Automated and Verified Deep Learning

Harkirat Singh Behl

St Cross College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2021

Acknowledgements

I would like to thank my supervisors, Pawan and Phil, for their support. I am indebted to their patience throughout my PhD. I would like to thank Phil for providing me the opportunity to pursue a PhD, for training me and for giving me the freedom to explore my areas of interest and to collaborate. His enthusiasm and energy are infectious. I want to express my deepest gratitude to Pawan. I want to thank him for teaching important research skills, encouraging me, and instilling confidence in me. Pawan's dedication and kindness will remain a source of inspiration.

I would like to thank my mentors, Michael, Gunes, Vibhav, and DJ - who have played a significant role in my development as a researcher. I have also enjoyed and learnt from my collaborations with Puneet, Anurag, Mohammad, Arnab, Alessandro, Amartya, Pau, Stuart, Saumya and Vinay. I would like to thank the members of TVG and OVAL groups for insightful discussions during the reading groups, help and support along the journey.

St Cross College provided a wonderful environment and a second home. I am thankful for the social events and lunches, for feeding me and through which I met some wonderful people. I would like to thank Arnab, Brian, Deepak, Erica, Gurpreet, Melisa, Shivangi, and Srinandini for being supportive and amazing friends.

Cricket has played an essential role in keeping me sane and recharging me. I would like to thank the different clubs, OUCCC, Oxenford, OIS, Wolfson College, their managing committees, and players for the wonderful time. There are too many people to name here, some with whom I had as much fun off the field as on the field.

I am deeply indebted to my parents Jagjit Kaur and Ajmer Singh Behl. I am also thankful to my sister and Jiju for entertaining me on my impromptu visits to London.

I would like to thank God for giving me this wonderful opportunity. I also want to thank God for sending the strength and the wonderful people, with the help of which this became possible.

Finally, I am grateful to Tencent for fully funding my PhD.

Abstract

In the last decade, deep learning has enabled remarkable progress in various fields such as image recognition, machine translation, and speech recognition. We are also witnessing an explosion in the range of applications. However, there are many challenges that stand in the way of the widespread deployment of deep learning. In this thesis, we focus on two of the key challenges, namely, neural network verification and automated machine learning.

Firstly, deep neural networks are infamous for being 'black boxes' and making unexpected mistakes. For reliable AI, we want systems that are consistent with specifications like fairness, unbiasedness and robustness. We focus on verifying the adversarial robustness of neural networks, which aims at proving the existence or non-existence of an adversarial example. This non-convex problem is commonly approximated with a convex relaxation. We make two important contributions in this direction. First, we propose a specialised dual solver for a new convex relaxation. This was essential because although the relaxation is tighter than previous relaxations, it has an exponential number of constraints which make the existing dual solvers inapplicable. Second, we design a tighter relaxation for the problem of verifying robustness to input perturbations within the probability simplex. The size of our relaxation is linear in the number of neurons, which enables us to design simpler and efficient algorithms. Empirically, we demonstrate the performance by verifying the respective specifications on common verification benchmarks.

Secondly, deep neural networks require extensive human effort and expertise. We consider automated machine learning or meta learning which aims at automating the process of applying machine learning. We make three contributions in this context. First, we propose efficient approximations for the bi-level formulation of meta learning. We show its efficiency in the context of learning to generate synthetic data for training neural networks by optimizing state-of-the-art photorealistic renderers. Second, we propose a technique to automatically optimize the learning rate of gradient-based meta learning algorithms. We demonstrate a substantial reduction in the need to tune training hyperparameters. Third, we show an application by tackling video segmentation as a meta learning problem and demonstrating state-of-the-art results on common benchmarks.

Contents

List of Figures

The final test of a theory is its capacity to solve the problems which originated it.

— George Dantzig

1 Introduction

Contents

1.1 Preamble

Deep neural networks have enabled remarkable progress on a wide range of problems, such as image recognition (Krizhevsky et al., 2012), machine translation (Luong et al., 2015) and speech recognition (Hinton et al., 2012). Alongside the improvement in performance on established academic tasks (Deng et al., 2009), we are also witnessing an explosion in the range of applications. Deep neural networks are now being used in many parts of daily life, like web-searches (Raghavan, 2020), social media (Naumov et al., 2019; Covington et al., 2016), spam filtering (Dada et al., 2019), virtual assistants (van den Oord et al., 2016) and many more.

However, many challenges stand in the way of widespread deployment of deep learning. One of the key challenges is that deep learning algorithms remain unreliable (Goodfellow et al., 2015; Carlini and Wagner, 2017; Biggio and Roli, 2018). Since neural networks are infamous for being 'black boxes' and make unexpected mistakes, they pose a substantial risk in safety critical applications. In applications like autonomous cars, aircraft collision avoidance, healthcare, and finance, a small mistake can cause loss of human life or millions of dollars. Thus, it is important to verify them, that is, give guarantees that they will behave in a certain way for a given set of inputs.

Another important challenge is that deep learning requires significant human effort and expertise. Instances of this include collecting and preprocessing data, designing appropriate models, post-processing predictions, and tuning hyperparameters. This limits its use to a smaller community of scientists and engineers; and such hand-designed components can make the overall system sub-optimal. Automated Machine learning (He et al., 2019), which is the process of automating the task of applying machine learning to real-world problems, is thus essential. AutoML, which includes meta-learning (Finn et al., 2017c), hyper-parameter optimization (Bergstra et al., 2011) and architecture search (Elsken et al., 2019), can help towards removing the requirement of machine learning expertise, making machine learning pipelines end-to-end, and producing models that often outperform hand-designed models.

In this thesis, we provide solutions for both automated machine learning and verification of neural networks. We begin by formulating both these problems using the framework of *optimization*. This allows us to build on the existing knowledge of optimization to provide solutions to the challenges encountered in practice. In the next sections, we provide a more detailed discussion of these two problems. Finally, we outline our contributions in the context of automated and verified deep learning.

1.2 Neural Network Verification

Current Artificial Intelligence (AI) systems are biased (Alvi et al., 2018), unfair (Mehrabi et al., 2021), and non-robust (Goodfellow et al., 2015). This is concerning as it brings into question the safety of these systems. For safe and reliable AI, we want systems that are consistent with specifications like fairness, unbiasedness and robustness to adversaries.

We focus our attention on the specification of robustness to adversaries. One way to check the robustness of neural networks is to use adversarial attacks (Goodfellow et al., 2015; Carlini and Wagner, 2017; Moosavi-Dezfooli et al., 2016), which aim at finding a counter-example to the specification. However, these attacks provide a false sense of security as they cannot provide a guarantee that no such counter-example exists. Thus, they are not sufficient for safety critical applications, where it is crucial to give provable guarantees about the consistency of the system with respect to the specification.

Neural network verification aims at providing guarantees that a neural network either satisfies or violates a particular specification. For the specification of adversarial robustness, this would mean to either: (i) prove that no counter-example exists (the specification is true); or (ii) identify a counter-example (the specification is false). As discussed above, adversarial attacks are typically used to identify counter-examples. In contrast, establishing the veracity of a specification requires solving verification. A visualization of the neural network verification problem is provided in Figures 1.1 and 1.2. We are interested in proving a property on an input domain by examining whether the final output satisfies the property. The

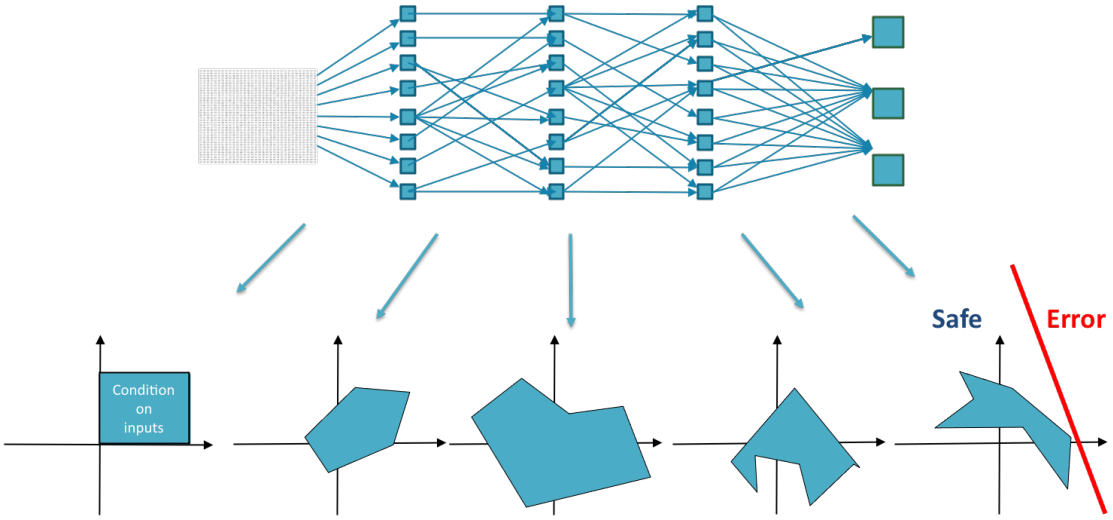


Figure 1.1: Visualization of neural network verification: We are interested in proving a property on an input domain by examining whether the final output satisfies the property.
Input: The dotted polygon represents the input domain and the blue connected part is a trained neural network. We have some conditions on the input to determine the domain. For example, the domain can be an ℓ_∞ norm ball around the input image to include all slightly perturbed images.
Network: We pass this domain into the neural network. The domain is transformed by each layer which is a linear transformation, followed by a ReLU operation. The ReLU operation truncates all negative values to zero. So after each layer, the image of the domain is a polyhedron. The non-linearity can make these polyhedrons non-convex, as shown in blue. This non-convexity makes the verification problem difficult to solve.
Property: The red line represents our property. The property is satisfied if the output area is to the left side of the line.

