s)}{\pi^{(k)}(a|s)} A^{(k+1)}(s, a, s') \right) \right], \\ \mathcal{L}_{\text{TD}}(\theta_{k+1}) &= \mathbb{E}_{s \sim d_{\pi^{(k)}}, a \sim \pi^{(k)}, s' \sim T(s,a)} \left[\left(A^{(k+1)}(s, a, s') \right)^2 \perp \tfrac{\pi^{(k+1)}(a|s)}{\pi^{(k)}(a|s)} \right], \end{aligned} \tag{6.4}$$

where $A^{(k+1)}(s, a, s') = r(s, a) + \gamma \perp V^{(k+1)}(s') - V^{(k+1)}(s)$ denotes the estimated advantage of choosing action a and $\perp$ is a `stop-gradient` operator, its operand is treated as a constant when taking derivatives of the objective. In practice, the losses in Eq. (6.3) remain nonzero during distillation, potentially causing a drop

in performance once the student replaces the teacher. The off-policy RL losses in Eq. (6.4) allow the student to already take performance on the RL task into account, reducing or eliminating this performance drop. We use PPO losses to optimise Eq. (6.4) in our experiments. An important assumption made in this approach is that the data provided by the teacher is sufficiently diverse for training a student during the distillation phase, such that the student is robust to the covariate shift introduced when replacing the teacher. In practise we found this not to be a problem and while small initial drops in performance were observed on some environments, the student always recovered very quickly to the same or better performance than the teacher.

6.4.2 Combining Training and Distillation

To fully eliminate non-stationarity during the distillation step we need to collect additional data from the environment using a fixed teacher policy. However, this would slow down training by increasing the total number of samples required. Here, to improve sample efficiency, we propose two practical implementations of ITER which reuse data between teacher and student:

Sequential training: Store the last N transitions that were used to update the teacher in a dataset $\mathcal{D}$. During the distillation phase, draw batches from $\mathcal{D}$ instead of collecting new data from the environment. While this does not introduce non-stationarity, it leads to evaluating the teacher on old, off-policy data, for which the quality of its outputs may be degraded. Furthermore, some of the data might be obsolete under the current state-distribution and we require additional memory to store $\mathcal{D}$.

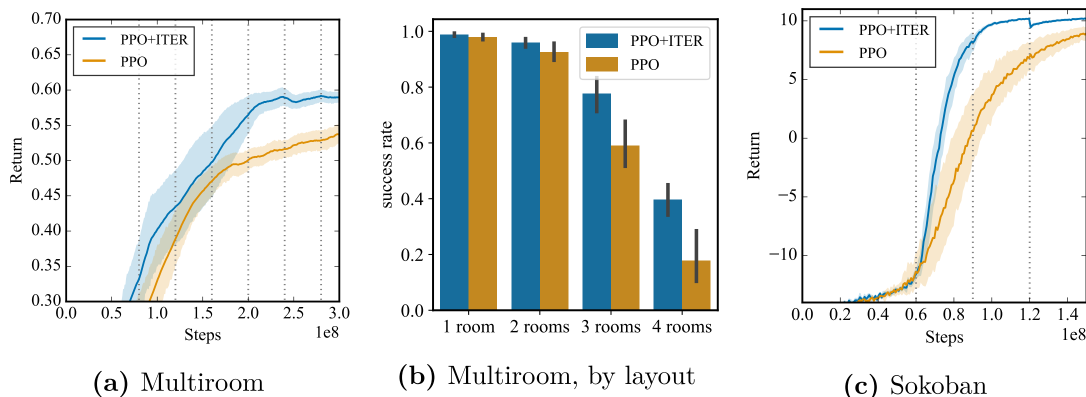
Parallel training: Whenever the teacher is updated on a batch of data $\mathcal{B}$, also update the the student on the same batch. This approach introduces some non-stationarity as distillation is performed over multiple batches $\mathcal{B}$ while the teacher is changing. However, the teacher evolves much less over the course of the distillation phase than does a policy trained from scratch to achieve the same performance. In practise we found this to be a worthy trade-off. Advantages of

this method are that no additional memory $\mathcal{D}$ is required, the teacher is only evaluated on data on which it is currently trained and updates to the student and teacher can be performed in parallel.

Both approaches perform similarly in our experimental validation. We use parallel training for the main experiments due to the smaller memory requirements, the ability to efficiently perform the student distillation in parallel and because tuning the hyper-parameter was significantly easier: While tuning the size of $\mathcal{D}$ for sequential ITER involves trading off overfitting (for small $\mathcal{D}$) against off-policy data (for large $\mathcal{D}$), in parallel ITER the results were robust to the choice of how many batches $\mathcal{B}$ were used in the distillation phase as long as some minimum number was surpassed. Consequently, hyper-parameter tuning for parallel ITER only involved increasing the length of the distillation phase until no drop in student performance was observed when replacing the teacher. Similarly, we found it useful to repeat the distillation phase as often as possible, while leaving enough training steps at the end of training to make sure that the last policy reached its (approximate) limiting performance. We set $\alpha_\pi = 1$ and $\alpha_V = 0.5$ as initial values without further tuning as preliminary experiments showed no impact within reasonable ranges.

6.5 Experiments

In the following, we evaluate ITER on *Multiroom* [133] and on several environments from the *ProcGen* [132] benchmark which was specifically designed to measure generalisation by introducing separate test- and training levels. We also provide ablation studies showing that parallel and sequential training perform comparably, and that the loss terms Eq. (6.4) in Eq. (6.2) are beneficial. We find that ITER improves generalisation, which also supports our hypothesis about the negative impact of transient non-stationarity in RL. Lastly, we re-visit the SL setting from Section 6.3 and perform additional experiments leading us to the formulation of the *legacy feature* hypothesis to explain the observed effects.



(a) Multiroom **(b)** Multiroom, by layout **(c)** Sokoban

Figure 6.2: Evaluation on *Multiroom* and *Sokoban*. Shown are mean and standard error over twelve training seeds. *Left:* Return for PPO with and without ITER on *Multiroom*. Dotted lines indicate when the network was replaced by a new student. *Middle:* Evaluation on layouts with a fixed number of rooms; training is still with a random number of rooms. ITER's advantage is more pronounced for harder levels. *Right:* Return on Sokoban.

6.5.1 Experimental Results on Multiroom

First, we evaluate ITER on the *Multiroom* environment. The agent's task is to traverse a sequence of rooms to reach the goal (see Fig. D.1 for example layout) as quickly as possible. It can take discrete actions to rotate 90° in either direction, move forward, and open or close the doors between rooms. The observation contains the full grid, one pixel per square. Object type, including empty space and walls, as well as any object status, like direction, are encoded in the three 'colour' channels. For each episode, a new layout is generated by randomly placing between one and four connected rooms on the grid. The agent is always initialised in the room furthest from the goal. This randomness favours agents that are better at generalising between layouts as memorisation is impossible due to the high number of feasible layouts. The results are shown in Fig. 6.2: Using ITER on top of PPO increases performance. The performance difference is more pronounced for layouts with more rooms, possibly because such layouts are harder and likely only solved later in training, at which point negative effects due to prior non-stationarity in training are more pronounced.

6.5.2 Experimental Results on Sokoban

We also evaluate ITER on *Sokoban* [191][1]. See Fig. D.1 for an example layout. Similarly to *Multiroom* the observation contains the full grid, one pixel per square. Object types are encoded by colour. Again, a new layout is generated at the beginning of each new episode, favouring agents that can generalise well between states. Actions allow to push, pull or move in all four cardinal directions, or do nothing. The goal of the agent is to position the two boxes, which can be pushed or pulled, on the two available targets. Walls prevent movement for both the agent and the boxes. Positive rewards are provided for positioning a box on a target ($r_b = 1$) and successfully solving each level ($r_l = 10$). A small negative reward per time-step ($r_s = -0.1$) encourages fast solutions. The results can be seen in Fig. 6.2: ITER learns much faster and has also a slightly higher asymptotic performance.

Note that both for *Multiroom* and *Sokoban* we train and test on the same (very large) set of possible layout configurations, so the main expected advantage of ITER is a more sample efficient training through better generalisation. In the next section, we will evaluate the agent on previously unseen environments, directly measuring its generalisation performance.

6.5.3 Experimental Results on ProcGen

Next, we evaluate ITER on several environments from the *ProcGen* benchmark. We follow the evaluation protocol proposed in [132]: for each environment, we train on 500 randomly generated level layouts and test on additional, previously unseen levels. Due to computational constraints, we consider a subset of six environments. We chose *StarPilot*, *Dodgeball*, *Climber*, *Ninja*, *Fruitbot* and *BigFish* based on the results presented in [132] as they showed baseline generalisation performance better than a random policy, but with a large enough generalisation gap. ITER improves performance for both PPO and IBAC with selective noise injection [28]. Results are presented in Fig. 6.3 and more individual plots, including performance on training levels, can be found in Appendix D. In Fig. 6.4 we show in ablations that both the

[1]We are using `PushAndPull-Sokoban-v2`

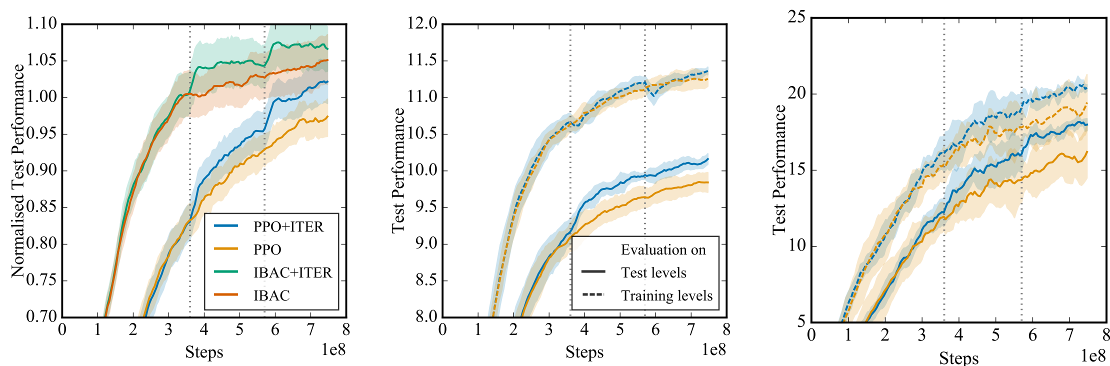


Figure 6.3: Evaluation on *ProcGen*. Dashed lines indicate replacing the teacher. *Left:* Test performance averaged over six environments (*StarPilot*, *Dodgeball*, *Climber*, *Ninja*, *Fruitbot* and *BigFish*). Shown are mean and standard error over all 30 runs (five per environment). Results are normalised by the final test-performance of the PPO baseline on each respective environment to make them comparable. We also compare against the previous state of the art method IBAC-SNI [28]. *Middle:* Evaluation on *Climber*. ITER improves test performance without improving training, supporting our claim that ITER improves the latent representation of the agent. *Right:* Evaluation on *BigFish*. On some environments, ITER improves both train- and test-performance.

parallel and sequential implementations of ITER perform comparably, while not using the off-policy RL terms $\mathcal{L}_{\text{PG}}$ and $\mathcal{L}_{\text{TD}}$ in Eq. (6.2) decreases performance.

Similarly to previous literature [28, 145], we found that weight decay improves performance and apply it to all algorithms evaluated on *ProcGen*. Our results show that the negative effects of non-stationarity cannot easily be avoided through standard network regularisation: we can improve test returns through ITER despite regularisation with weight decay and IBAC, both shown to be among the most effective regularisation methods on this benchmark [28].

In the previous two sections we have shown the effectiveness of ITER in improving generalisation of RL agents. Because the main mechanism of ITER is in reducing the non-stationarity in the training data which is used to train the agent, this result further supports that such transient non-stationarity is detrimental to generalisation in RL.

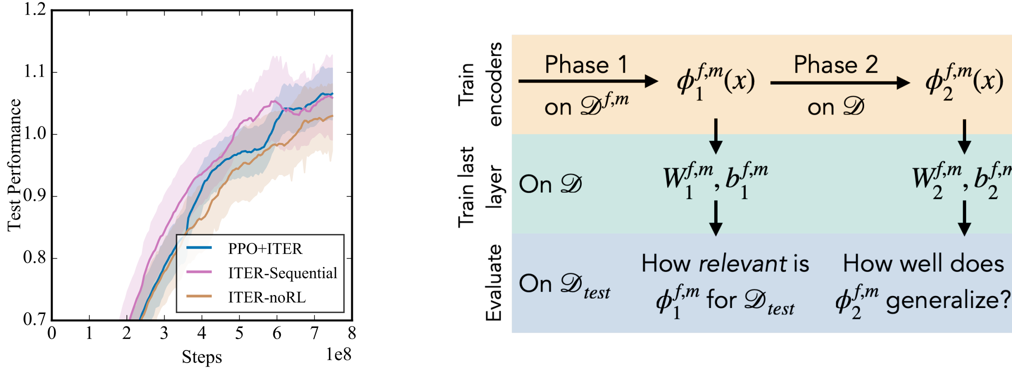


Figure 6.4: *Left:* Ablation studies with sequential ITER and ITER without terms $\mathcal{L}_{\text{PG}}$ and $\mathcal{L}_{\text{TD}}$ (Eq. (6.4)). *Right:* Schematic depiction of training setup for Fig. 6.5 (middle and right). More details are given in Section 6.5.4. $\mathcal{D}$ is the unmodified CIFAR-10 training data while for $\mathcal{D}^{f,m}$ modification $m \in \{\texttt{Noisy Labels}, \texttt{Wrong Labels}, \texttt{Dataset Size}\}$ is applied to the fraction $(1-f)$ of all data-points. In this two phase training setup, we first train on $\mathcal{D}^{f,m}$ during phase 1 and continue on $\mathcal{D}$ during phase 2. A linear predictor parameterised by $(W_i^{f,m}, b_i^{f,m})$ is trained on $\mathcal{D}$ after each phase i, while holding the encoder $\phi_i^{f,m}(x)$ fixed. Evaluation of the resulting classifiers is performed on the original test data. Classifier $i=1$ measures the relevance of the legacy features while classifier $i=2$ measure the final generalisation performance.

6.5.4 Supervised Learning on CIFAR-10

In this section, we aim to further understand the mechanism by which non-stationarity impacts generalisation by revisiting the easily controlled SL setting presented in Section 6.3.

First, we confirm that, like with ITER for RL, we can improve generalisation while only learning to mimic the outputs of a poorly generalising teacher for the training data. We train the teacher using the same setup as in Section 6.3. In a second step, we train a freshly initialised student for 1000 epochs to fit the argmax predictions of this teacher on the training data, i.e., the true labels in the training data are unused. Test accuracy is still measured using the true labels. The results of this distillation phase are shown in Fig. 6.5 (left). The student (solid lines) recovers the test accuracy achieved by stationary training, compared to the poor asymptotic teacher performance (dashed lines) from Fig. 6.1. This confirms that the teacher's outputs on the training data are suitable targets for distillation.

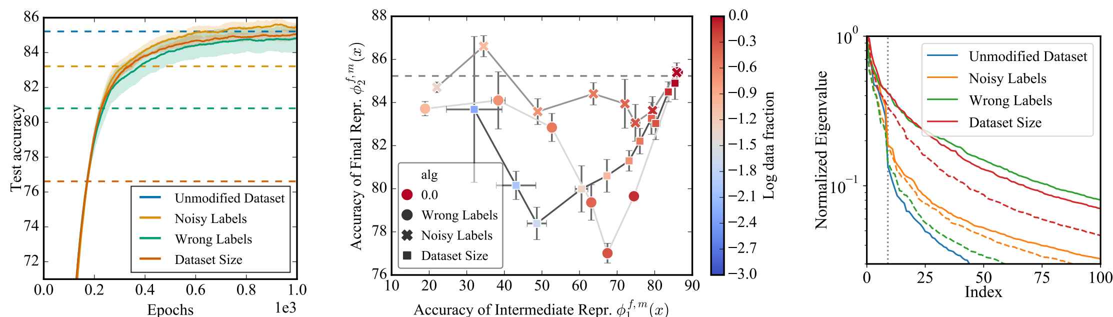


Figure 6.5: *Left:* Test accuracy of students (solid lines) that only learn to mimic the behaviour of poorly generalising teachers in Fig. 6.1 (dashed lines). *Middle:* Final test accuracy of networks trained consecutively on two different datasets. The x-axis shows the accuracy of using encoders trained on the first dataset, retraining only the last layer on the second: nearly useless earlier representations impact future learning much less than slightly sub-optimal ones. Markers indicate modifications to first dataset; colours indicate the fraction of unmodified data points f. Dashed line shows accuracy for $f = 1$. *Right:* Singular values of feature matrix Φ, normalised by the largest SV. Solid lines show intermediate values of f with low test accuracy, dashed lines small values of f with higher accuracy. More plots can be found in the appendix.