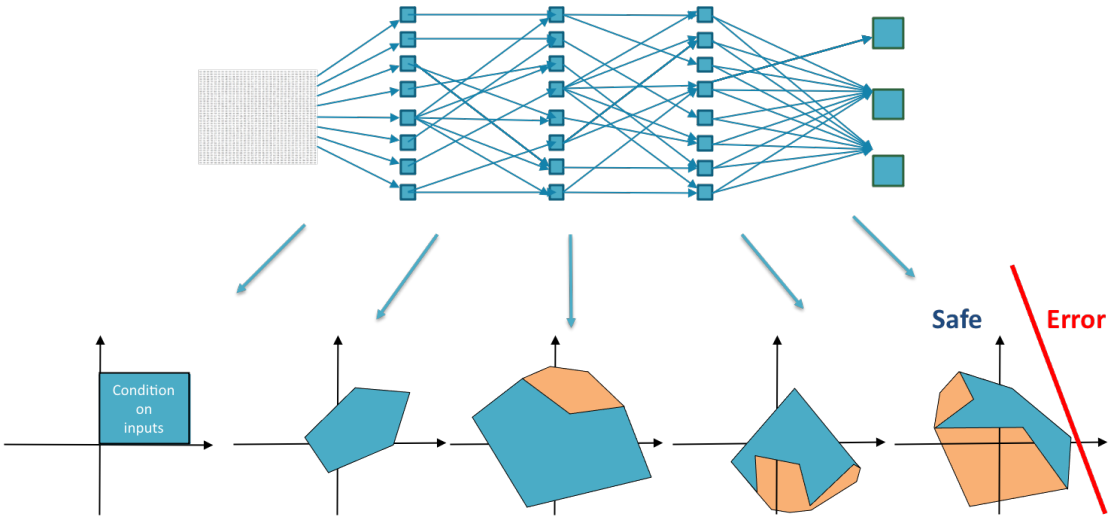


Figure 1.2: *Convex Relaxation:* The non-linear activation can be relaxed with a convex relaxation, such that the domain of each layer becomes convex as shown in orange. This convexity makes the problem tractable. The most commonly used relaxation for ReLU activation is the Planet relaxation (Ehlers, 2017).
Property: The property is satisfied only if the entire over-approximated output is to the left side. Thus only a subset of properties can be verified. In this example, the property is satisfied. The whole process is neural network verification.

input domain is transformed by each layer which is a linear transformation, followed by a ReLU operation. The non-linearity of the commonly used activation functions, for example ReLU, makes the problem non-convex. One possible way is to solve a suitable convex relaxation to obtain a lower bound on the minimum output.

1.3 Automated Machine Learning

Automating the process of applying machine learning to real-world problems is crucial for the widespread deployment of machine learning. AutoML is a broad area encompassing architecture search (Elsken et al., 2019), hyper-parameter optimization (Bergstra et al., 2011), meta-learning (Finn et al., 2017c), etc. In this work, we focus on automated machine learning and meta-learning from a computer vision perspective. Meta-learning—or "learning to learn"—concerns machine learning models that can improve their learning quality by altering aspects of the learning process such as the model architecture, optimization rules, initialization, or hyper-parameters (Thrun and Pratt, 2012; Schmidhuber, 1987a; Hochreiter et al., 2001a). Meta-learning methods are often classified into three categories, namely, gradient-based, non-parametric and model-based methods. We briefly discuss the first two since they are most relevant here.

Both automated machine learning and meta-learning can be formulated as a bi-level optimization problem, as shown in Figure 1.3. The inner problem is the neural network training and the outer problem optimizes aspects of the learning process, for example, the learning rate, architecture or activation function. In fact, all gradient-based methods can be seen as some approximation to the original bi-level problem. Different approximations are chosen based on the application and the aspect of the learning process that the outer problem is trying to optimize.

Metric learning methods are also effective for meta learning (Koch et al., 2015), where one learns a metric space in which learning is sample efficient. These works are inspired from the classical machine learning approaches like the nearest neighbors algorithm. Most popular amongst these are the prototypical networks

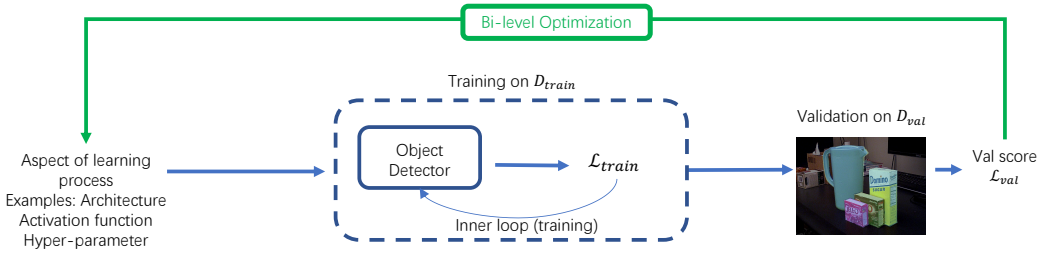


Figure 1.3: Bi-level formulation for AutoML: The inner problem is the neural network training and the outer problem optimizes aspects of the learning process, for example, the learning rate, architecture etc. The objective is to find the optimal hyper-parameter such that the model (Object Detector in this figure) trained with this hyper-parameter, achieves maximum accuracy on a downstream task represented by the validation set D_{val}.

(Snell et al., 2017b) and matching networks (Vinyals et al., 2016a). These methods are discussed in more detail in the thesis.

1.4 Thesis Outline & Contributions

Chapters 2 and 3 of the thesis investigate neural network verification. Chapters 4, 5 and 6 investigate automated machine learning.

In the remainder of this section, we discuss the contents and contributions of different chapters. Each chapter has a dedicated related work section discussing the relevant literature. This thesis is in an integrated format and is thus a collection of papers. Each of the chapters corresponds to a publication of which I am the first author.

One of my papers titled 'Incremental Tube construction for Human Action Detection' was published in BMVC 2018. It has however not been incorporated in this thesis.

1.4.1 Scaling the Convex Barrier with Active Sets

Corresponding publication. de Palma et al. (2021), published in ICLR 2021.

Summary. We present a specialised dual solver for a tight ReLU convex relaxation for neural network verification. We demonstrate that it speeds up formal

verification significantly.

General Contributions. Tighter and efficient neural network bounding is critical for scaling neural network verification. Recent work has shown the effectiveness of dual solvers for the verification problem. More recently, a tighter relaxation was proposed. However, this relaxation has an exponential number of constraints, thus the existing dual solvers are not easily applicable. We propose a novel dual solver for this relaxation. We also present a unified dual treatment that includes both the linearly sized LP relaxation (Ehlers, 2017) and the tighter relaxation (Anderson et al., 2019b). The solver obtains bounds better than off-the-shelf solvers in only a fraction of their running time. We demonstrate that the solver allows us to obtain significant speed-ups in formal verification.

Algorithmic Contributions. Designing an efficient solver for the Anderson relaxation is non-trivial because it has an exponential number of constraints. We propose an efficient solver for this relaxation. Our solver satisfies the following properties: (i) sparsity: linear in terms of memory and computation, (ii) achieves tight bounds like the primal solver, (iii) provides valid bounds at each step. These properties are crucial for a solver to be effective in the complete verification setting. The solver uses an active set of dual variables corresponding to primal constraints, and we propose an efficient criterion for selecting which constraints to add to the active set. The proposed algorithm enjoys massive parallelism within a GPU implementation. It also leverages the convolutional network structure with custom designed 'masked' forward and backward passes.

1.4.2 Overcoming the Convex Barrier for Simplex Inputs

Corresponding publication. Behl et al. (2021), published in NeurIPS 2021.

Summary. We design a tighter convex relaxation for the non-convex optimization problem of verifying robustness to input perturbations within the probability simplex. The size of our relaxation is linear in the number of neurons, which enables us to design simpler and efficient algorithms.

General Contributions. Tighter and efficient relaxations are one of the key components needed for scaling neural network verification. Recently, Tjandraatmadja et al. (2020) proposed a tight relaxation for verifying the robustness of a neural network to ℓ_∞ input perturbations. This relaxation has exponentially many (in the number of variables) constraints, and thus requires customised solvers as we proposed in de Palma et al. (2021). Taking inspiration from this work, we propose a tighter relaxation for verification with simplex inputs. We derive the convex hull for the composition of a linear function with a convex non-linearity such as ReLU or SoftPlus. Our relaxation only requires constraints linear in the dimensionality of the simplex. We demonstrate the scalability of our approach using the specification of robustness to ℓ_1 perturbations for MNIST and CIFAR-10 classification. We also demonstrate the benefits of the proposed relaxation for the challenging specification of global robustness in multi-modal classification.

Algorithmic Contributions. As described above, we derived a concise description for the convex hull for the composition of a linear function with a convex non-linearity, when the input lies in a simplex. We first propose an algorithm to propagate simplex constraints through the different layers of the network, given the simplex on the input. This allows us to use our convex hull on different layers of the network. For obtaining the lower bound for our overall relaxation, we also propose an efficient propagation based solver (Zhang et al., 2018; Singh et al., 2018). This algorithm expresses the lower and upper bounds as linear functions of the inputs of each neuron. The complexity of our solver for computing a valid lower bound is equivalent to the cost of two backward passes through the network.

Theoretical Contributions. We prove that the proposed relaxation represents the convex hull of the composition of a linear function of a vector in a simplex with the ReLU non-linearity. We also characterize the tightness gap between our proposed relaxation and the Planet relaxation (Ehlers, 2017), and show that our relaxation is tighter than both the Planet and Anderson relaxations (Tjandraatmadja et al., 2020).

1.4.3 AutoSimulate: Learning Synthetic Data Generation

Corresponding publication. Behl et al. (2020a), published at ECCV 2020.

Summary. We design an efficient bi-level optimization algorithm to meta-learn synthetic data generation for training neural networks. The objective is to optimize the renderer such that the data generated is optimal for training neural networks. The method can also optimize non-differentiable photo-realistic renderers.

General Contributions. The current pipeline for generating synthetic data for training neural networks involves manually handcrafting the simulator parameters, which might not be optimal and requires substantial human effort. While recent work has formulated the setting of simulator parameters as a bi-level problem, they treat the entire data generation and model training pipeline as a black-box, and use policy gradients, which require multiple expensive objective evaluations at each iteration. As a result, learning synthetic data generation with photo-realistic renderers has remained a challenge. Our key contribution lies in proposing a novel differentiable approximation of the objective, which allows us to optimize the simulator requiring only one objective evaluation at each iteration, with improved speed and accuracy. The proposed method can be used with non-differentiable simulators and handle very deep neural networks. We demonstrate the effectiveness of our method on two renderers, the Clevr data generator and the state-of-the-art photo-realistic renderer Arnold.

Algorithmic Contributions. We formulate the simulator optimization problem as a bi-level optimization problem of training on simulated data (inner) and validation on real data (outer). We propose a differentiable approximation for the inner objective which exploits the local convergence of the Newton method (Nocedal and Wright, 2006). This allows us to optimize the simulator requiring only one objective evaluation at each iteration. This is critical since the overall objective evaluation is quite expensive as it requires data generation from the simulator and training of the neural network on it. We also propose a technique to propagate batch gradients through non-differentiable computer vision simulators. These approximations allow us to propose a differentiable objective of the overall bi-level objective, which in turn allows us to optimize the simulator in an efficient end-to-end manner. Furthermore, we also propose efficient approximations for the second-order term in the gradient which offer different speed-accuracy trade-offs.

1.4.4 Meta-Learning Deep Visual Words for Fast Video Object Segmentation

Corresponding publication. Behl et al. (2020b), published at IROS 2020 and NeurIPS Workshop 2019.

Summary. We propose a fast and causal algorithm for video object segmentation. Unlike previous methods, we ensure that the training objective matches the inference procedure by utilizing meta-learning. It achieves state-of-the-art results on common video object segmentation datasets.

General Contributions. Personal robots and driverless cars need to work in novel environments, which requires them to learn new objects and segment them in a causal and efficient manner. Previous video segmentation methods take pretrained segmentation networks and fine-tune them on the labelled first frame, which is very time consuming. In this work, we propose a metric-learning based meta-learning approach, which learns to learn about the objects from the first frame, to segment them in the rest of the video. Our approach does not require fine-tuning and

uses no optical flow or extra post-processing, in order to develop a fast and causal algorithm. And it can segment a variable number of objects in a single forward pass. The proposed method performs at par with the offline fine-tuning based methods while being an order of magnitude faster. It also lies on the Pareto front for four of the most common video segmentation datasets, namely DAVIS-16, DAVIS-17, Youtube-Objects, and SegTrack-v2.

Algorithmic Contributions. We formulate the video segmentation problem as a meta-learning problem, where each task is to learn from the object labels in the first frame of the video (support set) to segment and track them in rest of the video (query set). Taking inspiration from metric-learning methods, we represent each object with a fixed number of cluster centroids in the embedding space (visual words). These visual words are learned without any explicit supervision by clustering our embedding space. We use meta-learning to ensure that our training objective matches our inference procedure. This contrasts with related metric-learning based approaches (Chen et al., 2018b; Najafi et al., 2018; Li et al., 2018) which are trained with surrogate, and sometimes unstable, losses. Our meta-training algorithm alternates between the unsupervised learning of deep visual words and supervised learning of pixel classification given these visual words.

1.4.5 Alpha MAML: Adaptive Model-Agnostic Meta-Learning

Corresponding publication. Behl et al. (2019), published at ICML AutoML Workshop 2019.

Summary. We propose an extension of the state-of-the-art model-agnostic meta-learning (MAML) algorithm (Finn et al., 2017b) by incorporating adaptive tuning of both the learning rate and meta-learning rate. Our results demonstrate a substantial improvement in MAML training stability, with reduced sensitivity to hyperparameters.

General Contributions. *Model-agnostic meta-learning* (MAML) (Finn et al., 2017b) is a conceptually simple and general algorithm that has been shown to outperform existing approaches in tasks including few-shot image classification and few-shot adaptation in reinforcement learning. However, the generality of MAML comes with the difficulty of choosing hyperparameters to achieve stable training in practice (Antoniou et al., 2019). In this work, we provide a conceptually simple solution to this problem, by introducing an extension of the MAML algorithm to incorporate adaptive tuning of both the learning rate and the meta-learning rate. This eliminates the need to tune meta-learning and learning rates. Our results demonstrate a substantial reduction in the need to tune MAML training hyperparameters and an improvement in training stability.

Algorithmic Contributions. Our aim is to make it possible to use MAML without or with significantly less parameter tuning, and to make the algorithm converge in fewer iterations. We derive online update rules for both the learning rate and the meta-learning rate. Our solution is inspired from hypergradient descent (HD) algorithm (Baydin et al., 2018), which automatically updates the learning rate by performing gradient descent on the learning rate alongside the original optimization steps. The proposed algorithm does not need any extra gradient computations and just involves storing the gradients from the previous optimization step.

2

Scaling the Convex Barrier with Active Sets

Contents

Abstract

Tight and efficient neural network bounding is of critical importance for the scaling of neural network verification systems. A number of efficient specialised dual solvers for neural network bounds have been presented recently, but they are often too loose to verify more challenging properties. This lack of tightness is linked to the weakness of the employed relaxation, which is usually a linear program of size linear in the number of neurons. While a tighter linear relaxation for piecewise linear activations exists, it comes at the cost of exponentially many constraints and thus currently lacks an efficient customised solver. We alleviate this deficiency via a novel dual algorithm that realises the full potential of the new relaxation by operating on a small active set of dual variables. Our method recovers the strengths of the new relaxation in the dual space: tightness and a linear separation oracle. At the same time, it shares the benefits of previous dual approaches for weaker relaxations: massive parallelism, GPU implementation, low cost per iteration and valid bounds at any time. As a consequence, we obtain better bounds than off-the-shelf solvers in only a fraction of their running time and recover the speed-accuracy trade-offs of looser dual solvers if the computational budget is small. We demonstrate that this results in significant formal verification speed-ups.