Second, we aim to better understand why non-stationarity affects the generalisation performance. To do so, we investigate the latent representation of the network. We view all but the last layer as the encoder, producing the latent *representation* $\phi(x) \in \mathbb{R}^p$ for input x, on which the classification is linear: $y = \text{softmax}(W\phi(x) + b)$ with $W \in \mathbb{R}^{|C| \times p}$ denoting a weight matrix and $b \in \mathbb{R}^{|C|}$ denoting a bias vector. By *features* we refer not to the representation $\phi(x)$ as a whole, but to aspects of the input to which the encoder learned to pay attention and which therefore can impact the latent representation [192]. More quantitatively, we can define the representation matrix $\Phi \in \mathbb{R}^{N \times p}$, consisting of the latent representations of all $N = 10000$ test data points. Performing singular value decomposition (SVD) on Φ yields mutually orthogonal directions (the right-singular vectors) in which the latent representations of the various inputs are different from one another. We can see each such direction as corresponding to one feature, with the corresponding singular value expressing its strength, i.e., how strongly it impacts the latent representation.

Our hypothesis is that under a non-stationary data distribution, the encoder is more likely to reuse previously learned features (as these are already available)

instead of learning new features from scratch. If these old (or 'legacy') features generalise worse on the new data, for example because they are overfit to a smaller or less diverse dataset, this in turn deteriorates generalisation permanently if they are not replaced. This leads to two predictions: First, the observed drop in generalisation should only occur if the previously learned features are relevant for the new task, but suboptimal. If they are irrelevant, they will not be reused. If they are optimal, they do not negatively impact generalisation. Second, we expect the final network to rely on more features in its latent representation if it is reusing suboptimal features: because these are not as general, more features are required to discriminate between all inputs.

To experimentally evaluate both predictions, we simplify the experimental setting to two phases (see Fig. 6.4 for a schematic depiction of the training setup). The first training phase uses a stationary, but modified, dataset $\mathcal{D}^{f,m}$, and the second phase uses the full, unmodified, training dataset $\mathcal{D}$. To generate $\mathcal{D}^{f,m}$, we use the same modifications as before, $m \in \{\texttt{Noisy Labels}, \texttt{Wrong Labels}, \texttt{Dataset Size}\}$, but instead of annealing the fraction of correct data points f from 0 to 1 as in Section 6.3, it is fixed at a certain value. Changing this value f allows us to tune the relevance of the features learned on $\mathcal{D}^{f,m}$ (also see Fig. D.1). In this setup the only non-stationarity is the change in data from phase 1 to phase 2. We first train the network for 700 epochs on $\mathcal{D}^{f,m}$, which yields an intermediate representation $\phi_1^{f,m}(x)$, followed by another 800 epochs on $\mathcal{D}$ yielding the final representation $\phi_2^{f,m}(x)$ (see Fig. D.2 for training curves).

To test our first hypothesis, we want to measure how relevant the representation $\phi_1^{f,m}(x)$ is for the final data distribution and how well $\phi_2^{f,m}(x)$ generalises. We train a linear predictor for each *fixed* representation on the *full* dataset $\mathcal{D}$.[2] The test accuracy of the classifier based on $\phi_1^{f,m}(x)$ measures how well we can perform on $\mathcal{D}$ with features learned on $\mathcal{D}^{f,m}$, i.e., their *relevance*. The test accuracy of the classifier based on $\phi_2^{f,m}$ measures how well the final network was able to recover from the initial bias and learn to generalise well despite non-stationarity.

[2]The linear predictor is $y = \sigma(W\phi_{f,m}(x) + b)$, where σ is the softmax function and x the input image.

In Fig. 6.5 (middle), we plot the accuracy of the intermediate predictor (i.e., the relevance) on the x-axis and of the final generalisation performance on the y-axis. Each point corresponds to one value of $f \in (0, 1]$, shown as colour, and one modification type m as indicated by the marker shape. By changing f from 0 to 1 (i.e., from blue to dark red) we can increase the relevance of the intermediate features from nearly irrelevant to optimal on $\mathcal{D}$. Interestingly, an almost useless intermediate representation (30% on the x-axis) does not impede the final performance much, while relevant but suboptimal intermediate features (around 60% on the x-axis) lead to a marked drop in performance. This supports our first hypothesis and also rules out that the observed effect is due to a decrease in network flexbility. The strong final performance for $f \to 0$ (i.e., for low relevance) also rules out decreased network flexibility, for example due to dead neurons (for ReLUs) or saturation (for tanh), as the main driver of reduced generalisation.

Our second prediction is that relevant, but suboptimal features in $\phi_1^{f,m}$ should lead to the usage of more features in $\phi_2^{f,m}$ compared to irrelevant or optimal features. To test this prediction, we plot in Figure 6.5 (right) the singular values (SVs) of the representation matrix $\Phi \in \mathbb{R}^{N \times p}$ as defined above. We plot the values for the smallest values of f (dashed lines, "irrelevant features") and for intermediate values of f (solid lines, 'relevant but sub-optimal features'). The blue line corresponds to optimal features. The tails of the singular values are heavier for intermediate values of f, indicating that the network is relying on more features in those cases, supporting our second prediction.

6.6 Related Work

Knowledge distillation [193] with identical teacher and student architectures has been shown to improve test accuracy [194], even in the absence of non-stationarities in the data. This improvement has been attributed to the ease of predicting the output distribution of the teacher compared to the original 'hard' labels [195, 196]. While we apply such 'soft' distillation for ITER on RL, we use 'hard' labels in our SL experiments.

Policy distillation has been applied to RL [197], for example for multi-task learning and compression [101, 198, 199], imitation learning [200], or faster training [201, 202]. Closer to ITER, Czarnecki et al. [203] use policy distillation to learn a sequence of agents. However, their Mix & Match algorithm solves tasks of growing complexity, for example, to grow the action space of the agent, not to tackle generalisation or non-stationarity.

While the topic of non-stationarity is central to the area of continual learning (see [204] for a recent review), the field is primarily concerned with preventing catastrophic forgetting [205] when the environment or task changes during training [206, 207]. For non-stationary environments during agent deployment, the approach is typically to detect such changes and respond accordingly [208–210]. By contrast, we assume a stationary environment and investigate the impact of transient non-stationarity, for example induced by an improving policy. We also show that intentionally forgetting the representation, but not the learned outputs, can improve generalisation in this case.

Neural networks are used in deep RL to allow generalisation across similar states [211]. Several possibilities have been proposed to further improve generalisation, including to provide more diverse training environments [161], inject noise into the environment [162, 212], incorporate inductive biases in the architecture [181], or regularise the network [28, 145, 213]. While regularisation reduces overfitting, we show in our experiments that this is insufficient to counter the negative effects of non-stationarity, and that ITER can be complementary to other types of regularisation.

6.7 Conclusion

In this chapter, we further expanded on the impact of transient non-stationarity on the generalisation performance of trained RL agents. First, in several SL experiments on the CIFAR-10 dataset, we confirmed that non-stationarity can considerably degrade test performance while leaving training performance nearly unchanged. To explain this effect, we proposed and experimentally supported the *legacy feature* hypothesis that networks exhibit a tendency to adopt, rather then forget, features

learned earlier during training if they are sufficiently relevant, though not necessarily optimal, for the new data. We also showed that self-distillation, even without using the true training labels, improves performance on the test-data.

As discussed, many deep RL algorithms induce similar transient non-stationarity, for example due to a gradually converging policy. Consequently, to improve generalisation in deep RL, we proposed ITER which reduces the non-stationarity the agent networks experience during training. Our experimental results on the *Multiroom* and *ProcGen* benchmarks empirically support the benefits of ITER, indicating that transient non-stationarity has a negative impact in deep RL.

7 Afterword

In this thesis we proposed a series of methods, all centered around the idea of generalisation and inductive biases, to improve the performance of deep reinforcement learning agents.

In Part I, we focussed on embedding inductive biases into the agent architecture. First, with DVRL (Chapter 3), this allowed us to improve asymptotic performance in complex, partially observable environments compared to previous state of the art results. However, the field is moving fast and many novel architectures for encoding sequential data are emerging (e.g. [41]). Nevertheless, we believe that particle filtering and better belief computation remain two powerful ideas when combined with neural networks. For example, the focus on belief states rather than attending to the entire past sequence of observations at each moment in time could act as regularisation and could thereby help with agent generalisation in POMDPs. We are excited about future work in this direction, hopefully facilitated by very recent developments in providing suitable benchmarking environments [e.g. 214]. Furthermore, the idea of variationally computed belief states has recently been picked up in the Meta-RL literature [e.g. 126, 215], opening up the possibility of further improvements through the use of particles.

As a second application of architectural inductive biases, we investigated hierarchical agents. By utilising a multitask setting, MSOL (Chapter 4) is able

to learn easily re-usable sub-policies to facilitate transfer. While these results are promising, and we hope impactful for the important field of hierarchical RL, they are also limited as we only focussed on transfer to novel *tasks* (i.e. different reward functions) but not to novel *states*.

To overcome this often encountered limitation, in Part II we turn our attention away from inductive biases and towards measuring and improving generalisation to novel states which were not previously seen during training. Generalisation in deep RL is an exciting direction, not least because many aspects about why neural networks generalise are still not well understood. In this thesis, instead of trying to answer this very broad question, we zoomed in on peculiarities of the RL setting, as compared to other training paradigms like SL, and investigated how these affect generalisation with neural networks. First, in Chapter 5, we found that stochastic regularisation methods like e.g. Dropout or VIB cause complications when used in conjunction with state of the art online-RL methods. Second, in Chapter 6, we turned our attention towards the transient non-stationarity induced by many RL algorithms and showed that it can degenerate the generalisation of trained agents. In both cases, we were able to significantly improve the performance by proposing solutions to overcome or alleviate these problems. We hope our proposed algorithms and insights help improving the robustness of RL agents.

However, as discussed in the beginning of this work, to greatly expand the scope of domains to which deep RL can successfully be applied, we also need to greatly reduce the number of environmental interactions required for training. While we do observe improved sample complexity through better generalisation on some environments (e.g. in Chapter 6), the magnitude of the observed effect won't be enough by itself.

Instead, one exciting future direction of work could be to apply improved generalisation of the type discussed in Part II to methods like MSOL or Meta-RL in order to better transfer knowledge not only between tasks, but also to novel states or environments. An alternative approach is offline-RL [see e.g. 216] which allows better re-use of once generated data. Currently, offline-RL still struggles to provide

competitive results, but we are excited about future work in this direction and hope that insights into better generalisation can help to make better use of limited data.

We are convinced insights into generalisation will significantly contribute towards making deep RL more robust, widely applicable and easier to use. But much work remains to be done. One the one hand, very little theoretical insights are available on this topic in deep RL. Furthermore, a better understanding is required on what makes generalisation in RL hard. For example, it is noteworthy that the number of levels required for near perfect generalisation in various domains in *ProcGen* [132] vary by orders of magnitude. Similarly, we also expect interesting insights from comparing different RL approaches (e.g. model-based vs. model-free methods) with regards to their generalisation capabilities on different domains.

The progress made by the deep RL research community over the recent years has been breathtaking. We hope our work contributed to it and will perhaps even inspire further advances and more ideas. There is huge potential for RL to transform many aspects of our life for the better and it is exciting to see progress being made on many of the remaining challenges still preventing an even wider adoption. We are also delighted to see a great and increasing interest for ethical questions in the community, providing us with the confidence that AI will indeed deliver the positive impact it promises and is capable of.

Appendices

Deep Variational Reinforcement Learning

A.1 Experiments

A.1.1 Implementation Details

In our implementation, the transition and proposal distributions $p_\theta(Z_t \mid h_{t-1}, a_{t-1})$ and $q_\theta(Z_t|h_{t-1}, a_{t-1}, o_t)$ are multivariate normal distributions over Z_t whose mean and diagonal variance are determined by neural networks. For image data, the decoder $p_\theta(O_t \mid z_t, a_{t-1})$ is a multivariate independent Bernoulli distribution whose parameters are again determined by a neural network. For real-valued vectors we use a normal distribution.

When several inputs are passed to a neural network, they are concatenated to one vector. ReLUs are used as nonlinearities between all layers. Hidden layers are, if not otherwise stated, all of the same dimension as h. Batch normalisation was used between layers for experiments on Atari but not on Mountain Hike as they significantly hurt performance. All RNNs are GRUs.

Encoding functions φ^o, φ^a and φ^z are used to encode single observations, actions and latent states z before they are passed into other networks.

To encode visual observations, we use the the same convolutional network as proposed by Mnih et al. [217], but with only 32 instead of 64 channels in the final layer. The transposed convolutional network of the decoder has the reversed

structure. The decoder is preceeded by an additional fully connected layer which outputs the required dimension (1568 for Atari's 84×84 observations).

For observations in $\mathbb{R}^2$ we used two fully connected layers of size 64 as encoder. As decoder we used the same structure as for $p_\theta(Z|\dots)$ and $q_\phi(Z|\dots)$ which are all three normal distributions: One joint fully connected layer and two separated fully connected heads, one for the mean, one for the variance. The output of the variance layer is passed through a softplus layer to force positivity.

Actions are encoded using one fully connected layer of size 128 for Atari and size 64 for Mountain Hike. Lastly, z is encoded before being passed into networks by one fully connected layer of the same size as h.

The policy is one fully connected layer whose size is determined by the actions space, i.e. up to 18 outputs with softmax for Atari and only 2 outputs for the learned mean for Mountain Hike, together with a learned variance. The value function is one fully connected layer of size 1.

A2C used $n_e = 16$ parallel environments and $n_s = 5$-step learning for a total batch size of 80. Hyperparameters were tuned on *Chopper Command*. The learning rate of both DVRL and RNN was independently tuned on the set of values $\{3\times10^{-5}, 1\times 10^{-4}, 2\times10^{-4}, 3\times10^{-4}, 6\times10^{-4}, 9\times10^{-4}\}$ with 2×10^{-4} being chosen for DVRL on Atari and 1×10^{-4} for DVRL on MountainHike and RNN on both environments. Without further tuning, we set $\lambda^H = 0.01$ and $\lambda^V = 0.5$ as is commonly used.

As optimizer we use RMSProp with $\alpha = 0.99$. We clip gradients at a value of 0.5. The discount factor of the control problem is set to $\gamma = 0.99$ and lastly, we use 'orthogonal' initialization for the network weights.

The source code can be found here: `https://github.com/maximilianigl/DVRL`

A.1.2 Additional Experiments and Visualisations

Table A.1 shows the results on *deterministic* and flickering Atari, averaged over 5 random seeds. The values for DRQN and ADRQN are taken from the respective

papers. Note that DRQN and ADRQN rely on Q-learning instead of A2C, so the results are not directly comparable.

Figure A.1 and A.2 show individual learning curves for all 10 Atari games, either for the deterministic or the stochastic version of the games.

Table A.1: Final results on deterministic and flickering Atari environments, averaged over 5 random seeds. Bold numbers indicate statistical significance at the 5% level when comparing DVRL and RNN. The values for DRQN and ADRQN are taken from the respective papers.

Env	DVRL($\pm std$)	RNN	DRQN	ADRQN
Pong	**20.07**($\pm$0.39)	19.3($\pm$0.26)	12.1($\pm$2.2)	7($\pm$4.6)
Chopper	**6619**($\pm$532)	4619($\pm$306)	1330($\pm$294)	1608($\pm$707)
MsPacman	2156($\pm$127)	2113($\pm$135)	1739($\pm$942)	
Centipede	4171($\pm$127)	4283($\pm$187)	4319($\pm$4378)	
BeamRider	1901($\pm$67)	**2041**($\pm$81)	618($\pm$115)	
Frostbite	**296**($\pm$8.2)	259($\pm$5.7)	414($\pm$494)	2002($\pm$734)
Bowling	29.74($\pm$0.49)	29.38($\pm$0.52)	65($\pm$13)	
IceHockey	**−4.87**($\pm$0.24)	−6.49($\pm$0.27)	−5.4($\pm$2.7)	
DDunk	**−6.08**($\pm$3.08)	−15.25($\pm$0.51)	−14($\pm$2.5)	−13($\pm$3.6)
Asteroids	1610($\pm$63)	**1750**($\pm$97)	1032($\pm$410)	1040($\pm$431)

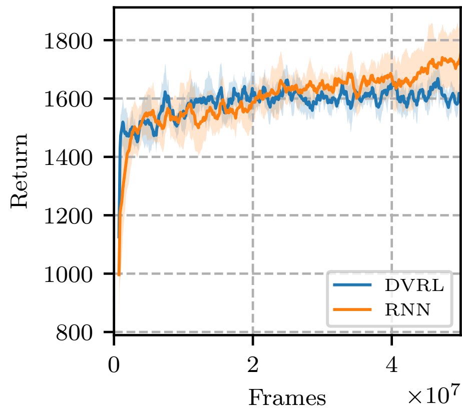
A.1.3 Computational Speed

The approximate training speed in frames per second (FPS) is on one GPU on a dgx1 for Atari:

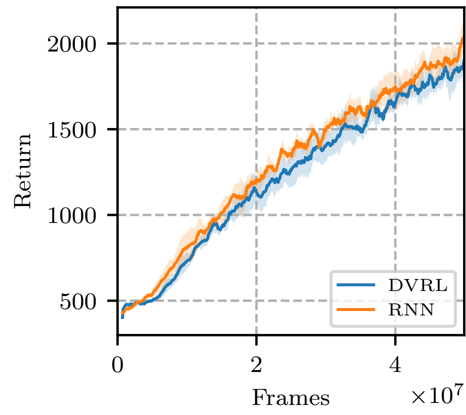
- RNN: 124k FPS
- DVRL (1 Particle): 64k FPS
- DVRL (10 Particles): 48k FPS
- DVRL (30 Particle): 32k FPS

A.1.4 Model Predictions

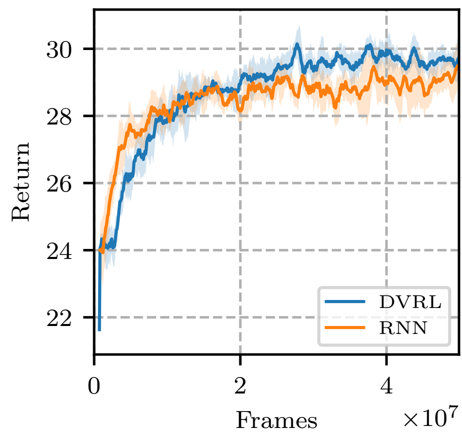
In Figure A.3 we show reconstructed and predicted images from the DVRL model for several Atari games. The current observation is in the leftmost column. The second column ('dt0') shows the reconstruction after encoding and decoding the current



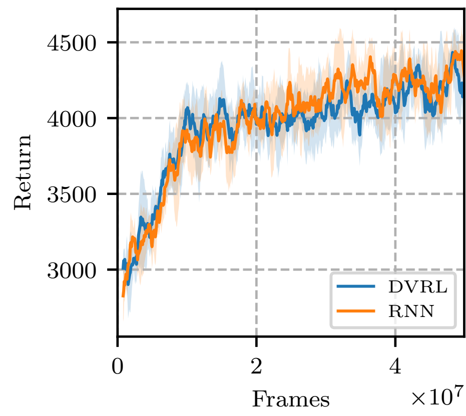
(a) Asteroids



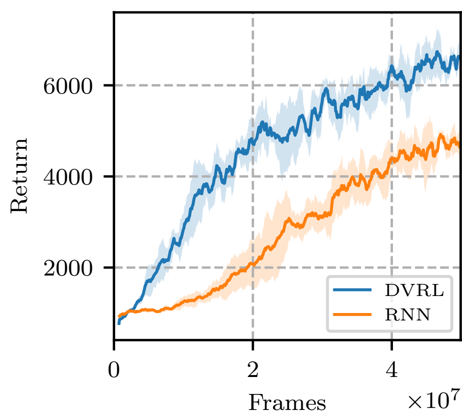
(b) Beam Rider



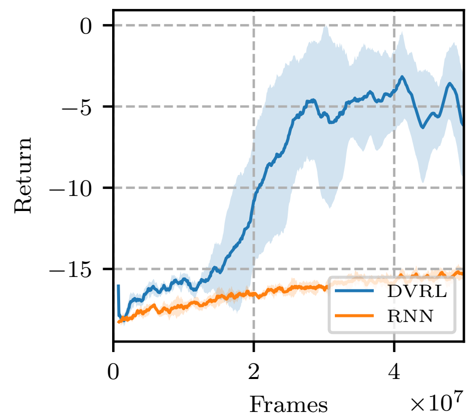
(c) Bowling



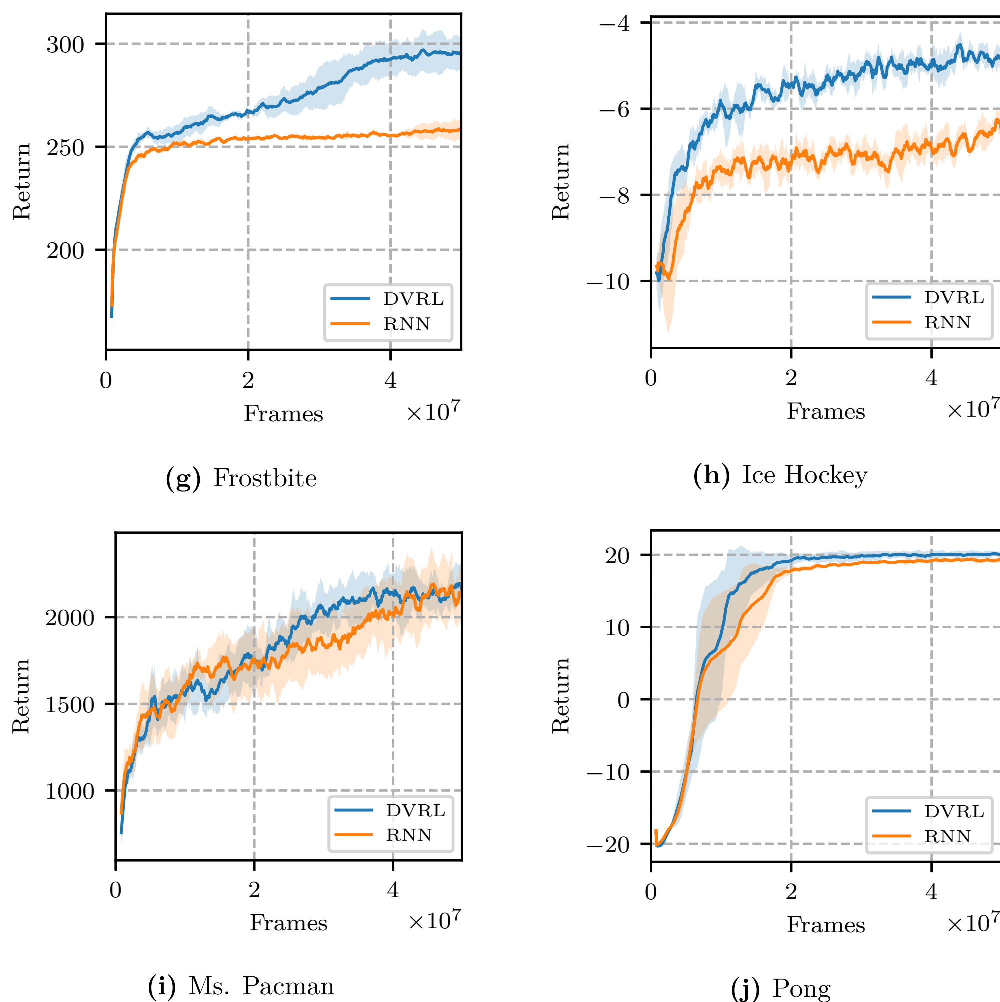
(d) Centipede



(e) Chopper Command



(f) Double Dunk



(g) Frostbite

(h) Ice Hockey

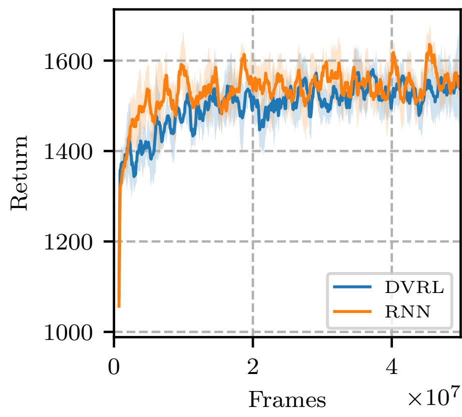
(i) Ms. Pacman

(j) Pong

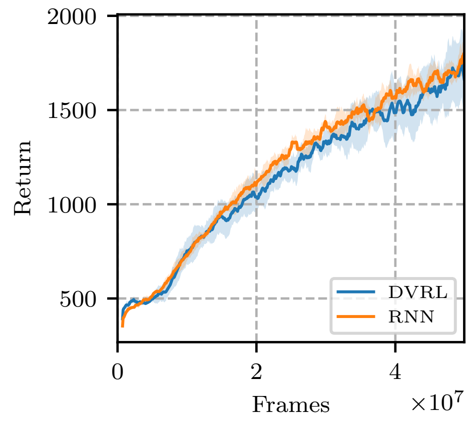
Figure A.1: Training curves on the full set of evaluated Atari games, in the case of flickering and *deterministic* environments.

observation. For the further columns, we make use of the learned generative model to predict future observations. For simplicity we repeat the last action. Columns 2 to 7 show predicted observations for $dt \in \{1, 2, 3, 10, 30\}$ unrolled timesteps. The model was trained as explained in the main chapter. The reconstructed and predicted images are a weighted average over all 16 particles.

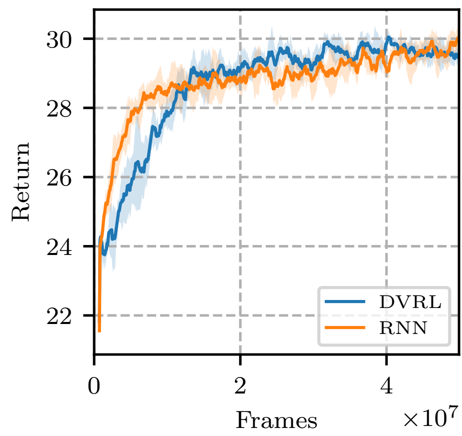
Note that the model is able to correctly predict features of future observations, for example the movement of the cars in ChopperCommand, the (approximate) ball position in Pong or the missing pins in Bowling. Furthermore, it is able to do so, even if the current observation is blank like in Bowling. The model has also correctly learned to randomly predict blank observations.