2.1 Introduction

Verification requires formally proving or disproving that a given property of a neural network holds over all inputs in a specified domain. We consider properties in their canonical form (Bunel et al., 2018), which requires us to either: (i) prove that no input results in a negative output (property is true); or (ii) identify a counter-example (property is false). The search for counter-examples is typically performed by efficient methods such as random sampling of the input domain (Webb et al., 2019), or projected gradient descent (Carlini and Wagner, 2017). In contrast, establishing the veracity of a property requires solving a suitable convex relaxation to obtain a lower bound on the minimum output. If the lower bound is positive, the given property is true. If the bound is negative and no counter-example is found, either: (i) we make no conclusions regarding the property (incomplete verification); or (ii) we further refine the counter-example search and lower bound computation within a branch-and-bound framework until we reach a concrete conclusion (complete verification).

The main bottleneck of branch and bound is the computation of the lower bound for each node of the enumeration tree via convex optimization. While earlier works relied on off-the-shelf solvers (Ehlers, 2017; Bunel et al., 2018), it was quickly established that such an approach does not scale-up elegantly with the size of the neural network. This has motivated researchers to design specialized dual solvers (Dvijotham et al., 2019; Bunel et al., 2020a), thereby providing initial evidence that verification can be realised in practice. However, the convex relaxation considered in the dual solvers is itself very weak (Ehlers, 2017), hitting what is now commonly referred to as the "convex barrier" (Salman et al., 2019). In practice, this implies that either several properties remain undecided in incomplete verification, or take several hours to be verified exactly.

Multiple works have tried to overcome the convex barrier for piecewise linear activations (Raghunathan et al., 2018; Singh et al., 2019a). Here, we focus on the single-neuron Linear Programming (LP) relaxation by Anderson et al. (2020). Unfortunately, its tightness comes at the price of exponentially many (in the

number of variables) constraints. Therefore, existing dual solvers (Dvijotham et al., 2018; Bunel et al., 2020a) are not easily applicable, limiting the scaling of the new relaxation.

We address this problem by presenting a specialized dual solver for the relaxation by Anderson et al. (2020), which realises its full potential by meeting the following desiderata:

- By keeping an *active set* of dual variables, we obtain a *sparse* dual solver that recovers the strengths of the original primal problem (Anderson et al., 2020) in the dual domain. In line with previous dual solvers, our approach yields valid bounds at anytime, leverages convolutional network structure and enjoys *massive parallelism* within a GPU implementation, resulting in better bounds in an order of magnitude less time than off-the-shelf solvers (Gurobi Optimization, 2020).
- We present a unified dual treatment that includes both a linearly sized LP relaxation (Ehlers, 2017) and the tighter formulation. As a consequence, our solver provides a *wide range of speed-accuracy trade-offs*: (i) it is competitive with dual approaches on the looser relaxation (Dvijotham et al., 2018; Bunel et al., 2020a); and (ii) it yields much tighter bounds if a larger computational budget is available. Owing to this flexibility, we show that our dual algorithm yields large complete verification gains compared to primal approaches (Anderson et al., 2020) and previous dual algorithms.

2.2 Preliminaries: Neural Network Relaxations

We denote vectors by bold lower case letters (for example, $\mathbf{x}$) and matrices by upper case letters (for example, W). We use $\odot$ for the Hadamard product, $[\![\cdot]\!]$ for integer ranges, $\mathbb{1}_{\mathbf{a}}$ for the indicator vector on condition $\mathbf{a}$ and brackets for intervals ($[\mathbf{l}_k, \mathbf{u}_k]$) and vector or matrix entries ($\mathbf{x}[i]$ or $W[i, j]$). In addition, given $W \in \mathbb{R}^{m \times n}$ and $\mathbf{x} \in \mathbb{R}^m$, we will employ $W \diamond \mathbf{x}$ and $W \square \mathbf{x}$ as shorthands for respectively $\sum_i \text{col}_i(W) \odot \mathbf{x}$ and $\sum_i \text{col}_i(W)^T \mathbf{x}$, where $\text{col}_i(W)$ denotes the i-th column of matrix W.

Let $\mathcal{C}$ be the network input domain. Similar to Dvijotham et al. (2018); Bunel et al. (2020a), we assume that linear minimisation over $\mathcal{C}$ can be performed in closed-form. Our goal is to compute bounds on the scalar output of a piecewise-linear feedforward neural network. The tightest possible lower bound can be obtained by solving the following optimization problem:

$$\min_{\mathbf{x},\hat{\mathbf{x}}} \quad \hat{x}_n \qquad \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \tag{2.1a}$$

$$\hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} \quad k \in [\![0, n-1]\!], \tag{2.1b}$$

$$\mathbf{x}_k = \sigma\left(\hat{\mathbf{x}}_k\right) \qquad k \in [\![1, n-1]\!], \tag{2.1c}$$

where the activation function $\sigma\left(\hat{\mathbf{x}}_k\right)$ is piecewise-linear, $\hat{\mathbf{x}}_k, \mathbf{x}_k \in \mathbb{R}^{n_k}$ denote the outputs of the k-th linear layer (fully-connected or convolutional) and activation function respectively, W_k and $\mathbf{b}_k$ denote its weight matrix and bias, n_k is the number of activations at layer k. We will focus on the ReLU case ($\sigma\left(\mathbf{x}\right) = \max\left(\mathbf{x}, 0\right)$), as common piecewise-linear functions can be expressed as a composition of ReLUs (Bunel et al., 2020b).

Problem equation 2.1 is non-convex due to the activation function's non-linearity equation 2.1c. As solving it is NP-hard (Katz et al., 2017), it is commonly approximated by a convex relaxation (see §2.4). The quality of the corresponding bounds, which is fundamental in verification, depends on the tightness of the relaxation. Unfortunately, tight relaxations usually correspond to slower bounding procedures. We first review a popular ReLU relaxation in §2.2.1). We then consider a tighter one in §2.2.2.

2.2.1 Planet Relaxation

The so-called Planet relaxation (Ehlers, 2017) has enjoyed widespread use due to its amenability to efficient customised solvers (Dvijotham et al., 2018; Bunel et al., 2020a) and is the "relaxation of choice" for many works in the area (Bunel et al., 2020b; Lu and Kumar, 2020). Here, we describe it in its non-projected form $\mathcal{M}_k$, the

LP relaxation of the Big-M Mixed Integer Programming (MIP) formulation (Tjeng et al., 2019). Applying $\mathcal{M}_k$ to problem equation 2.1 results in:

$$\begin{aligned}
\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \quad \hat{x}_n \quad \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C} \\
& \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!], \\
& \left.\begin{array}{l} \mathbf{x}_k \geq \hat{\mathbf{x}}_k, \quad \mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k, \\ \mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k), \\ (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \times [\mathbf{0},\ \mathbf{1}] \end{array}\right\} := \mathcal{M}_k && k \in [\![1, n-1]\!],
\end{aligned} \tag{2.2}$$

where $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ and $\mathbf{l}_k, \mathbf{u}_k$ are *intermediate bounds* respectively on pre-activation variables $\hat{\mathbf{x}}_k$ and post-activation variables $\mathbf{x}_k$. These constants play an important role in the structure of $\mathcal{M}_k$ and, together with the relaxed binary constraints on $\mathbf{z}$, define box constraints on the variables. We detail how to compute intermediate bounds in appendix A.5. Projecting out auxiliary variables $\mathbf{z}$ results in the Planet relaxation (cf. appendix A.2.1 for details), which replaces equation 2.1c by its convex hull.

Problem equation 2.2, which is linearly-sized, can be easily solved via commercial black-box LP solvers (Bunel et al., 2018). This does not scale-up well with the size of the neural network, motivating the need for specialised solvers. Customised dual solvers have been designed by relaxing constraints equation 2.1b, equation 2.1c (Dvijotham et al., 2018) or replacing equation 2.1c by the Planet relaxation and employing Lagrangian Decomposition (Bunel et al., 2020a). Both approaches result in bounds very close to optimality for problem equation 2.2 in only a fraction of the runtime of off-the-shelf solvers.