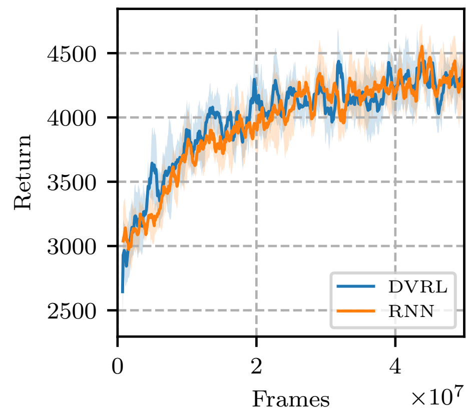
(a) Asteroids



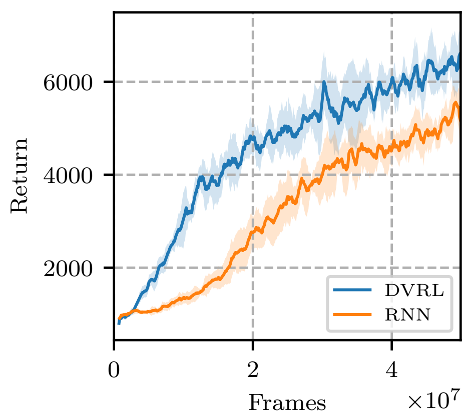
(b) Beam Rider



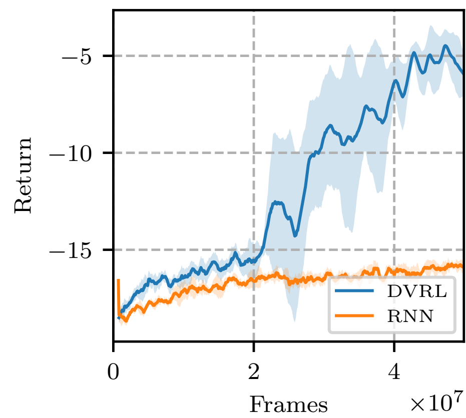
(c) Bowling



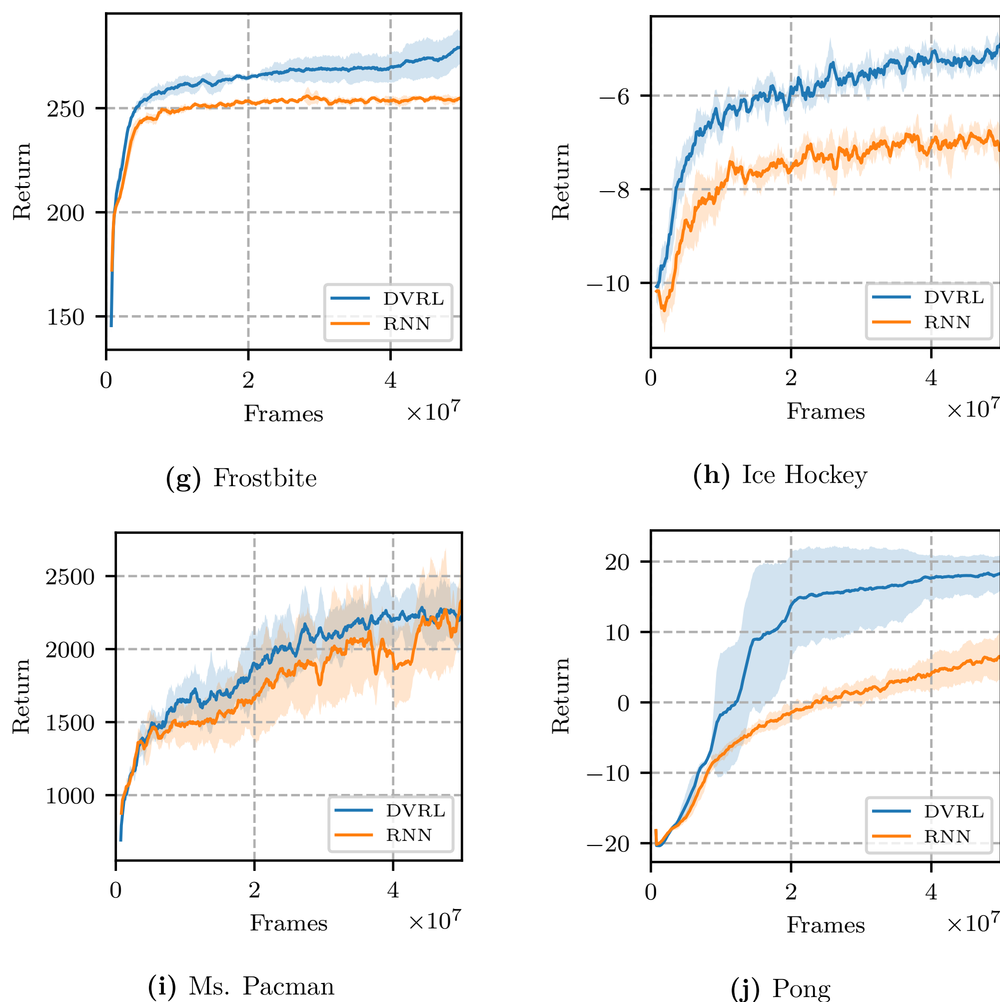
(d) Centipede



(e) Chopper Command



(f) Double Dunk



(g) Frostbite

(h) Ice Hockey

(i) Ms. Pacman

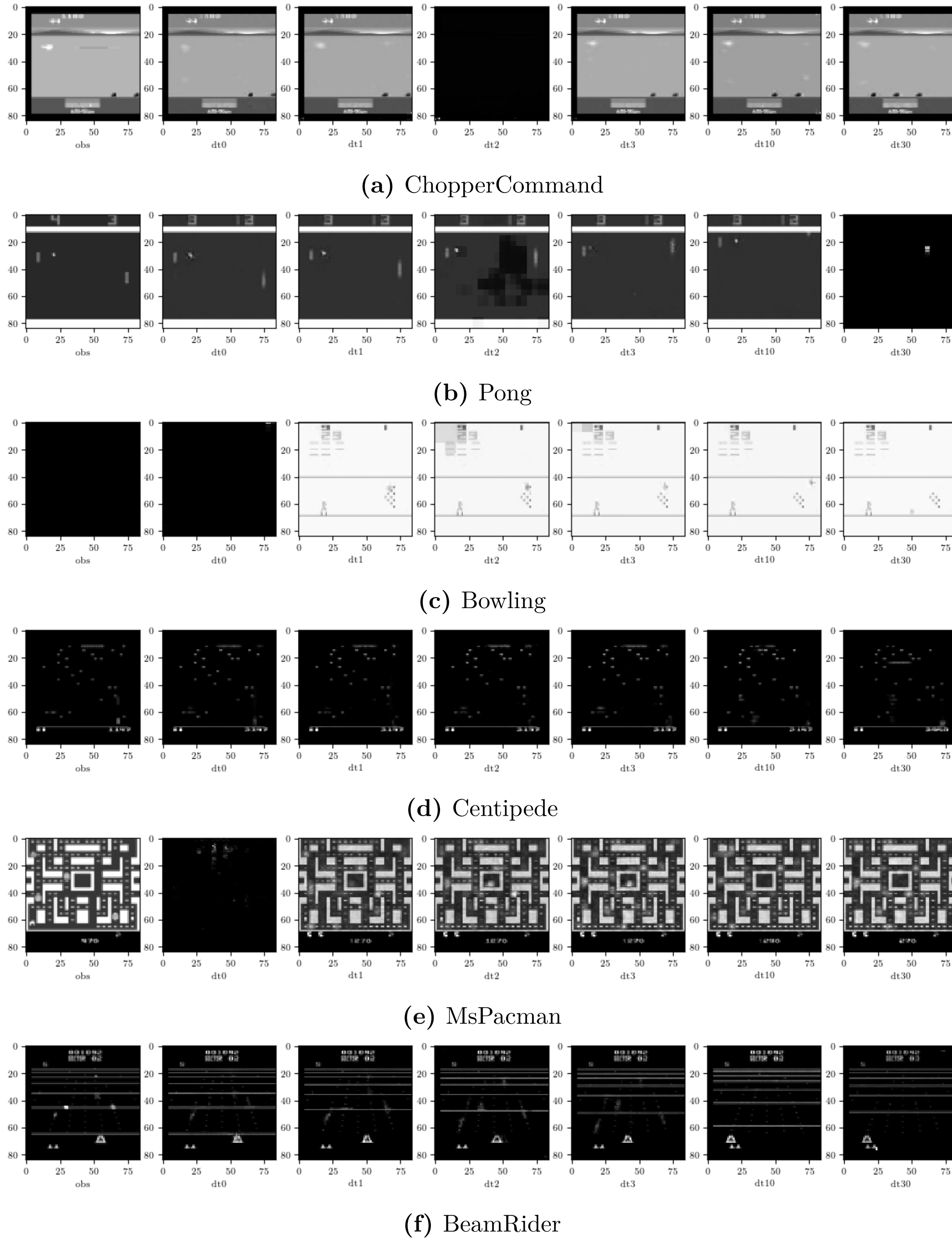
(j) Pong

Figure A.2: Training curves on the full set of evaluated Atari games, in the case of flickering and *stochastic* environments.

It can remember feature of the current state fairly well, like the positions of barriers (white dots) in Centipede. On the other hand, it clearly struggles with the amount of information present in MsPacman like the positions of all previously eaten fruits or the location of the ghosts.

A.2 Algorithms

Algorithm 1 details the recurrent (belief) state computation (i.e. history encoder) for DVRL. Algorithm 2 details the recurrent state computation for RNN. Algorithm 3 describes the overall training algorithm that either uses one or the other to aggregate the history. Despite looking complicated, it is just a very detailed implementation



(a) ChopperCommand

(b) Pong

(c) Bowling

(d) Centipede

(e) MsPacman

(f) BeamRider

Figure A.3: Reconstructions and predictions using the learned generative model for several Atari games. First column: Current observation (potentially blank). Second column: Encoded and decoded reconstruction of the current observation. Columns 3 to 7: Predicted observations using the learned generative model for timesteps $dt \in \{0, 1, 2, 3, 10, 30\}$ into the future.

of n-step A2C with the additional changes: Inclusion of L^{ELBO} and inclusing of the option to not delete the computation graph to allow longer backprop in n-step A2C.

Results for also using the reconstruction loss L^{ENC} for the RNN based encoder aren't shown in the paper as they reliably performed worse than RNN without reconstruction loss.

Algorithm 1: DVRL encoder

1. **Input:** Previous state $\hat{b}_{t-1}$, observation o_t, action a_{t-1}
2. Unpack $w_{t-1}^{1:K}, z_{t-1}^{1:K}, h_{t-1}^{1:K}, \hat{h}_{t-1} \leftarrow \hat{b}_{t-1}$
3. $x^o \leftarrow \varphi_\theta^o(o_t)$
4. $x^a \leftarrow \varphi_\theta^a(a_{t-1})$
5. **for** $k = 1$ ***to*** K **do**
6. Sample $h_{t-1}^k \sim h_{t-1}^{1:K}$ based on weights
7. Sample $z_t^k \sim q_\theta(z_t^k | h_{t-1}^k, x^o, x^a)$
8. $x^z \leftarrow \varphi_\theta^z(z_t)$
9. $w_j^k \leftarrow p_\theta(z_t^k | h_{t-1}^k, x^a) p_\theta(o_t | h_t^k, x^z, x^a) / q_\theta(z_t^k | h_{t-1}^k, x^o, x^a)$
10. $h_t^k \leftarrow \text{GRU}(h_{t-1}^k, x^z, x^o, x^a)$
11. $L_t^{\text{ELBO}} \leftarrow -\log \sum_k w_t^k - \log(K)$
12. $\hat{h}_t \leftarrow \text{GRU}(\text{Concat}(w_t^k, x^z, h_t^k)_{k=1}^K \text{passed sequentially})$
13. Pack $\hat{b}_t \leftarrow w_t^{1:K}, z_t^{1:K}, h_t^{1:K}, \hat{h}_t$
14. *// When V or π is conditioned on $\hat{b}_t$, the summary $\hat{h}_t$ is used.*
15. **Output:** $\hat{b}_t, L_t^{\text{ELBO}}$

Algorithm 2: RNN encoder

1. **Input:** Previous state h_{j-1}, observation o_j, action a_{j-1}
2. $x^o \leftarrow \varphi_\theta^o(o_t)$
3. $x^a \leftarrow \varphi_\theta^a(a_{t-1})$
4. $\hat{b}_j \leftarrow \text{GRU}_\theta(\hat{b}_{j-1}, x^o, x^a)$
5. $L_j^{\text{ENC}} \leftarrow -\log p_\theta(o_j | \hat{b}_{j-1})$
6. **Output:** h_j, L_j^{ENC}

Algorithm 3: Training Algorithm

1 **Input:** Environment Env, Encoder $\text{Enc}_{\theta,\phi}$ (either RNN or DVRL)
2 Initialize observation o_1 from Env.
3 Initialize encoder latent state $s_0 \leftarrow s_0^{init}$ as either h_0 (for RNN) or $\hat{b}_{0,\theta}$ (for DVRL)
4 Initialize action $a_0 = 0$ to no-op
5 Set $s'_0, a'_0 \leftarrow s_0, a_0$.
6 *The distinction between s'_t and s_t is necessary when the environment resets at time t.*
7 **while** not converged **do**
8 $\quad L_j^{Enc}, a_j, a'_j, s_j, s'_j, o_{j+1}, r_{j+1}, done_{j+1} \leftarrow NULL \quad j = 1 \ldots n$
9 $\quad$ *// Run n steps forward:*
10 $\quad$ **for** $j = 1$ ***to*** n **do**
11 $\quad\quad s_j, L_j^{\text{ELBO}} \leftarrow \text{Enc}_{\theta,\phi}(o_j, a'_{j-1}, s'_{j-1})$
12 $\quad\quad$ Sample $a_j \sim \pi_\rho(a_j|s_j)$
13 $\quad\quad o_{j+1}, r_{j+1}, done_{j+1} \leftarrow \text{Env}(a_j)$
14 $\quad\quad$ **if** $done_{j+1}$ **then**
15 $\quad\quad\quad s'_j, a'_j \leftarrow s_0^{init}, 0$
16 $\quad\quad\quad$ *// s_j is still available to compute $V_\eta(s_j)$*
17 $\quad\quad\quad o_{j+1} \leftarrow$ Reset Env()
18 $\quad\quad$ **else**
19 $\quad\quad\quad s'_j, a'_j \leftarrow s_j, a_j$
20 $\quad$ *// Compute targets* $s_{n+1} \leftarrow \text{Enc}_{\theta,\phi}(o_{n+1}, a'_n, s'_n)$
21 $\quad Q_{n+1}^{target} \leftarrow V_\eta(s_{n+1}).detach()$
22 $\quad$ **for** $j = n$ ***to*** 1 **do**
23 $\quad\quad Q_j^{target} \leftarrow \gamma \cdot Q_{j+1}^{target}$
24 $\quad\quad$ **if** $done_{j+1}$ **then**
25 $\quad\quad\quad Q_j^{target} \leftarrow 0$
26 $\quad\quad Q_j^{target} \leftarrow Q_j^{target} + r_{j+1}$
27 $\quad$ *Compute losses*
28 $\quad$ **for** $j = n$ ***to*** 1 **do**
29 $\quad\quad L_j^V \leftarrow (Q_j^{target} - V_\eta(s_j))^2$
30 $\quad\quad L_j^A \leftarrow -\log \pi_\rho(a_j|s_j)(Q_j^{target} - V_\eta(s_j))$
31 $\quad\quad L_j^H \leftarrow -\text{Entropy}(\pi_\rho(\cdot|s_j))$
32 $\quad \mathcal{J} \leftarrow \sum_j (\lambda^V L_j^V + L_j^A + \lambda^H L_j^H + \lambda^E L_j^{\text{ELBO}})$
33 $\quad$ TakeGradientStep($\nabla \mathcal{J}$)
34 $\quad$ Delete or save computation graph of s_n to determine backpropagation length
35 $\quad a'_0, s'_0 \leftarrow a_n, s_n$
36 $\quad o_1 \leftarrow o_{n+1}$

B
Multitask Soft Option Learning

B.1 Pseudo code

Please see Algorithm 4 for pseudo code of MSOL.

B.2 MSOL training details

B.2.1 Optimisation

Even though R_i^{reg} depends on ϕ_i, its gradient w.r.t. ϕ_i vanishes.[1] Consequently, we can treat the regularized reward as a classical RL reward and use any RL algorithm to find the optimal hierarchical policy parameters ϕ_i. In the following, we explain how to adapt A2C [54] to soft options. The extension to PPO [39] is straightforward.[2]

The joint posterior policy in (4.4) depends on the current state s_t and the previously selected option z_{t-1}. The expected sum of regularized future rewards of task i, the value function V_i, must therefore also condition on this pair:

$$V_i(s_t, z_{t-1}) \;:=\; \mathbb{E}_{\tau\sim q}\left[\sum_{t'=t}^{T} \gamma^{t'-t} R_{i,t'}^{\text{reg}} \,\middle|\, s_t, z_{t-1}\right]. \tag{B.1}$$

As $V_i(s_t, z_{t-1})$ cannot be directly observed, we approximate it with a parametrized model $V_{\phi_i}(s_t, z_{t-1})$. The k-step advantage estimation at time t of trajectory τ is given

[1] $\int p(x)\,\nabla \ln p(x)\,dx = \int \nabla p(x)\,dx = \nabla \int p(x)\,dx = 0$.

[2] However, for PAI frameworks like ours, unlike in the original PPO implementation, the advantage function must be updated after each epoch.

by

$$A_{\phi_i}(\tau_{t:(t+k)}) \quad := \quad \sum_{j=0}^{k-1} \gamma^j R^{\text{reg}}_{t+j} + \gamma^k V^{-}_{\phi_i}(s_{t+k}, z_{t+k-1}) - V_{\phi_i}(s_t, z_{t-1}) \,, \tag{B.2}$$

where the superscript '$-$' indicates treating the term as a constant. The approximate value function V_{ϕ_i} can be optimized towards its bootstrapped k-step target by minimizing $\mathcal{L}_V(\phi_i, \tau_{1:T}) := \sum_{t=1}^{T} (A_{\phi_i}(\tau_{t:(t+k)}))^2$. As per A2C, $k \in [1 \dots n_s]$ depending on the state [54]. The corresponding policy gradient loss is

$$\mathcal{L}_A(\phi_i, \tau_{1:T}) := \sum_{t=1}^{T} A^{-}_{\phi_i}(\tau_{t:(t+k)}) \ln q_{\phi_i}(a_t, z_t, b_t | s_t, z_{t-1}) \,.$$

The gradient w.r.t. the prior parameters θ is[3]

$$\nabla_\theta \mathcal{L}_P(\theta, \tau_{1:T}, \tilde{b}_{1:T}) \quad := \quad -\sum_{t=1}^{T} \Big(\nabla_\theta \ln p^L_\theta(a_t | s_t, z_t) + \nabla_\theta \ln p^T_\theta(\tilde{b}_t | s_t, z_{t-1}) \Big) \,, \tag{B.3}$$

where $\tilde{b}_t = \delta_{z_{t-1}}(z'_t)$ and $z'_t \sim q^H(z'_t | s_t, z_{t-1}, b_t = 1)$. To encourage exploration in all policies of the hierarchy, we also include an entropy maximization loss:

$$\mathcal{L}_H(\phi_i, \tau_{1:T}) \quad := \quad \sum_{t=1}^{T} \Big(\ln q^H_{\phi_i}(z_t | s_t, z_{t-1}, b_t) + \ln q^L_{\phi_i}(a_t | s_t, z_t) + \ln q^T_{\phi_i}(b_t | s_t, z_{t-1}) \Big) \,. \tag{B.4}$$

Note that term ① in (4.7) already encourages maximizing $\mathcal{L}_H(\phi_i, \tau)$ for the master policy, since we chose a uniform prior $p^H(z_t | b_t = 1)$. As both terms serve the same purpose, we are free to drop either one of them. In our experiments, we chose to drop the term for q^H in R^{reg}_t, which proved slightly more stable to optimize than the alternative.

We can optimize all parameters jointly with a combined loss over all tasks i, based on sampled trajectories $\tau^i := \tau^i_{1:T} \sim q_{\phi_i}$ and corresponding sampled values of $\tilde{b}^i := \tilde{b}^i_{1:T}$:

$$\mathcal{L}(\{\phi_i\}, \theta, \{\tau^i\}, \{\tilde{b}^i\}) = \sum_{i=1}^{n} \Big(\mathcal{L}_A(\phi_i, \tau^i) + \lambda_V \mathcal{L}_V(\phi_i, \tau^i) + \lambda_P \mathcal{L}_P(\theta, \tau^i, \tilde{b}^i) + \lambda_H \mathcal{L}_H(\phi_i, \tau^i) \Big) \,.$$

[3]Here we ignore β as it is folded into λ_P later.