2.2.2 A Tighter Relaxation

A much tighter approximation of problem equation 2.1 than the Planet relaxation (§2.2.1) can be obtained by representing the convex hull of the composition of equation 2.1b and equation 2.1c rather than the convex hull of equation 2.1c alone. A formulation of this type was recently introduced by Anderson et al. (2020). Let us define $\check{L}_{k-1}, \check{U}_{k-1} \in \mathbb{R}^{n_k \times n_{k-1}}$ as: $\check{L}_{k-1}[i,j] = \mathbf{l}_{k-1}[j]\mathbb{1}_{W_k[i,j]\geq 0} + \mathbf{u}_{k-1}[j]\mathbb{1}_{W_k[i,j]<0}$, and $\check{U}_{k-1}[i,j] = \mathbf{u}_{k-1}[j]\mathbb{1}_{W_k[i,j]\geq 0} + \mathbf{l}_{k-1}[j]\mathbb{1}_{W_k[i,j]<0}$. Additionally, let us introduce $2^{W_k} = \{0,1\}^{n_k \times n_{k-1}}$, the set of all possible binary masks of weight matrix W_k,

and $\mathcal{E}_k := 2^{W_k} \setminus \{0, 1\}$, which excludes the all-zero and all-one masks. The new representation results in the following primal problem:

$$
\begin{aligned}
\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \quad & \hat{x}_n \\
\text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C} \\
& \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!]\,, \\
& (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in \mathcal{M}_k \\
& \mathbf{x}_k \leq \left.\begin{pmatrix} (W_k \odot I_k)\,\mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k \\ -\left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k) \\ +\left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \end{pmatrix} \; \forall I_k \in \mathcal{E}_k \right\} := \mathcal{A}_k && k \in [\![1, n-1]\!]\,.
\end{aligned}
\tag{2.3}
$$

Both $\mathcal{M}_k$ and $\mathcal{A}_k$ yield valid MIP formulations for problem equation 2.1 when imposing integrality constraints on $\mathbf{z}$. However, the LP relaxation of $\mathcal{A}_k$ will yield tighter bounds. In the worst case, this tightness comes at the cost of exponentially many constraints: one for each $I_k \in \mathcal{E}_k$. On the other hand, given a set of primal assignments $(\mathbf{x}, \mathbf{z})$ that are not necessarily feasible for problem equation 2.3, one can efficiently compute the most violated constraint (if any) at that point. The mask associated to such constraint can be computed in linear-time (Anderson et al., 2020) as:

$$
I_k[i,j] = \mathbb{1}^T_{\left((1-\mathbf{z}_k[i]) \odot \check{L}_{k-1}[i,j] + \mathbf{z}_k[i] \odot \check{U}_{k-1}[i,j] - \mathbf{x}_{k-1}[i]\right) W_k[i,j] \geq 0}. \tag{2.4}
$$

We point out that $\mathcal{A}_k$ slightly differs from the original formulation of Anderson et al. (2020), which does not explicitly include pre-activation bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ (which we treat via $\mathcal{M}_k$). While this was implicitly addressed in practical applications (Botoeva et al., 2020), not doing so has a strong negative effect on bound tightness, possibly to the point of yielding looser bounds than problem equation 2.2. In appendix A.6, we provide an example in which this is the case and extend the original derivation by Anderson et al. (2020) to recover $\mathcal{A}_k$ as in problem equation 2.3.

Owing to the exponential number of constraints, problem equation 2.3 cannot be solved as it is. As outlined by Anderson et al. (2020), the availability of a linear-time separation oracle equation 2.4 offers a natural primal *cutting plane* algorithm, which can then be implemented in off-the-shelf solvers: solve the Big-M

LP equation 2.2, then iteratively add the most violated constraints from $\mathcal{A}_k$ at the optimal solution. When applied to the verification of small neural networks via off-the-shelf MIP solvers, this leads to substantial gains with respect to the looser Big-M relaxation (Anderson et al., 2020).

2.3 An Efficient Dual Solver for the Tighter Relaxation

Inspired by the success of dual approaches on looser relaxations (Bunel et al., 2020a; Dvijotham et al., 2019), we show that the formal verification gains by Anderson et al. (2020) (see §2.2.2) scale to larger networks if we solve the tighter relaxation in the dual space. Due to the particular structure of the relaxation, a customised solver for problem equation 2.3 needs to meet a number of requirements.

Fact 1. *In order to replicate the success of previous dual algorithms on looser relaxations, we need a solver for problem equation 2.3 with the following properties: (i)* sparsity*: a memory cost linear in the number of network activations in spite of exponentially many constraints, (ii)* tightness*: the bounds should reflect the quality of those obtained in the primal space, (iii)* anytime*: low cost per iteration and valid bounds at each step.*

The anytime requirement motivates dual solutions: any dual assignment yields a valid bound due to weak duality. Unfortunately, as shown in appendix A.1, neither of the two dual derivations by Bunel et al. (2020a); Dvijotham et al. (2018) readily satisfy all desiderata at once. Therefore, we need a completely different approach. Let us introduce dual variables $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and functions thereof:

$$\begin{aligned} \boldsymbol{f}_k(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \;& \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \textstyle\sum_{I_k} \boldsymbol{\beta}_{k,I_k} + \sum_{I_{k+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}}, \\ \boldsymbol{g}_k(\boldsymbol{\beta}) = & \left[\begin{array}{l} \sum_{I_k \in \mathcal{E}_k} \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k \\ + \sum_{I_k \in \mathcal{E}_k} \left(W_k \odot I_k \odot \check{L}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \sum_{I_k \in \mathcal{E}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k, \end{array} \right. \end{aligned} \tag{2.5}$$

where $\sum_{I_k}$ is a shorthand for $\sum_{I_k \in 2^{W_k}}$. Starting from primal equation 2.3, we relax all constraints in $\mathcal{A}_k$ except box constraints (see §2.2.1). We obtain the following

dual problem (derivation in appendix A.3), where functions $\boldsymbol{f}_k, \boldsymbol{g}_k$ appear in inner products with primal variables $\mathbf{x}_k, \mathbf{z}_k$:

$$\begin{aligned} &\max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\geq 0} d(\boldsymbol{\alpha},\boldsymbol{\beta}) \qquad \text{where:} \qquad d(\boldsymbol{\alpha},\boldsymbol{\beta}) := \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}), \\ &\mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}) = \left[\begin{array}{l} \sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k - \sum_{k=0}^{n-1} \boldsymbol{f}_k(\boldsymbol{\alpha},\boldsymbol{\beta})^T \mathbf{x}_k - \sum_{k=1}^{n-1} \boldsymbol{g}_k(\boldsymbol{\beta})^T \mathbf{z}_k \\ + \sum_{k=1}^{n-1} \left(\sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \square \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k) \right) \end{array} \right. \\ &\text{s.t.} \qquad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad k \in [\![1, n-1]\!]. \end{aligned} \tag{2.6}$$

This is again a challenging problem: the exponentially many constraints in the primal equation 2.3 are now associated to an exponential number of variables. Nevertheless, we show that the requirements of Fact 1 can be met by operating on a restricted version of dual equation 2.6. To this end, we present Active Set, a specialised solver for the relaxation by Anderson et al. (2020) that is sparse, anytime and yields bounds reflecting the tightness of the new relaxation. Starting from the dual of problem equation 2.2, our solver iteratively adds variables to a small active set of dual variables $\boldsymbol{\beta}_{\mathcal{B}}$ and solves the resulting reduced version of problem equation 2.6. We first describe our solver on a fixed $\boldsymbol{\beta}_{\mathcal{B}}$ and then outline how to iteratively modify the active set (§2.3.2). Pseudo-code can be found in appendix A.4.

2.3.1 Active Set Solver

We want to solve a version of problem equation 2.6 for which the sums over the I_k masks of each layer k are restricted to $\mathcal{B}_k \subseteq \mathcal{E}_k$[1], with $\mathcal{B} = \cup_k \mathcal{B}_k$. By keeping $\mathcal{B} = \emptyset$, we recover a novel dual solver for the Big-M relaxation equation 2.2 (explicitly described in appendix A.2), which is employed as initialisation. Setting $\boldsymbol{\beta}_{k,I_k} = 0, \forall\ I_k \in \mathcal{E}_k \setminus \mathcal{B}_k$ in equation 2.5, equation 2.6 and removing these from the formulation, we obtain:

$$\begin{aligned} \boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}) &= \left[\begin{array}{l} \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \sum_{I_k \in \mathcal{B}_k \cup \{0,1\}} \boldsymbol{\beta}_{k,I_k}, \\ + \sum_{I_{k+1} \in \mathcal{B}_{k+1} \cup \{0,1\}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}} \end{array} \right. \\ \boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_{\mathcal{B}}) &= \left[\begin{array}{l} \sum_{I_k \in \mathcal{B}_k} \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k \\ + \sum_{I_k \in \mathcal{B}_k} \left(W_k \odot I_k \odot \check{L}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \sum_{I_k \in \mathcal{B}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k, \end{array} \right. \end{aligned} \tag{2.7}$$

[1] As dual variables $\boldsymbol{\beta}_{k,I_k}$ are indexed by I_k, $\mathcal{B} = \cup_k \mathcal{B}_k$ implicitly defines an active set of variables $\boldsymbol{\beta}_{\mathcal{B}}$.

along with the reduced dual problem:

$$\max_{(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}})\geq 0} d_{\mathcal{B}}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}) \qquad \text{where:} \qquad d_{\mathcal{B}}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}) := \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}_{\mathcal{B}}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}),$$
$$\mathcal{L}_{\mathcal{B}}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}) = \left[\begin{array}{l} \sum_{k=1}^{n-1} \mathbf{b}_k^T\boldsymbol{\alpha}_k - \sum_{k=0}^{n-1} \boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}})^T\mathbf{x}_k - \sum_{k=1}^{n-1} \boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_{\mathcal{B}})^T\mathbf{z}_k \\ + \sum_{k=1}^{n-1} \Big(\sum_{I_k\in\mathcal{B}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \square \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T(\hat{\mathbf{l}}_k - \mathbf{b}_k)\Big) \end{array}\right.$$
$$\text{s.t.} \qquad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k,\mathbf{z}_k) \in [\mathbf{l}_k,\mathbf{u}_k]\times[\mathbf{0},\ \mathbf{1}] \qquad k\in[\![1,n-1]\!]. \tag{2.8}$$

We can maximize $d_{\mathcal{B}}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}})$, which is concave and non-smooth, via projected supergradient ascent or variants thereof, such as Adam (Kingma and Ba, 2015). In order to obtain a valid supergradient, we need to perform the inner minimisation over the primals. Thanks to the structure of problem equation 2.8, the optimisation decomposes over the layers. For $k\in[\![1,n-1]\!]$, we can perform the minimisation in closed-form by driving the primals to their upper or lower bounds depending on the sign of their coefficients:

$$\mathbf{x}_k^* = \mathbb{1}_{\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}})\geq 0} \odot \hat{\mathbf{u}}_k + \mathbb{1}_{\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}})<0} \odot \hat{\mathbf{l}}_k, \qquad \mathbf{z}_k^* = \mathbb{1}_{\boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_{\mathcal{B}})\geq 0} \odot \mathbf{1}. \tag{2.9}$$

The subproblem corresponding to $\mathbf{x}_0$ is different, as it involves a linear minimization over $\mathbf{x}_0 \in \mathcal{C}$:

$$\mathbf{x}_0^* \in \ \arg\min_{\mathbf{x}_0} \quad \boldsymbol{f}_{\mathcal{B},0}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}})^T\mathbf{x}_0 \qquad \text{s.t.} \quad \mathbf{x}_0\in\mathcal{C}. \tag{2.10}$$

We assumed in § 2.2 that equation 2.10 can be performed efficiently. We refer the reader to Bunel et al. (2020a) for descriptions of the minimisation when $\mathcal{C}$ is a ℓ_∞ or ℓ_2 ball, as common for adversarial examples.

Given $(\mathbf{x}^*,\mathbf{z}^*)$ as above, the supergradient of $d_{\mathcal{B}}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}})$ is a subset of the one for $d(\boldsymbol{\alpha},\boldsymbol{\beta})$, given by:

$$\nabla_{\boldsymbol{\alpha}_k} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = W_k\mathbf{x}_{k-1}^* + \mathbf{b}_k - \mathbf{x}_k^*, \qquad \nabla_{\boldsymbol{\beta}_{k,0}} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathbf{x}_k^* - \mathbf{z}_k^* \odot \hat{\mathbf{u}}_k,$$
$$\nabla_{\boldsymbol{\beta}_{k,1}} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathbf{x}_k^* - \Big(W_k\mathbf{x}_{k-1}^* + \mathbf{b}_k\Big) + (1-\mathbf{z}_k^*)\odot\hat{\mathbf{l}}_k,$$
$$\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = \left(\begin{array}{l} \mathbf{x}_k^* - (W_k\odot I_k)\,\mathbf{x}_{k-1}^* + \Big(W_k\odot I_k\odot\check{L}_{k-1}\Big)\diamond(1-\mathbf{z}_k^*) \\ -\mathbf{z}_k^*\odot\mathbf{b}_k + \Big(W_k\odot(1-I_k)\odot\check{U}_{k-1}\Big)\diamond\mathbf{z}_k^* \end{array}\right) I_k\in\mathcal{B}_k, \tag{2.11}$$

for each $k\in[\![0,n-1]\!]$. At each iteration, after taking a step in the supergradient direction, the dual variables are projected to the non-negative orthant by clipping negative values.

2.3.2 Extending the Active Set

We initialise the dual equation 2.6 with a tight bound on the Big-M relaxation by solving for $d_\emptyset(\boldsymbol{\alpha}, \boldsymbol{\beta}_\emptyset)$ in equation 2.8. To satisfy the tightness requirement in Fact 1, we then need to include constraints (via their Lagrangian multipliers) from the exponential family of $\mathcal{A}_k$ into $\mathcal{B}_k$. Our goal is to tighten them as much as possible while keeping the active set small to save memory and compute.

The active set strategy is defined by a *selection criterion* for the I_k^* to be added[2] to $\mathcal{B}_k$, and the *frequency* of addition. In practice, *we add the variables maximising the entries of supergradient* $\nabla_{\beta_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ *after a fixed number of dual iterations.* We now provide motivation for both choices.

Selection criterion The selection criterion needs to be computationally efficient. Thus, we proceed greedily and focus only on the immediate effect at the current iteration. Let us map a restricted set of dual variables $\boldsymbol{\beta}_\mathcal{B}$ to a set of dual variables $\boldsymbol{\beta}$ for the full dual equation 2.6. We do so by setting variables not in the active set to 0: $\boldsymbol{\beta}_{\bar{\mathcal{B}}} = 0$, and $\boldsymbol{\beta} = \boldsymbol{\beta}_\mathcal{B} \cup \boldsymbol{\beta}_{\bar{\mathcal{B}}}$. Then, for each layer k, we add the set of variables $\boldsymbol{\beta}_{k,I_k^*}$ maximising the corresponding entries of the supergradient of the full dual problem equation 2.6: $\boldsymbol{\beta}_{k,I_k^*} \in \arg\max_{\boldsymbol{\beta}_{k,I_k}} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}\}$. Therefore, we use the subderivatives as a proxy for short-term improvement on the full dual objective $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$. Under a primal interpretation, our selection criterion involves a call to the separation oracle equation 2.4 by Anderson et al. (2020).

Proposition 1. *$\boldsymbol{\beta}_{k,I_k^*}$ (as defined above) represents the Lagrangian multipliers associated to the most violated constraints from $\mathcal{A}_k$ at $(\mathbf{x}^*, \mathbf{z}^*) \in \arg\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_\mathcal{B}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_\mathcal{B})$, the primal minimiser of the current restricted Lagrangian.*

Proof. See appendix A.4.1. □

[2] adding a single I_k^* mask to $\mathcal{B}_k$ extends $\boldsymbol{\beta}_\mathcal{B}$ by n_k variables: one for each neuron at layer k.

Frequency Finally, we need to decide the frequency at which to add variables to the active set.

Fact 2. *Assume we obtained a dual solution* $(\boldsymbol{\alpha}^\dagger, \boldsymbol{\beta}_{\mathcal{B}}^\dagger) \in \arg\max d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$ *using Active Set on the current* $\mathcal{B}$*. Then* $(\mathbf{x}^*, \mathbf{z}^*) \in \arg\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^\dagger, \boldsymbol{\beta}_{\mathcal{B}}^\dagger)$ *is not necessarily an optimal primal solution for the primal of the current restricted dual problem (Sherali and Choi, 1996).*

The primal of $d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$ (restricted primal) is the problem obtained by setting $\mathcal{E}_k \leftarrow \mathcal{B}_k$ in problem equation 2.3. While the primal cutting plane algorithm by Anderson et al. (2020) calls the separation oracle equation 2.4 at the optimal solution of the current restricted primal, Fact 2 shows that our selection criterion leads to a different behaviour even at dual optimality for $d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$. Therefore, as we have no theoretical incentive to reach (approximate) subproblem convergence, we add variables after a fixed tunable number of supergradient iterations. Furthermore, we can add more than one variable “at once" by running the oracle equation 2.4 repeatedly for a number of iterations.

We conclude this section by pointing out that, while recovering primal optima is possible in principle (Sherali and Choi, 1996), doing so would require dual convergence on each restricted dual problem equation 2.8. As the main advantage of dual approaches (Dvijotham et al., 2018; Bunel et al., 2020a) is their ability to quickly achieve tight bounds (rather than formal optimality), adapting the selection criterion to mirror the primal cutting plane algorithm would defeat the purpose of Active Set.