B.2.2 Training schedule

For faster training, it is important to prevent the master policies q^H from converging too quickly to allow sufficient updating of all options. On the other hand, a lower exploration rate leads to more clearly defined options. We consequently anneal the exploration bonus λ_H with a linear schedule during training.

Similarly, a high value of β leads to better options but can prevent finding the extrinsic reward $r_i(s_t, a_t)$ early on in training. Consequently, we increase β over the course of training, also using a linear schedule.

B.3 Architecture

All policies and value functions share the same encoder network with two fully connected hidden layers of size 64 for the Moving Bandits environment and three hidden layers of sizes 512, 256, and 512 for the Taxi environments. Distral was tested with both model sizes on the Moving Bandits task to make sure that limited capacity is not the problem. Both models resulted in similar performance, the results shown in the chapter are for the larger model. Master-policies, as well as all prior- and posterior policies and value functions consist of only one layer which takes the latent embedding produced by the encoder as input. Furthermore, the encoder is shared across tasks, allowing for much faster training since observations can be batched together.

Options are specified as an additional one-hot encoded input to the corresponding network that is passed through a single 128 dimensional fully connected layer and concatenated to the state embedding before the last hidden layer. We implement the single-column architecture of Distral as a hierarchical policy with just one option and with a modified loss function that does not include terms for the master and termination policies. Our implementation builds on the A2C/PPO implementation by, and we use the implementation for MLSH that is provided by the authors (`https://github.com/openai/mlsh`).

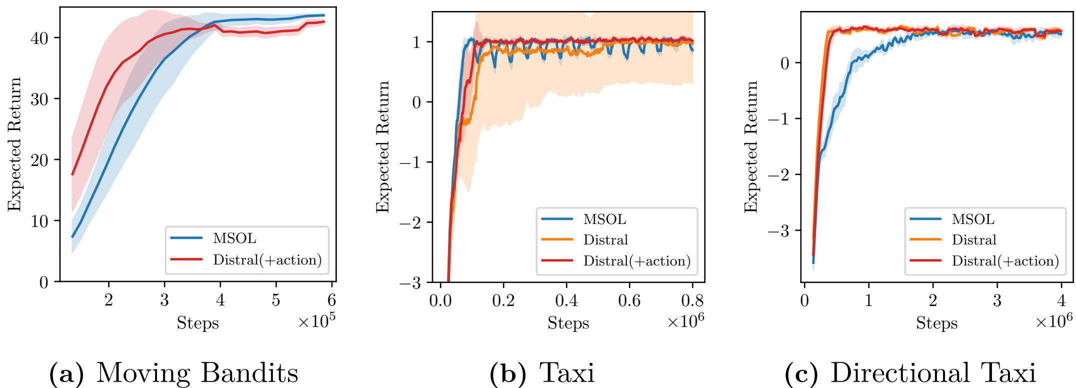
B.4 Hyper-parameters and additional environment details

We use $2\lambda_V = \lambda_A = \lambda_P = 1$ in all experiments. Furthermore, we train on all tasks from the task distribution, regularly resetting individual tasks by resetting the corresponding master and re-initializing the posterior policies. Optimizing β for MSOL and Distral was done over $\{0.01, 0.02, 0.04, 0.1, 0.2, 0.4\}$. We use $\gamma = 0.95$ for Moving Bandits and Taxi.

B.4.1 Moving bandits

For MLSH, we use the original hyper-parameters [108]. The duration of each option is fixed to 10. The required warm-up duration is set to 9 and the training duration set to 1. We also use 30 parallel environments split between 10 tasks. This and the training duration are the main differences to the original paper. Originally, MLSH was trained on 120 parallel environments which we were unable to do due to hardware constraints. Training is done over 6 million frames per task.

For MSOL and Distral we use the same number of 10 tasks and 30 processes. The duration of options are learned and we do not require a warm-up period. We set the learning rate to 0.01 and $\beta = 0.2$, $\alpha = 0.95$, $\lambda_H = 0.05$. Training is



(a) Moving Bandits **(b)** Taxi **(c)** Directional Taxi

Figure B.1: Performance during training phase. Note that MSOL and MSOL(frozen) share the same training as they only differ during testing. Further, note that the highest achievable performance for Taxi and Directional Taxi is higher during training as they can be initialized closer to the final goal (i.e. with the passenger on board).

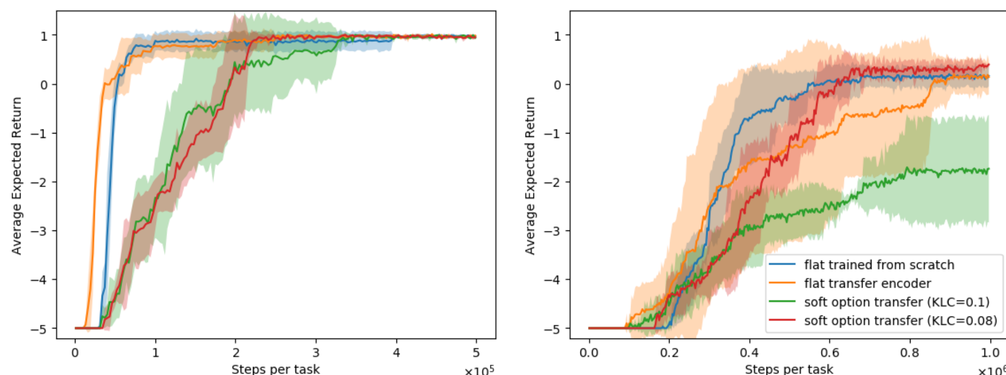


Figure B.2: Results on a 'further modified' taxi environment in which the goal locations at test time were shifted compared to training, making the learned options misspecified, similar to Figs. 4.5b and 4.5c. Here, the goal locations were shifted further, making the options more misspecified. *Left:* Results on a 'small' 8x8 grid. *Right:* Results on a 'large' 10x10 grid.

done over 0.6 million frames per task. For Distral we use $\beta = 0.04$, $\lambda_H = 0.05$ and also 0.6 million frames per task.

B.4.2 Taxi

For MSOL we anneal β from 0.02 to 0.1 and λ_H from 0.1 to 0.05. For Distral we use $\beta = 0.04$. We use 3 processes per task to collect experience for a batch size of 15 per task. Training is done over 1.4 million frames per task for *Taxi* and 4 million frames per task for *Directional Taxi*. MLSH was trained on 0.6 million frames for *Taxi* as due to it's long runtime of several days, using more frames was infeasible. Training was already converged.

Tasks further out of distribution In Fig. B.2 we provided additional results for Section 4.5.3. We show the performance on *further modified* environments, for which the goal locations were moved by a second block from the original location for which the options were trained. We only compare soft options with flat policies trained from scratch, as we already showed in Section 4.5.3 that hard options are unable to cope well with goal modifications.

As expected, the flat policy trained from scratch performs similarly as before, as the moved goal location does not impact it much. On the other hand, using

a pre-trained encoder performs slightly worse. On the smaller task (left figure) the options are too misspecified to be competitive, despite being soft. For the larger grid (right figure) and for a sufficienlty small value of β ('KLC'), the options, despite misspecification, are still competitive.

Consequently, while there is no hard limitation of our approach for appropriately chosen β, if the target task is too different from the source task, it will be faster to learn a new policy from scratch. Which algorithm trains faster depends mainly on the difficulty of exploration in the target task. Hard exploration makes options more useful compared to a new, flat policy, even if the options are misspecified. However, the more misspecified the options are, the smaller the advantage. If target and source task are too different, very little positive transfer can be expected, and learning a new (flat) policy becomes more efficient.

Algorithm 4: Pseudo-Code for MSOL

1 **Input** Number m of options to learn, n_i different training tasks env$_i$, *fixed* termination prior $p^T(b_t) = (1-\alpha)^{b_t}\alpha^{1-b_t}$ and *fixed* master prior $p^H(z_t|z_{t-1}, b_t) = (1-b_t)\,\delta(z_t - z_{t-1}) \,+\, b_t\frac{1}{m}$

2 **Initialize once:** *Learnable* termination prior $p^T_\theta(b_t|s_t, z_{t-1})$, intra-option prior $p^L_\theta(a_t|s_t, z_t)$

3 **Initialize for each task** i**:** Lernable termination posterior $q^T_{\phi_i}(b_t|s_t, z_{t-1})$, master policy $q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)$, intra-option policy $q^L_{\phi_i}(a_t|s_t, z_t)$

4 // *Note that this leads to a total of* m *intra option priors for the* m *different values of* $z \in \{1 \ldots m\}$

5 // *and a total of* $m \times n_i$ *intra option posteriors.*

6 **while** not converged **do**

7   // *Collect data*

8   **for** each task i **do**

9   **if** beginning of episode **then**

10             $s_0 \leftarrow$ env.reset()

11             $b_0 \leftarrow 1$ // *This allows* q^H *to sample a new option* z_0*.*

12   **else**

13             $b_t \sim q^T_{\phi_i}(b_t|s_t, z_{t-1})$

14         $z_t \sim q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)$

15         $a_t \sim q^L_{\phi_i}(a_t|s_t, z_t)$

16         $s_{t+1}, r_t \sim \text{env}_i(a_t)$

17   // *Compute regularized reward (eq. Eq.* (4.7)*); note that we use the fixed priors here* $R^{\text{reg}}_t \leftarrow$
$r_i(s_t, a_t) - \beta \ln \frac{q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)}{p^H(z_t|z_{t-1}, b_t)} - \beta \ln \frac{q^L_{\phi_i}(a_t|s_t, z_t)}{p^L_\theta(a_t|s_t, z_t)} - \beta \ln \frac{q^T_{\phi_i}(b_t|s_t, z_{t-1})}{p^T(b_t)}$

18   Add $s_t, s_{t+1}, r_t, R^{\text{reg}}_t$ to $\mathcal{D}_i$

19   // *Update parameters* ϕ_i

20   **for** each task i **do**

21   Update ϕ_i using A2C or PPO on $\mathcal{D}_i$ as desribed in Appendix B.2.1. Note that for PPO, R^{reg}_t needs to be re-computed and updated between gradient updates to ϕ_i as the regularisation terms change.

22   // *Update parameters* θ

23   Update θ to minimize
$\sum_i \mathbb{E}_{\mathcal{D}_i}\left[\mathbb{D}_{\text{KL}}\left(q^L_{\phi_i}(a_t|s_t, z_t) \,\|\, p^L_\theta(a_t|s_t, z_t)\right) + \mathbb{D}_{\text{KL}}\left(\hat{q}_{\phi_i}(b=1|s_t, z_{t-1}) \,\|\, p^T_\theta(b_t|s_t, z_{t-1})\right)\right]$

24   Here, $\hat{q}_{\phi_i}(b=1|s_t, z_{t-1}) = \sum_{z_t \neq z_{t-1}} q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t = 1)$, i.e. instead of distilling the average of $q^T_{\phi_i}$ into p^T_θ, we distill whether the *master policy* $q^H_{\phi_i}$ would have changed the option z_t if it had the chance (i.e. if $b_t = 1$). Since $q^T_{\phi_i}$ is regularized to be similar to the *fixed* p^T, this approach allows us to learn a termination prior p^T_θ which is less influenced by our manually specified prior p^T, and more by what is needed for the task.

C

Information Bottleneck Actor Critic

C.1 Dropout with SNI

In order to apply SNI to Dropout, we need to decide how to 'suspend' the noise to compute $\bar{\pi}_\theta$. While one could apply no dropout mask and scale the activations accordingly, we empirically found it to be better to instead sample one dropout mask and keep it fixed for all gradient updates using the thus collected data. This follows the implementation used in [145].

C.2 Supervised Classification Task

Network architecture and hyperparameters The network consist of a 1D-convolutional layer with 10 filters and a kernel size of 11 followed by two hidden, fully connected layers of size 1024 and 256 and the last layer which outputs n_c logits. When the VIB or Dropout are used, they are applied to the last hidden layer. We use a learning rate of $1e-4$. The relative weight for weight decay was $\lambda_w = 1e-3$, which performed best out of $\{1e-2, 1e-3, 1e-4, 1e-5\}$. For the VIB we used $\beta = 1e-3$, which performed best out of $\{1e-2, 1e-3, 1e-4, 1e-5\}$. Lastly, For dropout we tested the dropout rates $p_d \in \{0.1, 0.2, 0.5\}$, out of which 0.2 performed best. Our results were stable across a range of hyperparameters, see Fig. C.2.

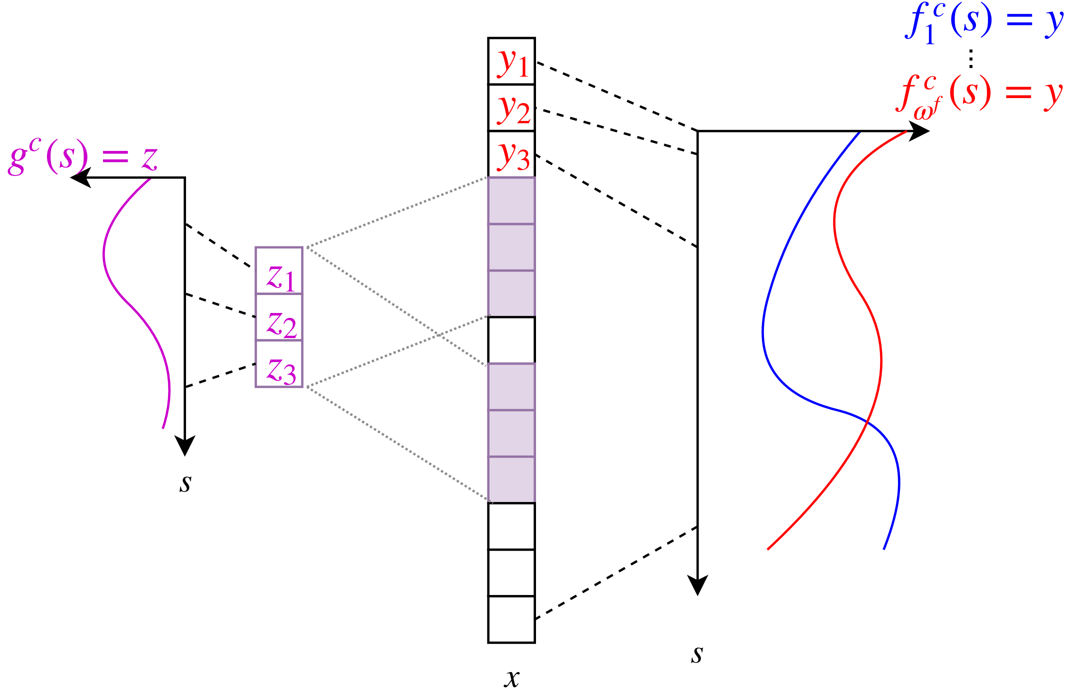


Figure C.1: Generation of the input data x: We embed the information about c twice, once through f^c and once through g^c. See text for details.

Data generation For each data-point, after drawing the class label c_i, we want to encode the information about c_i in two ways, using the encoding functions f^c and g^c which use one of ω^f and ω^g different patterns to encode the information. The larger ω is, the less *general* the encoding is as it applies to fewer data-points. Note that there are ω different patterns *per class*.

We generate the patterns by first generating a set of random functions $\{f_j^c\}_{j=1}^{\omega^f}$ and $\{g_j^c\}_{j=1}^{\omega^g}$ by randomly drawing Fourier coefficients from $[0, 1]$. Those functions are converted into vectors by evaluating them at d_x sorted points randomly drawn from $[0, 1]$. The resulting pattern-vectors for f^c will have a dimension of d_x, whereas the ones for g^c will be smaller, $d_g < d_x$.

To encode the information about c_i we first choose one pattern from $\{f_j^c\}_j$ (slightly overloading notation between functions and pattern-vectors) and add some noise:

$$x_i' = f_j^c + \epsilon_i \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon) \quad \text{and} \quad j \sim Cat(\omega^f) \tag{C.1}$$

Next, to also encode the information about c_i using g^c, we choose one of the

ω^g patterns $\{g_j^c\}_j$ and *replace a part of the vector* x_i', which is possible because the g^c patterns are shorter: $d_g < d_x$. The location of replacement is randomly drawn for each data-point, but restricted to a a set of n_g possible locations which are also random, but kept fixed for the experiment and the same between training and testing set. The process is pictured in Fig. C.1.