2.3.3 Implementation Details, Technical Challenges

Analogously to previous dual algorithms (Dvijotham et al., 2018; Bunel et al., 2020a), our approach can leverage the massive parallelism offered by modern GPU architectures in three different ways. First, we execute in parallel the computations of lower and upper bounds relative to all the neurons of a given layer. Second, in complete verification, we can batch over the different Branch and Bound (BaB) subproblems. Third, as most of our solver relies on standard

linear algebra operations employed during the forward and backward passes of neural networks, we can exploit the highly optimised implementations commonly found in modern deep learning frameworks.

An exception are what we call “masked" forward/backward passes: operations of the form $(W_k \odot I_k)\,\mathbf{x}_k$ or $(W_k \odot I_k)^T\,\mathbf{x}_{k+1}$, which are needed whenever dealing with constraints from $\mathcal{A}_k$. In our solver, they appear if $\mathcal{B}_k \neq \emptyset$ (see equations equation 2.8, equation 2.11). Masked passes require a customised lower-level implementation for a proper treatment of convolutional layers, detailed in appendix A.7.

2.4 Related Work

In addition to those described in §2.2, many other relaxations have been proposed in the literature. In fact, all bounding methods are equivalent to solving some convex relaxation of a neural network. This holds for conceptually different ideas such as bound propagation (Gowal et al., 2018b), specific dual assignments (Wong and Kolter, 2018), dual formulations based on Lagrangian Relaxation (Dvijotham et al., 2018) or Decomposition (Bunel et al., 2020a). The degree of tightness varies greatly: from looser relaxations associated to closed-form methods (Gowal et al., 2018b; Weng et al., 2018; Wong and Kolter, 2018) to tighter formulations based on Semi-Definite Programming (SDP) (Raghunathan et al., 2018).

The speed of closed-form approaches results from simplifying the triangle-shaped feasible region of the Planet relaxation (§2.2.1) (Singh et al., 2018; Wang et al., 2018). On the other hand, tighter relaxations are more expressive than the linearly-sized LP by (Ehlers, 2017). The SDP formulation by Raghunathan et al. (2018) can represent interactions between activations in the same layer. Similarly, Singh et al. (2019a) tighten the Planet relaxation by considering the convex hull of the union of polyhedra relative to k ReLUs of a given layer at once. Alternatively, tighter LPs can be obtained by considering the ReLU together with the affine operator before it: standard MIP techniques (Jeroslow, 1987) lead to a formulation that is quadratic in the number of variables (see appendix A.6.2). The relaxation by Anderson et al. (2020) detailed in §2.2.2 is a more convenient representation of the same set.

By projecting out the auxiliary $\mathbf{z}$ variables, (Tjandraatmadja et al., 2020) recently introduced another formulation equivalent to the one by Anderson et al. (2020), with half as many variables and a linear factor more constraints compared to what described in §2.2.2. Therefore, the relationship between the two formulations mirrors the one between the Planet and Big-M relaxations (see appendix A.2.1). Our dual derivation and the Active Set algorithm can be adapted to operate on the projected relaxations.

Specialised dual solvers significantly improve in bounding efficiency with respect to off-the-shelf solvers for both LP (Bunel et al., 2020a) and SDP formulations (Dvijotham et al., 2019). Therefore, the design of similar solvers for other tight relaxations is an interesting line of future research. We contribute with a specialised dual solver for the relaxation by Anderson et al. (2020) (§2.3). In what follows, we demonstrate empirically that by seamlessly transitioning from the Planet relaxation to the tighter formulation, we can obtain large incomplete and complete verification improvements.

2.5 Experiments

We empirically demonstrate the effectiveness of our method under two settings. On incomplete verification (§2.5.1), we assess the speed and quality of bounds compared to other bounding algorithms. On complete verification (§2.5.2), we examine whether our speed-accuracy trade-offs correspond to faster exact verification. Our implementation is based on Pytorch (Paszke et al., 2017) and is available at `https://github.com/oval-group/scaling-the-convex-barrier`.

2.5.1 Incomplete Verification

We evaluate incomplete verification performance by upper bounding the robustness margin (the difference between the ground truth logit and the other logits) to adversarial perturbations (Szegedy et al., 2014) on the CIFAR-10 test set (Krizhevsky and Hinton, 2009). If the upper bound is negative, we can certify the network's vulnerability to adversarial perturbations. We replicate the experimental setting

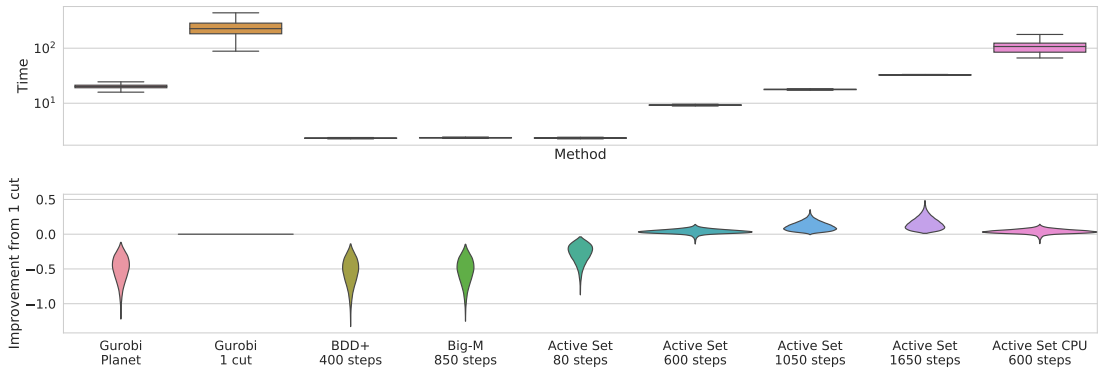
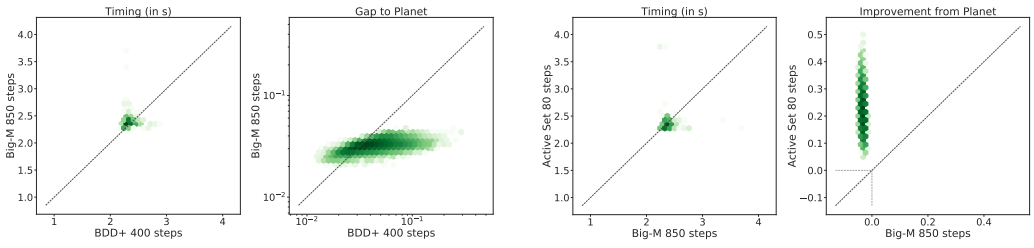


Figure 2.1: Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the SGD-trained network from Bunel et al. (2020a). The width at a given value represents the proportion of problems for which this is the result. Comparing Active Sets with 1650 steps to Gurobi 1 Cut, tighter bounds are achieved with a smaller runtime.

from Bunel et al. (2020a). The networks correspond to the small network architecture from Wong and Kolter (2018). Here, we present results for a network trained via standard SGD and cross entropy loss, with no modification to the objective for robustness. Perturbations for this network lie in a ℓ_∞ norm ball with radius $\epsilon_{ver} = 5/255$ (which is hence lower than commonly employed radii for robustly trained networks). In appendix A.9, we provide additional CIFAR-10 results on an adversarially trained network using the method by Madry et al. (2018), and on MNIST (LeCun et al., 1998), for a network adversarially trained with the algorithm by Wong and Kolter (2018).

We compare both against previous dual iterative methods and Gurobi (Gurobi Op-



(a) Comparison of runtime (left) and gap to Gurobi Planet bounds (right). For the latter, lower is better.

(b) Comparison of runtime (left) and difference with the Gurobi Planet bounds (right). For the latter, higher is better.

Figure 2.2: Pointwise comparison for a subset of the methods on the data presented in Figure 2.1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

timization, 2020), the commercial black-box solver employed by Anderson et al. (2020). For Gurobi-based baselines, **Planet** means solving the Planet Ehlers (2017) relaxation of the network, while **Gurobi cut** starts from the Big-M relaxation and adds constraints from $\mathcal{A}_k$ in a cutting-plane fashion, as the original primal algorithm by Anderson et al. (2020). We run both on 4 CPU threads. Amongst dual iterative methods, run on an Nvidia Titan Xp GPU, we compare with **BDD+**, the recent proximal-based solver by Bunel et al. (2020a), operating on a Lagrangian Decomposition dual of the Planet relaxation. As we operate on (a subset of) the data by Bunel et al. (2020a), we omit both their supergradient-based approach and the one by Dvijotham et al. (2018), as they both perform worse than BDD+ (Bunel et al., 2020a). For the same reason, we omit cheaper (and looser) methods, like interval propagation Gowal et al. (2018b) and the one by Wong and Kolter (2018). **Active Set** denotes our solver for problem equation 2.3, described in §2.3.1. By keeping $\mathcal{B} = \emptyset$, Active Set reduces to **Big-M**, a solver for the non-projected Planet relaxation (appendix A.2), which can be seen as Active Set's initialiser. In line with previous bounding algorithms (Bunel et al., 2020a), we employ Adam updates (Kingma and Ba, 2015) for supergradient-type methods due to their faster empirical convergence. Finally, we complement the comparison with Gurobi-based methods by running Active Set on 4 CPU threads (**Active Set CPU**). Further details, including hyper-parameters, can be found in appendix A.9.