By changing the number of possible locations n_g and the strength of the noise added to f^c, σ_ϵ,we can tune the relative difficulty of learning to recognize patterns g^c and f^c, allowing us to find a regime where both *can* be found. Within this regime, our qualitative results were stable. We use $n_g = 3$ and $\sigma_\epsilon = 1$. Furthermore, we have for the dimension of of the observations $d_x = 100$, and for the size of the patterns g^c we have $d_g = 20$. We use $n_c = 5$ different classes.

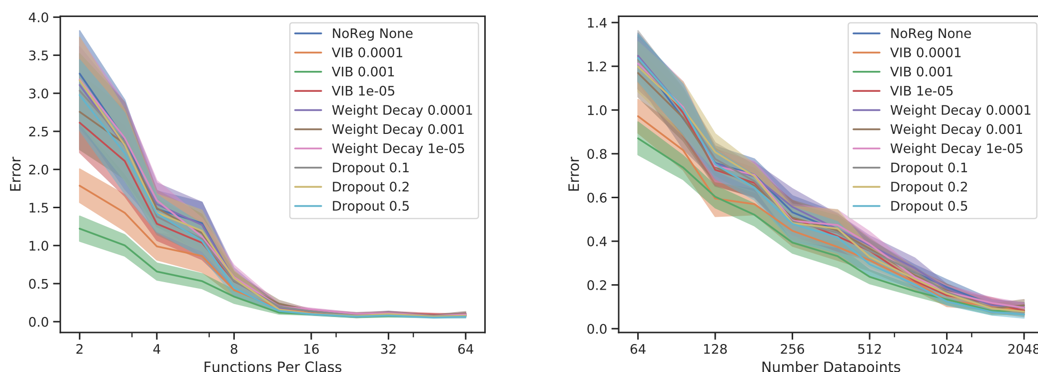


Figure C.2: Loss function (error) on test set. Same results as in main text, but for multiple hyperparameters. The qualitative results are stable under a wide range of hyperparameters.

C.3 Multiroom

The observation space measures $11 \times 11 \times 3$ where the 3 channels are used to encode object type and object features like orientation or 'open/closed' and 'color' for doors on each of the 11×11 spatial locations (see Fig. 5.3 for a typical layout for $n_r = 3$).

The agent uses a 3-layer CNN with $16, 32$ and 32 filters respectively. All layers use a kernel of size 2. After the CNN, it uses one hidden layer of size 64 to which IBAC or Dropout are applied if they are used. Dropout uses $p_d = 0.2$ and was

tested for $\{0.1, 0.2, 0.5\}$. Both weight decay and IBAC were tried with a weighting factor of $\{1e-3, 1e-4, 1e-5, 1e-6\}$, with $1e-4$ performing best for weight decay and $1e-6$ performing best for IBAC. The output of the hidden layer is fed into a value function head and the policy head.

We use a discount factor $\gamma = 0.99$, a learning rate of $7e-4$, generalized value estimation with$\lambda_{\text{GAE}} = 0.95$ [38], an entropy coefficient of $\lambda_H = 0.01$, value loss coefficient $\lambda_V = 0.5$, gradient clipping at 0.5 [38], and PPO with the Adam optimizer [35].

C.4 Coinrun

Architecture and Hyperparameters We use the same architecture ('Impala') and default policy gradient hyperparameters as well as the codebase (`https://github.com/openai/coinrun`) from the authors of [145] to ensure staying as closely as possible to their proposed benchmark.

Dropout and IBAC where applied to the last hidden layer and both, as well as weight decay, were tried with the same set of hyperparameters as in Multiroom. The best performance was achieved with $p_d = 0.2$ for Dropout and $1e-4$ for IBAC and weight decay. Batch normalisation was applied between the layers of the convolutional part of the network. Note that the original architecture in [145] uses Dropout also on earlier layers, however, we achieve higher performance with our implementation.

In Fig. C.3 (left) we show results for Dropout with and without SNI and for $\lambda = 1$ and $\lambda = 0.5$. We find that $\lambda = 1$ learns fastest, possible due to the high importance weight variance in the stochastic term in SNI for $\lambda < 1$ (see Fig. 5.4 (right)). However, all Dropout implementations converge to roughly the same value, significantly below the 'baseline' agent, indicating that Dropout is not suitable for combination with weight decay and data augmentation.

In Fig. C.3 (right) we show the test performance for IBAC and Dropout with and without SNI, *without* using weight decay and data-augmentation. Again, we can see that SNI helps the performance. Interestingly, we can see that IBAC does not prevent

overfitting by itself (one can see the performance decreasing for longer training) but does lead to faster learning. Our conjecture is that it finds more general features early on in training, but ultimately overfits to the test-set of environments without additional regularisation. This further indicates that it's regularisation is different to techniques such as weight decay, explaining why their combination synergizes well.

In Fig. C.4 we show the training set performance of our experiments.

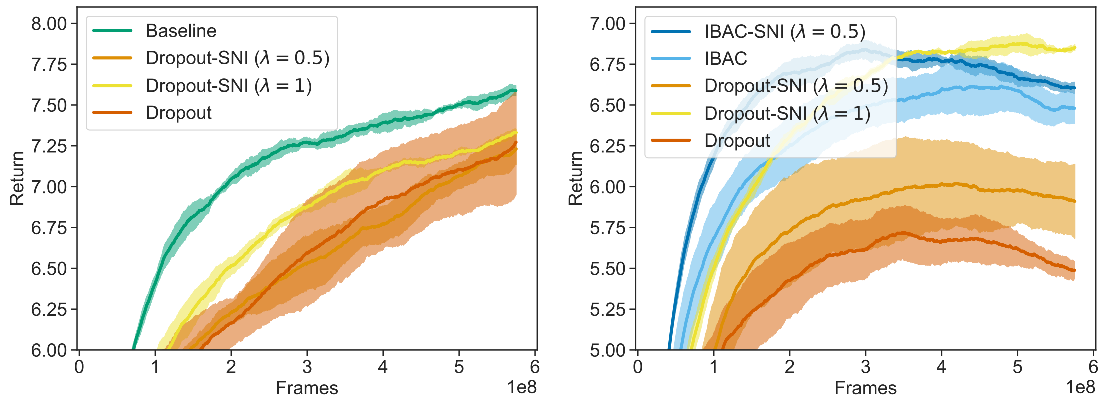


Figure C.3: *Left:* Comparison for different implementations of Dropout on the test environments. *Right:* Comparison of IBAC and Dropout, with and without SNI, *without weight decay and data augmentation.*

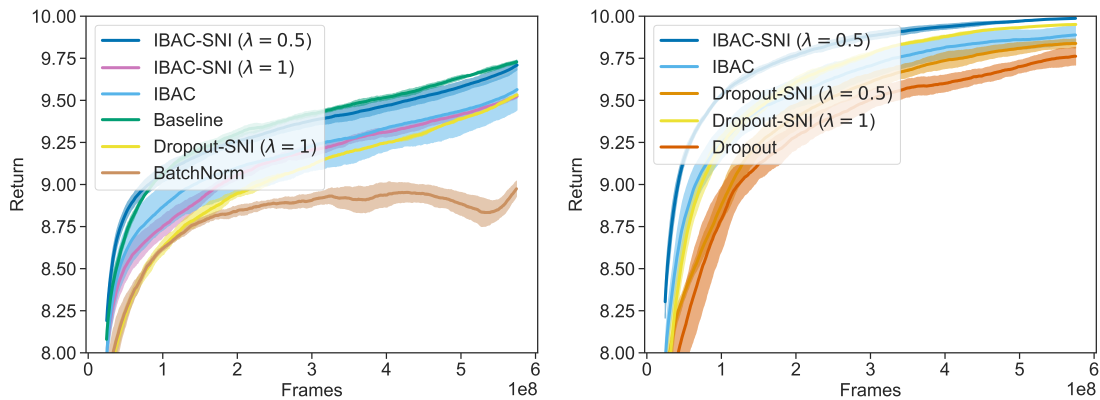


Figure C.4: Training Performance with weight decay and data augmentation (left) and without (right)

D

Iterated Relearning

D.1 Pseudo code

Algorithm 5: Pseudo-Code for parallel ITER

```
1  Input Length of initial RL training phase t_init, length of distillation phase
     t_distill
2  Initialise k ← 0, policy π^(k), value function V^(k)
3  // Normal RL training at the beginning
4  for t_init steps do
5  |  B ← collect trajectory data using π^(0)
6  |  Update π^(0) and V^(0) using standard RL method using B
7  // Combine further RL training of π^(k), V^(k) with distillation of
     π^(k+1), V^(k+1)
8  while not converged do
9  |  Initialise student policy π^(k+1) and value function V^(k+1)
10 |  for t_distill steps do
11 |  |  α_V, α_π ← linear annealing to 0 over t_distill steps
12 |  |  B ← collect trajectory data using π^(k)
13 |  |  Update π^(k) and V^(k) with standard RL method using B
14 |  |  Update π^(k+1) and V^(k+1) with Eq. (6.2) using B, α_V, α_π, π^(k) and
   |  |    V^(k)
15 |  // Housekeeping
16 |  Discard π^(k) and V^(k)
17 |  Set k ← k + 1
```

D.2 Supervised Learning

Table D.1: Numerical values of results presented in Fig. 6.1. The 'Rel' column shows the error normalised by the error of the unmodified dataset. The error on the test-data deteriorates worse than on the training data, not only in absolute, but also relativ terms.

	Training Error in %	Rel.	Testing Error in %	Rel.
Unmodified	0.17 ± 0.09	1.0	14.8 ± 0.70	1.0
Noisy Labels	0.19 ± 0.09	1.13	16.8 ± 0.70	1.14
Wrong Labels	0.20 ± 0.08	1.22	19.2 ± 0.43	1.30
Dataset Size	0.18 ± 0.08	1.05	23.4 ± 0.83	1.58

Table D.2: Hyper-parameters used in the supervised learning experiment on CIFAR-10

Hyper-parameter	Value
SGD: Learning rate	3×10^{-4}
SGD: Momentum	0.9
SGD: Weight decay	5×10^{-4}

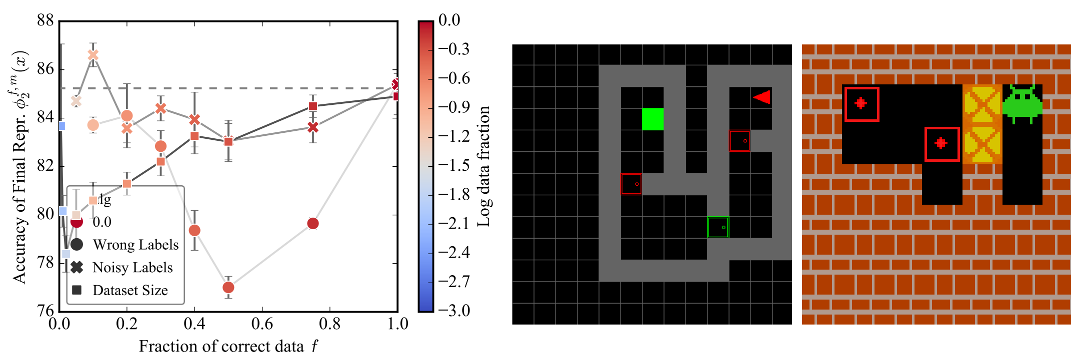


Figure D.1: *Left:* Same results as in Fig. 6.5 (middle), but with the fraction of correct data points f on the x-Axis. *Middle:* Multiroom example layout. The red agent needs to reach the green square, avoiding walls (grey) and passing through doors (blocks with coloured outline). *Right:* Sokoban example layout. The green agent needs to push (or pull) yellow boxes on the red targets, avoiding walls.

Here we provide additional training details and results for the supervised learning experiments performed on the CIFAR-10 dataset. We used a ResNet18 architecture without Batchnorm, hyper-parameters for the SGD optimiser are given in Table D.2.

In Table D.1 we provide exact numerical values for the results in Fig. 6.1. We also provide values for the *relative* change in error rate due to the introduction of non-stationarities, for which the test-performance is also more affected than the train performance.

In Fig. D.1, we show the same results as in Fig. 6.5, but here showing the f values used to generate $\mathcal{D}_{f,m}$ on the x-Axis. The same 'dips' in performance are visible, however from this figure it is clear that `Dataset Size` experiences it for much smaller values of f, which is unsurprising, giving the missing influences of a diverse input-data distribution.

Lastly, in Fig. D.2, we provide the individual training runs used to generate Fig. 6.5(middle) and Fig. D.1.

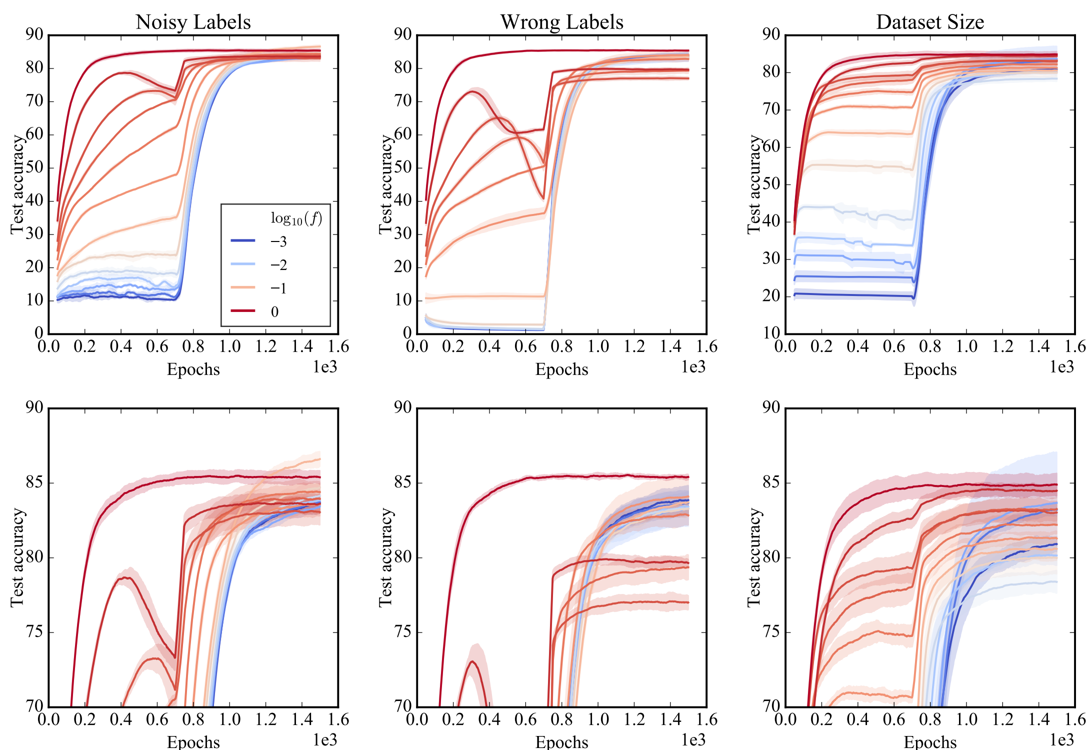


Figure D.2: Individual training curves for the data used in Fig. 6.5(middle) and Fig. D.1. The bottom row shows the same data as the top row, just 'zoomed in'.

D.2.1 Multiroom

In Table D.3 we show the hyper-parameters used for the Multiroom experiments which are shared between 'PPO' and 'PPO+ITER'. We note that our Multiroom en-

vironment uses the same modification that was used in [28] to make the environment fully observable. In the original environment, the agent only observed its immediate surrounding from an ego-centric perspective, thereby naturally generalising across various layouts. Instead, full observability introduces the need to *learn* how to generalise. Our network consists of a three layer CNN With 16, 32 and 32 filters respectively, followed by a fully connected layer of size 64. One max-pooling layer is used after the first CNN layer. We use $t_{init} = 4 \times 10^7$ and $t_{distill} = 4 \times 10^7$ (see algorithm 5) for the duration of the initial RL training phase and the following distillation phases.

Table D.3: Hyper-parameters used for Multiroom

Hyper-parameter	Value
PPO: $\lambda_{\text{Entropy Loss}}$	0.01
PPO: λ_{TD}	0.5
PPO: ϵ_{Clip}	0.2
PPO Epochs	4
PPO Minibatch Size	2048
Parallel Environments	32
Frames per Env per Update	256
λ_{GAE}	0.95
γ	0.99
Adam: Learning rate	7×10^{-4}
Adam: ϵ	1×10^{-5}

D.2.2 Sokoban

For Sokoban, we re-use the same hyper-parameters as for Multiroom, but with a reduced learning rate ($1.0e - 04$) and doubled minibatch and batch size in order to stabilise training. We re-use the same architecture but provide all CNN layers with 64 filters and remove the max-pooling layer due to the smaller grid-size of Sokoban.

D.2.3 ProcGen

In Fig. D.3 we show all results on the various *ProcGen* environment from which the summary plots in the main text (Figs. 6.3 and 6.4) are computed. We use

Table D.4: Hyper-parameters used for ProcGen

Hyperparameter	Value
PPO: $\lambda_{\text{Entropy Loss}}$	0.01
PPO: λ_{TD}	0.5
PPO: ϵ_{Clip}	0.2
PPO Epochs	3
PPO Nr. Minibatches	8
Parallel Environments	64
Frames per Env per Update	256
λ_{GAE}	0.95
γ	0.999
Adam: Learning rate	5×10^{-4}
Adam: ϵ	1×10^{-5}
Adam: Weight decay	1×10^{-4}

the same (small) IMPALA architecture as used by [145]. Training is done on 4 GPUs in parallel. One GPU is continuously evaluating the test performance, the other three are used for training. Their gradients are averaged at each update step. The hyper-parameters given in Table D.4 are *per GPU*. The x-Axis in Fig. D.3 shows the total number of consumed frames, i.e. 250×10^6 per *training* GPU. The distillation phase takes $t_{distill} = 70 \times 10^6$ frames (again per GPU) and we linearly anneal α_π from 1 to 0 and α_V from 0.5 to 1. The values of α_π and α_V were chosen to reflect the relative weight between $\mathcal{L}_{PG}$ and $\mathcal{L}_{TD}$ in Eq. (6.2) and no further tuning was done. The initial RL training phase takes $t_{init} = 50 \times 10^6$ frames. The distillation length was chosen based on preliminary experiments on *BigFish* by increasing its length in steps of 10×10^6 frames until no drop in training performance was experienced when switching to a new student.

Due to the high computation costs of running experiments on the *ProcGen* environment (4 GPUs for about 24h for each run), we decided to exclude environments from the original benchmark based on results presented by Cobbe et al. [132], figures 2 and 4. We excluded environments for two different reasons, either because the generalisation gap was small (Chaser, Miner, Leaper, Boss Fight, Fruitbot) or because generalisation did not improve at all during training after a very short

initial jump (CaveFlyer, Maze, Heist, Plunder, Coinrun), indicating that either it was too hard, or a very simple policy already achieved reasonable performance.

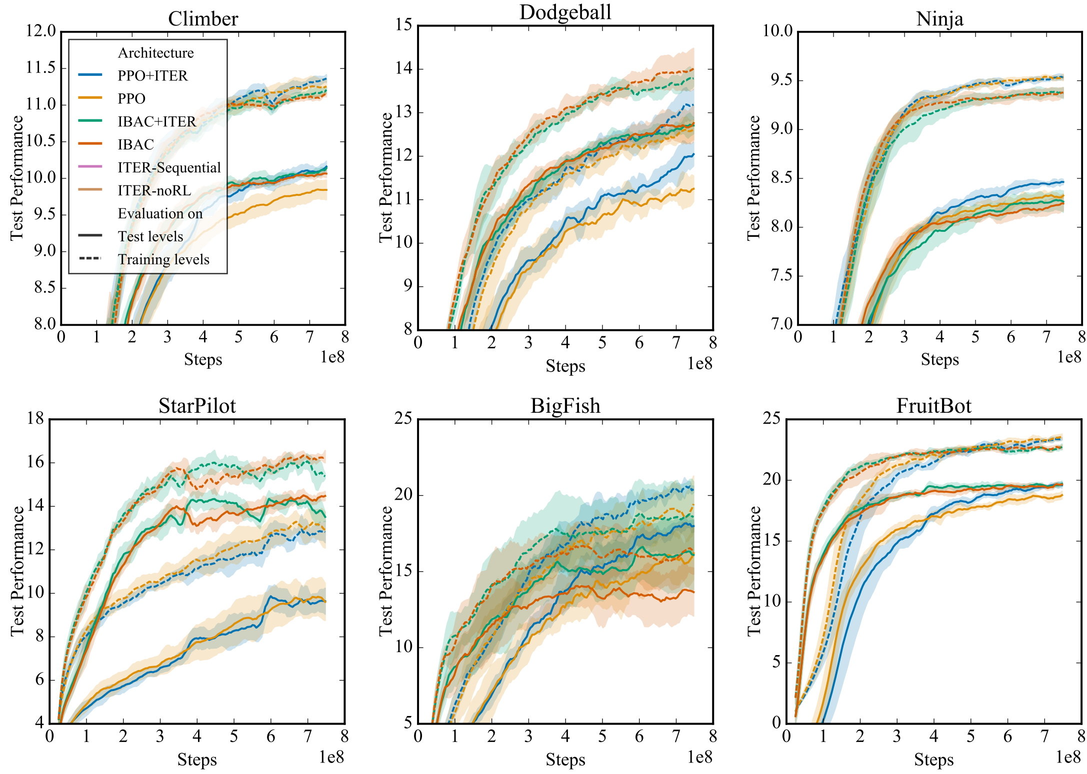


Figure D.3: All individual results on *ProcGen*. Shown is the mean and standard deviation across two random seeds.

References

[1] Stefano Nolfi and Dario Floreano. "Learning and evolution". In: *Autonomous robots* 7.1 (1999), pp. 89–113.

[2] Monica Gagliano et al. "Learning by association in plants". In: *Scientific reports* 6 (2016), p. 38427.

[3] Robert J Sternberg. *Intelligence*. John Wiley & Sons Inc, 2013.

[4] Elena L Grigorenko and Robert J Sternberg. "Dynamic testing." In: *Psychological Bulletin* 124.1 (1998), p. 75.

[5] Arthur Robert Jensen. *The g factor: The science of mental ability*. Vol. 648. Praeger Westport, CT, 1998.

[6] José Hernández-Orallo. *The measure of all minds: evaluating natural and artificial intelligence*. Cambridge University Press, 2017.

[7] William Stern. *The psychological methods of testing intelligence*. 13. Warwick & York, 1914.

[8] A. M. Turing. "Computing Machinery And Intelligence". In: *Mind* LIX.236 (Oct. 1950), pp. 433–460.

[9] Stuart Russell and Peter Norvig. "Artificial intelligence: a modern approach". In: (2002).

[10] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[11] Donald Michie and Roger A Chambers. "BOXES: An experiment in adaptive control". In: *Machine intelligence* 2.2 (1968), pp. 137–152.

[12] Donald E Knuth. *The art of computer programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998.

[13] Larry Tesler. *Adages and Coinages*. URL: http://www.nomodes.com/Larry_Tesler_Consulting/Adages_and_Coinages.html.

[14] Peter Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[15] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[16] Jared Kaplan et al. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020).

[17] Tom Henighan et al. "Scaling Laws for Autoregressive Generative Modeling". In: *arXiv preprint arXiv:2010.14701* (2020).

[18] Sam McCandlish et al. “An empirical model of large-batch training”. In: *arXiv preprint arXiv:1812.06162* (2018).

[19] John Maynard Keynes. *A tract on monetary reform*. London, Macmillan, 1923.

[20] Gerd Gigerenzer and Daniel G Goldstein. “Reasoning the fast and frugal way: models of bounded rationality.” In: *Psychological review* 103.4 (1996), p. 650.

[21] Chiyuan Zhang et al. “Understanding deep learning requires rethinking generalization”. In: *5th International Conference on Learning Representations, ICLR 2017*.

[22] Tom B Brown et al. “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165* (2020).

[23] Oriol Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. 2019.

[24] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (2019), pp. 350–354.

[25] Christopher Berner et al. “Dota 2 with large scale deep reinforcement learning”. In: *arXiv preprint arXiv:1912.06680* (2019).

[26] Maximilian Igl et al. “Deep Variational Reinforcement Learning for POMDPs”. In: *International Conference on Machine Learning*. 2018, pp. 2117–2126.

[27] Maximilian Igl et al. “Multitask soft option learning”. In: *Conference on Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 969–978.

[28] Maximilian Igl et al. “Generalization in reinforcement learning with selective noise injection and information bottleneck”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 13956–13968.

[29] Maximilian Igl et al. “Transient Non-stationarity and Generalisation in Deep Reinforcement Learning”. In: *International Conference on Learning Representations (ICLR)* (2021).

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[31] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[32] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[33] Richard HR Hahnloser et al. “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. In: *Nature* 405.6789 (2000), pp. 947–951.

[34] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[35] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015*. 2015.

[36] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning Representations by Back-Propagating Errors”. In: *Nature* 323.6088 (Oct. 9, 1986), pp. 533–536.

[37] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014*. 2014.

[38] John Schulman et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *4th International Conference on Learning Representations, ICLR 2016*.

[39] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[40] John Schulman et al. “Trust region policy optimization”. In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[41] Ashish Vaswani et al. “Attention Is All You Need”. In: (June 12, 2017).

[42] Pierre Del Moral. “Nonlinear filtering: Interacting particle resolution”. In: *Comptes Rendus de l’Académie des Sciences-Series I-Mathematics* 325.6 (1997), pp. 653–658.

[43] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *International Conference on Machine Learning*. 2016.

[44] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[45] Karl J Astrom. “Optimal control of Markov decision processes with incomplete state estimation”. In: *Journal of mathematical analysis and applications* 10 (1965), pp. 174–205.

[46] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1 (1998).

[47] R Andrew McCallum. “Overcoming incomplete perception with utile distinction memory”. In: *Proceedings of the Tenth International Conference on Machine Learning*. 1993, pp. 190–196.

[48] Matthew Hausknecht and Peter Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. In: *2015 AAAI Fall Symposium Series*. 2015.

[49] Pengfei Zhu, Xin Li, and Pascal Poupart. “On Improving Deep Reinforcement Learning for POMDPs”. In: *arXiv preprint 1704.07978* (2017).

[50] Tuan Anh Le et al. “Auto-Encoding Sequential Monte Carlo”. In: *ICLR*. 2018.

[51] Chris J Maddison et al. “Filtering Variational Objectives”. In: *Advances in Neural Information Processing Systems*. 2017.

[52] Christian A Naesseth et al. “Variational Sequential Monte Carlo”. In: *AISTATS (To Appear)*. 2018.

[53] Arnaud Doucet and Adam M Johansen. “A tutorial on particle filtering and smoothing: Fifteen years later”. In: *Handbook of nonlinear filtering* 12.656-704 (2009), p. 3.

[54] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1928–1937.

[55] Yuhuai Wu et al. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: *Advances in neural information processing systems*. 2017, pp. 5285–5294.

[56] Jan Buys, YonatanJan Bisk, and Yejin Choi. "Bridging HMMs and RNNs through Architectural Transformations". In: *NeurIPS 2018 Workshop on Interpretability and Robustness for Audio Speech and Language Workshop* (2018).

[57] Jakob N Foerster et al. "Learning to communicate to solve riddles with deep distributed recurrent q-networks". In: *arXiv preprint 1602.02672* (2016).

[58] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. "Language Understanding for Text-based Games using Deep Reinforcement Learning". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 1–11.

[59] Sebastian Thrun. "Monte carlo pomdps". In: *Advances in neural information processing systems*. 2000, pp. 1064–1070.

[60] David Silver and Joel Veness. "Monte-Carlo planning in large POMDPs". In: *Advances in neural information processing systems*. 2010, pp. 2164–2172.

[61] Junyoung Chung et al. "A recurrent latent variable model for sequential data". In: *Advances in neural information processing systems*. 2015.

[62] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. "Learning policies for partially observable environments: Scaling up". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 362–370.

[63] Manzil Zaheer et al. "Deep Sets". In: (Mar. 10, 2017).

[64] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning structured output representation using deep conditional generative models". In: *Advances in Neural Information Processing Systems*. 2015, pp. 3483–3491.

[65] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance Weighted Autoencoders". In: *ICLR*. 2016.

[66] Randal Douc and Olivier Cappé. "Comparison of resampling schemes for particle filtering". In: *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.* IEEE. 2005, pp. 64–69.

[67] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. "Learning to act using real-time dynamic programming". In: *Artificial intelligence* 72.1-2 (1995), pp. 81–138.

[68] David A McAllester and Satinder Singh. "Approximate planning for factored POMDPs using belief state simplification". In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. 1999.

[69] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. "Point-based value iteration: An anytime algorithm for POMDPs". In: *IJCAI*. Vol. 3. 2003.

[70] Stéphane Ross et al. "Online planning algorithms for POMDPs". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 663–704.

[71] Frans A Oliehoek et al. “Exploiting locality of interaction in factored Dec-POMDPs”. In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*. 2008.

[72] Diederik Marijn Roijers, Shimon Whiteson, and Frans A Oliehoek. “Point-Based Planning for Multi-Objective POMDPs.” In: *IJCAI*. 2015, pp. 1666–1672.

[73] Bram Bakker. “Reinforcement learning with long short-term memory”. In: *Advances in neural information processing systems*. 2002, pp. 1475–1482.

[74] Daan Wierstra et al. “Solving deep memory POMDPs with recurrent policy gradients”. In: *International Conference on Artificial Neural Networks*. Springer. 2007, pp. 697–706.

[75] Marvin Zhang et al. “Policy learning with continuous memory states for partially observed robotic control”. In: *CoRR* (2015).

[76] Nicolas Heess et al. “Memory-based control with recurrent neural networks”. In: *arXiv preprint 1512.04455* (2015).

[77] Emilio Parisotto et al. “Stabilizing transformers for reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7487–7498.

[78] Peter Karkus, David Hsu, and Wee Sun Lee. “Qmdp-net: Deep learning for planning under partial observability”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4697–4707.

[79] Aviv Tamar et al. “Value iteration networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2154–2162.

[80] Pierre-Arnaud Coquelin, Romain Deguest, and Rémi Munos. “Particle filter-based policy gradient in POMDPs”. In: *NIPS*. 2009.

[81] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. “Reinforcement Learning of POMDPs using Spectral Methods”. In: *Proceedings of the 29th Annual Conference on Learning Theory (COLT2016)*. 2016.

[82] Stéphane Ross et al. “A Bayesian approach for learning and planning in partially observable Markov decision processes”. In: *Journal of Machine Learning Research* (2011).

[83] Sammie Katt, Frans A Oliehoek, and Christopher Amato. “Learning in POMDPs with Monte Carlo Tree Search”. In: *International Conference on Machine Learning*. 2017.

[84] Finale Doshi-Velez et al. “Bayesian nonparametric methods for partially-observable reinforcement learning”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2015), pp. 394–407.

[85] MP Deisenroth and J Peters. “Solving Nonlinear Continuous State-Action-Observation POMDPs for Mechanical Systems with Gaussian Noise”. In: *The 10th European Workshop on Reinforcement Learning (EWRL 2012)*. 2012.

[86] Frans MJ Willems, Yuri M Shtarkov, and Tjalling J Tjalkens. “The context-tree weighting method: basic properties”. In: *IEEE Transactions on Information Theory* 41.3 (1995), pp. 653–664.

[87] Guillaume Lample and Devendra Singh Chaplot. “Playing FPS Games with Deep Reinforcement Learning.” In: *AAAI*. 2017, pp. 2140–2146.

[88] Max Jaderberg et al. "Reinforcement learning with unsupervised auxiliary tasks". In: *ICLR* (2017).

[89] Michael L Littman, Richard S Sutton, and Satinder P Singh. "Predictive representations of state." In: *NIPS*. Vol. 14. 1555. 2001, p. 30.

[90] Satinder Singh, Michael James, and Matthew Rudary. "Predictive state representations: A new theory for modeling dynamical systems". In: *arXiv preprint arXiv:1207.4167* (2012).

[91] Wen Sun et al. "Learning to filter with predictive state inference machines". In: *International conference on machine learning*. PMLR. 2016, pp. 1197–1205.

[92] Matthew Schlegel et al. "General value function networks". In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 497–543.

[93] Benedicte M Babayan, Naoshige Uchida, and Samuel J Gershman. "Belief state representation in the dopamine system". In: *Nature communications* 9.1 (2018), p. 1891.

[94] Xiao Ma et al. "Discriminative particle filter reinforcement learning for complex partial observations". In: *arXiv preprint arXiv:2002.09884* (2020).

[95] Yunbo Wang et al. "DualSMC: Tunneling differentiable filtering and planning under continuous POMDPs". In: *arXiv: 1909.13003* (2019).

[96] Kyunghyun Cho et al. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In: *Syntax, Semantics and Structure in Statistical Translation* (2014), p. 103.

[97] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[98] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. 1999.

[99] Yoshua Bengio et al. "Curriculum learning". In: ACM. 2009.

[100] Jane X. Wang et al. "Learning to reinforcement learn". In: *CoRR* abs/1611.05763 (2016). arXiv: `1611.05763`.

[101] Yee Teh et al. "Distral: Robust multitask reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4496–4506.

[102] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 1-2 (1999).

[103] Matthew M Botvinick, Yael Niv, and Andrew C Barto. "Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective". In: *Cognition* 3 (2009).

[104] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The Option-Critic Architecture." In: *AAAI*. 2017.

[105] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. "Variational intrinsic control". In: *ICLR* (2017).

[106] Alexander Sasha Vezhnevets et al. "FeUdal Networks for Hierarchical Reinforcement Learning". In: *ICML*. 2017.

[107] Ofir Nachum et al. "Data-efficient hierarchical reinforcement learning". In: *Advances in neural information processing systems*. 2018, pp. 3303–3313.

[108] Kevin Frans et al. "Meta Learning Shared Hierarchies". In: *ICLR*. 2018.

[109] Sergey Levine. "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: *arXiv:1805.00909* (2018).

[110] Emanuel Todorov. "General duality between optimal control and estimation". In: IEEE. 2008.

[111] Jean Harb et al. "When Waiting Is Not an Option: Learning Options With a Deliberation Cost". In: *AAAI*. 2018.

[112] Amy McGovern and Andrew G. Barto. "Automatic Discovery of Subgoals in Reinforcement Learning Using Diverse Density". In: *ICML*. 2001.

[113] Chen Tessler et al. "A deep hierarchical approach to lifelong learning in minecraft". In: *AAAI*. 2017.

[114] Anna Harutyunyan et al. "The Termination Critic". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. 2019, pp. 2231–2240.

[115] Karol Hausman et al. "Learning an Embedding Space for Transferable Robot Skills". In: *International Conference on Learning Representations*. 2018.

[116] Anirudh Goyal et al. "Transfer and Exploration via the Information Bottleneck". In: *ICLR*. 2019.

[117] Christian Daniel, Gerhard Neumann, and Jan Peters. "Hierarchical relative entropy policy search". In: *Artificial Intelligence and Statistics*. 2012.

[118] Christian Daniel et al. "Probabilistic inference for determining options in reinforcement learning". In: *Machine Learning* (2016).

[119] Nicolas Heess et al. "Learning and transfer of modulated locomotor controllers". In: *arXiv preprint arXiv:1610.05182* (2016).

[120] Tuomas Haarnoja et al. "Latent Space Policies for Hierarchical Reinforcement Learning". In: *International Conference on Machine Learning*. 2018, pp. 1851–1860.

[121] Sebastian Thrun and Anton Schwartz. "Finding structure in reinforcement learning". In: *NeurIPS*. 1995.

[122] Jacob Andreas, Dan Klein, and Sergey Levine. "Modular Multitask Reinforcement Learning with Policy Sketches". In: *ICML*. 2017.

[123] Dhruva Tirumala et al. "Exploiting Hierarchy for Learning and Transfer in KL-regularized RL". In: *arXiv:1903.07438* (2019).

[124] Roy Fox, Michal Moshkovitz, and Naftali Tishby. "Principled option learning in Markov decision processes". In: *arXiv:1609.05524* (2016).

[125] Kate Rakelly et al. "Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.

[126] Luisa Zintgraf et al. "VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning". In: *International Conference on Learning Representations*. 2019.

[127] Luisa Zintgraf et al. "Exploration in Approximate Hyper-State Space for Meta Reinforcement Learning". In: *arXiv preprint arXiv:2010.01062* (2020).

[128] Thomas G Dietterich. "The MAXQ Method for Hierarchical Reinforcement Learning." In: *ICML*. 1998.

[129] Alexandre Galashov et al. "Information asymmetry in KL-regularized RL". In: *ICLR*. 2019.

[130] Nicholas K Jong, Todd Hester, and Peter Stone. "The utility of temporal abstraction in reinforcement learning". In: *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems. 2008.

[131] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *IROS*. IEEE. 2012.

[132] Karl Cobbe et al. "Leveraging procedural generation to benchmark reinforcement learning". In: *International conference on machine learning*. PMLR. 2020, pp. 2048–2056.

[133] Maxime Chevalier-Boisvert and Lucas Willems. *Minimalistic Gridworld Environment for OpenAI Gym*. `https://github.com/maximecb/gym-minigrid`. 2018.

[134] Alexander A. Alemi et al. "Deep Variational Information Bottleneck". In: *5th International Conference on Learning Representations, ICLR 2017*.

[135] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[136] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[137] Xue Bin Peng et al. "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[138] OpenAI. *OpenAI Five*. `https://blog.openai.com/openai-five/`. 2018.

[139] Peter Henderson et al. "Deep reinforcement learning that matters". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[140] Shimon Whiteson et al. "Protecting against evaluation overfitting in empirical reinforcement learning". In: *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2011, pp. 120–127.

[141] Amy Zhang, Nicolas Ballas, and Joelle Pineau. "A dissection of overfitting and generalization in continuous reinforcement learning". In: *arXiv preprint arXiv:1806.07937* (2018).

[142] Chiyuan Zhang et al. "A study on overfitting in deep reinforcement learning". In: *arXiv preprint arXiv:1804.06893* (2018).

[143] Chenyang Zhao et al. "Investigating Generalisation in Continuous Deep Reinforcement Learning". In: *arXiv preprint arXiv:1902.07015* (2019).

[144] Devendra Singh Chaplot et al. “Transfer deep reinforcement learning in 3d environments: An empirical study”. In: *NIPS Deep Reinforcemente Leaning Workshop*. 2016.

[145] Karl Cobbe et al. “Quantifying generalization in reinforcement learning”. In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.

[146] Arthur Juliani et al. “Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning”. In: *CoRR* abs/1902.01378 (2019). arXiv: 1902.01378. URL: http://arxiv.org/abs/1902.01378.

[147] Niels Justesen et al. “Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation”. In: *arXiv preprint arXiv:1806.10729* (2018).

[148] Alex Nichol et al. “Gotta learn fast: A new benchmark for generalization in rl”. In: *arXiv preprint arXiv:1804.03720* (2018).

[149] Amy Zhang, Yuxin Wu, and Joelle Pineau. “Natural Environment Benchmarks for Reinforcement Learning”. In: *arXiv preprint arXiv:1811.06032* (2018).

[150] Mikhail Belkin et al. “Reconciling modern machine learning and the bias-variance trade-off”. In: *arXiv preprint arXiv:1812.11118* (2018).

[151] Ohad Shamir, Sivan Sabato, and Naftali Tishby. “Learning and generalization with the information bottleneck”. In: *Theoretical Computer Science* 411.29-30 (2010), pp. 2696–2711.

[152] Naftali Tishby and Noga Zaslavsky. “Deep learning and the information bottleneck principle”. In: *2015 IEEE Information Theory Workshop (ITW)*. IEEE. 2015, pp. 1–5.

[153] Terrance DeVries and Graham W Taylor. “Improved regularization of convolutional neural networks with cutout”. In: *arXiv preprint arXiv:1708.04552* (2017).

[154] Elad Hoffer, Itay Hubara, and Daniel Soudry. “Train longer, generalize better: closing the generalization gap in large batch training of neural networks”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 1731–1741.

[155] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[156] Ping Luo et al. “Towards Understanding Regularization in Batch Normalization”. In: *International Conference on Learning Representations*. 2019.

[157] Naftali Tishby, Fernando C Pereira, and William Bialek. “The information bottleneck method”. In: *arXiv preprint physics/0004057* (2000).

[158] Twan van Laarhoven. “L2 Regularization versus Batch and Weight Normalization”. In: *arXiv preprint arXiv:1706.05350* (2017).

[159] Sergey Levine et al. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”. In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.

[160] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. “The transferability approach: Crossing the reality gap in evolutionary robotics”. In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 122–145.

[161] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 23–30.

[162] Freek Stulp et al. “Learning to grasp under uncertainty”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 5703–5708.

[163] Rika Antonova et al. “Reinforcement learning for pivoting task”. In: *arXiv preprint arXiv:1703.00472* (2017).

[164] Charles Packer et al. “Assessing Generalization in Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1810.12282* (2018).

[165] Wenhao Yu et al. “Preparing for the Unknown: Learning a Universal Policy with Online System Identification”. In: *Robotics: Science and Systems XIII, Massachusetts Institute of Technology*. 2017.

[166] Ajay Mandlekar et al. “Adversarially robust policy learning: Active construction of physically-plausible perturbations”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3932–3939.

[167] Lerrel Pinto et al. “Robust adversarial reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2817–2826.

[168] Aravind Rajeswaran et al. “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles”. In: *5th International Conference on Learning Representations, ICLR 2017*.

[169] Aravind Rajeswaran et al. “Towards generalization and simplicity in continuous control”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6550–6561.

[170] Pascal Poupart et al. “An analytic solution to discrete Bayesian reinforcement learning”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 697–704.

[171] Maruan Al-Shedivat et al. “Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments”. In: *International Conference on Learning Representations*. 2018.

[172] Ignasi Clavera et al. “Learning to Adapt: Meta-Learning for Model-Based Control”. In: *CoRR* abs/1803.11347 (2018).

[173] Yan Duan et al. “RL2: Fast Reinforcement Learning via Slow Reinforcement Learning”. In: *arXiv preprint arXiv:1611.02779* (2016).

[174] Tom Schaul et al. “Universal value function approximators”. In: *International conference on machine learning*. 2015, pp. 1312–1320.

[175] Flood Sung et al. “Learning to learn: Meta-critic networks for sample efficient learning”. In: *arXiv preprint arXiv:1706.09529* (2017).

[176] Edward Beeching et al. “Deep Reinforcement Learning on a Budget: 3D Control and Reasoning Without a Supercomputer”. In: *CoRR* abs/1904.01806 (2019). arXiv: 1904.01806.

[177] Matthew Johnson et al. “The Malmo Platform for Artificial Intelligence Experimentation.” In: *IJCAI*. 2016, pp. 4246–4247.

[178] Yuji Kanagawa and Tomoyuki Kaneko. “Rogue-Gym: A New Challenge for Generalization in Reinforcement Learning”. In: *arXiv preprint arXiv:1904.08129* (2019).

[179] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. “ViZDoom competitions: playing doom from pixels”. In: *IEEE Transactions on Games* (2018).

[180] Fereshteh Sadeghi and Sergey Levine. “CAD2RL: Real Single-Image Flight Without a Single Real Image”. In: *Robotics: Science and Systems XIII, Massachusetts Institute of Technology*. 2017.

[181] Ken Kansky et al. “Schema networks: Zero-shot transfer with a generative causal model of intuitive physics”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1809–1818.

[182] Mario Srouji, Jian Zhang, and Ruslan Salakhutdinov. “Structured Control Nets for Deep Reinforcement Learning”. In: *International Conference on Machine Learning*. 2018, pp. 4749–4758.

[183] Vinicius Zambaldi et al. “Deep reinforcement learning with relational inductive biases”. In: *International Conference on Learning Representations*. 2019.

[184] Alon Brutzkus and Amir Globerson. “Over-parameterization Improves Generalization in the XOR Detection Problem”. In: *CoRR* abs/1810.03037 (2018). arXiv: 1810.03037.

[185] Anirudh Goyal et al. “InfoBot: Transfer and Exploration via the Information Bottleneck”. In: *International Conference on Learning Representations*. 2019.

[186] Ronald Kemker et al. “Measuring catastrophic forgetting in neural networks”. In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[187] William Fedus et al. “Revisiting fundamentals of experience replay”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3061–3071.

[188] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[189] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[190] Lasse Espeholt et al. “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures”. In: *arXiv preprint arXiv:1802.01561* (2018).

[191] Max-Philipp B. Schrader. *gym-sokoban*. https://github.com/mpSchrader/gym-sokoban. 2018.

[192] Nikolaus Kriegeskorte and Rogier A Kievit. “Representational geometry: integrating cognition, computation, and the brain”. In: *Trends in cognitive sciences* 17.8 (2013), pp. 401–412.

[193] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. “Distilling the Knowledge in a Neural Network”. In: *NIPS Deep Learning and Representation Learning Workshop*. 2015.

[194] Tommaso Furlanello et al. “Born Again Neural Networks”. In: *International Conference on Machine Learning*. 2018, pp. 1607–1616.

[195] Hossein Mobahi, Mehrdad Farajtabar, and Peter L Bartlett. “Self-distillation amplifies regularization in hilbert space”. In: *arXiv preprint arXiv:2002.05715* (2020).

[196] Akhilesh Gotmare et al. “A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation”. In: *International Conference on Learning Representations*. 2019.

[197] Wojciech M Czarnecki et al. “Distilling Policy Distillation”. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. 2019, pp. 1331–1340.

[198] Andrei A Rusu et al. “Policy distillation”. In: *arXiv preprint arXiv:1511.06295* (2015).

[199] Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. “Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning”. In: *5th International Conference on Learning Representations, ICLR 2016*. 2016.

[200] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[201] Simon Schmitt et al. “Kickstarting deep reinforcement learning”. In: *arXiv preprint arXiv:1803.03835* (2018).

[202] Dibya Ghosh et al. “Divide-and-Conquer Reinforcement Learning”. In: *International Conference on Learning Representations*. 2018.

[203] Wojciech Czarnecki et al. “Mix & Match Agent Curricula for Reinforcement Learning”. In: *International Conference on Machine Learning*. 2018, pp. 1087–1095.

[204] German I Parisi et al. “Continual lifelong learning with neural networks: A review”. In: *Neural Networks* (2019).

[205] Robert M French. “Catastrophic forgetting in connectionist networks”. In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[206] Zhizhong Li and Derek Hoiem. “Learning without forgetting”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.

[207] Jonathan Schwarz et al. “Progress & Compress: A scalable framework for continual learning”. In: *International Conference on Machine Learning*. 2018, pp. 4528–4537.

[208] Samuel PM Choi, Dit-Yan Yeung, and Nevin Lianwen Zhang. “An environment model for nonstationary reinforcement learning”. In: *Advances in neural information processing systems*. 2000, pp. 987–993.

[209] Bruno C Da Silva et al. “Dealing with non-stationary environments using context detection”. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 217–224.

[210] Kenji Doya et al. “Multiple model-based reinforcement learning”. In: *Neural computation* 14.6 (2002), pp. 1347–1369.

[211] Richard S Sutton and Andrew G Barto. “Reinforcement learning: An introduction”. In: (2018).

[212] Kimin Lee et al. “Network Randomization: A Simple Technique for Generalization in Deep Reinforcement Learning”. In: *International Conference on Learning Representations*. 2020.

[213] Zhuang Liu et al. “Regularization Matters in Policy Optimization”. In: *arXiv preprint arXiv:1910.09191* (2019).

[214] Heinrich Küttler et al. “The nethack learning environment”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[215] Jan Humplik et al. “Meta reinforcement learning as task inference”. In: *arXiv preprint arXiv:1905.06424* (2019).

[216] Sergey Levine et al. “Offline reinforcement learning: Tutorial, review, and perspectives on open problems”. In: *arXiv preprint arXiv:2005.01643* (2020).

[217] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.