Figure 2.1 shows the distribution of runtime and the bound improvement with respect to Gurobi cut for the SGD-trained network. For Gurobi cut, we only add the single most violated cut from $\mathcal{A}_k$ per neuron, due to the cost of repeatedly solving the LP. We tuned BDD+ and Big-M, the dual methods operating on the weaker relaxation equation 2.2, to have the same average runtime. They obtain bounds comparable to Gurobi Planet in one order less time. Initialised from 500 Big-M iterations, at 600 iterations Active Set already achieves better bounds on average than Gurobi cut in around $1/20^{th}$ of the time. With a computational budget twice as large (1050 iterations) or four times as large (1650 iterations), the bounds significantly improve over Gurobi cut in still a fraction of the time.

As we empirically demonstrate in appendix A.9, the tightness of the Active Set bounds is strongly linked to our active set strategy (§2.3.2). Remarkably, even if our method is specifically designed to take advantage of GPU acceleration, executing it on CPU proves to be strongly competitive with Gurobi cut, producing better bounds in less time for the benchmark of Figure 2.1.

Figure 2.2 shows pointwise comparisons for a subset of the methods of Figure 2.1, on the same data. Figure 2.2a shows the gap to the (Gurobi) Planet bound for BDD+ and our Big-M solver. Surprisingly, our Big-M solver is competitive with BDD+, achieving on average better bounds than BDD+, in the same time. Figure 2.2b shows the improvement over Planet bounds for Big-M and Active Set. The latter achieves markedly better bounds than Big-M in the same time, demonstrating the benefit of operating (at least partly) on the tighter dual equation 2.6.

2.5.2 Complete Verification

We next evaluate the performance on complete verification, verifying the adversarial robustness of a network to perturbations in ℓ_∞ norm on a subset of the dataset by Lu and Kumar (2020), replicating the experimental setting from Bunel et al. (2020a). The dataset associates a different perturbation radius ϵ_{verif} to each CIFAR-10 image, so as to create challenging verification properties. Its difficulty makes the dataset an appropriate testing ground for tighter relaxations like the one

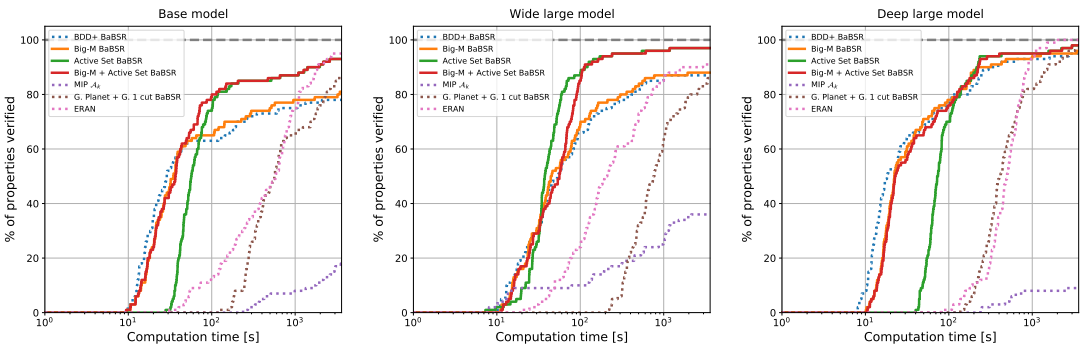


Figure 2.3: Cactus plots on properties from Lu and Kumar (2020), displaying the percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

by Anderson et al. (2020) (§2.2.2). Further details, including network architectures, can be found in appendix A.9.

Here, we aim to solve the non-convex problem equation 2.1 directly, rather than an approximation like §2.5.1. In order to do so, we use BaSBR, a branch and bound algorithm from Bunel et al. (2020b). Branch and Bound works by dividing the problem domain into subproblems (branching) and bounding the local minimum over those domains. Any domain which cannot contain the global lower bound is pruned away, whereas the others are kept and branched over. In BaBSR, branching is carried out by splitting an unfixed ReLU into its passing and blocking phases. The ReLU which induces maximum change in the domain's lower bound, when made unambiguous, is selected for splitting.

A fundamental component of a BaB method is the bounding algorithm, which is, in general, the computational bottleneck (Lu and Kumar, 2020). Therefore, we compare the effect on final verification time of using the different bounding methods in §2.5.1 within BaBSR. In addition, we evaluate **MIP** $\mathcal{A}_k$, which encodes problem equation 2.1 as a Big-M MIP (Tjeng et al., 2019) and solves it in Gurobi by adding cutting planes from $\mathcal{A}_k$, analogously to the original experiments by Anderson et al. (2020). Finally, we also compare against **ERAN** (Singh et al., 2020), a state-of-the-art complete verification toolbox: results on the dataset by Lu and Kumar (2020) are taken from the recent VNN-COMP competition (VNN-COMP, 2020). We use 100 iterations for Active Set, 100 iterations for BDD+ and 180 iterations for Big-M. For dual iterative algorithms, we solve 300 subproblems at once for the base network and 200 for the deep and wide networks (see §2.3.3). Additionally, dual variables are initialised from their parent node's bounding computation. As in Bunel et al. (2020a), the time-limit is kept at one hour. Due to the difference in computational cost between algorithms operating on the tighter relaxation by Anderson et al. (2020) and the other bounding algorithms[3], we also experiment with a *stratified* version of the bounding within BaBSR. We devise a set of heuristics to determine whether a given subproblem is easy (therefore looser bounds are

[3]For Active Set, this is partly due to the masked forward/backward pass described in appendix A.7.

	Base			Wide			Deep		
Method	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout
BDD+ BaBSR	883.55	82 699	22.00	568.25	43 752	13.00	281.47	10 763	5.00
Big-M BaBSR	826.60	68 582	19.00	533.79	35 877	12.00	253.37	9347	4.00
A. Set BaBSR	422.32	**9472**	**7.00**	**169.73**	**1873**	**3.00**	227.26	**2302**	**2.00**
Big-M + A. Set BaBSR	**402.88**	11 409	**7.00**	179.73	3713	**3.00**	**197.99**	3087	**2.00**
G. Planet + G. 1 cut BaBSR	1191.48	2044	14.00	1272.99	1352	10.00	704.59	678	3.00
MIP $\mathcal{A}_k$	3227.50	226	82.00	2500.70	101	64.00	3339.37	435	91.00
ERAN	805.89	-	5.00	632.12	-	9.00	545.72	-	0.00

Table 2.1: We compare average solving time, average number of solved sub-problems and the percentage of timed out properties on data from Lu and Kumar (2020). The best dual iterative method is highlighted in bold.

sufficient) or whether we need to operate on the tighter relaxation. Instances of this approach are **Big-M + Active Set** and **Gurobi Planet + Gurobi 1 cut**. Further details are provided in appendix A.8.

Figure 2.3 and Table 2.1 show that Big-M performs competitively with BDD+. Active Set verifies a larger share of properties than the methods operating on the looser formulation equation 2.2, demonstrating the benefit of tighter bounds (§2.5.1) in complete verification. On the other hand, the poor performance of MIP + $\mathcal{A}_k$ and of Gurobi Planet + Gurobi 1 cut, tied to scaling limitations of off-the-shelf solvers, shows that tighter bounds are effective only if they can be computed efficiently. Nevertheless, the difference in performance between the two Gurobi-based methods confirms that customised Branch and Bound solvers (BaBSR) are preferable to generic MIP solvers, as observed by Bunel et al. (2020b) on the looser Planet relaxation. Moreover, the stratified bounding system allows us to retain the speed of Big-M on easier properties, without excessively sacrificing Active Set's gains on the harder ones. Finally, while ERAN verifies 2% more properties than Active Set on two networks, BaBSR (with any dual bounding algorithm) is faster on most of the properties. BaBSR-based results could be further improved by employing the learned branching strategy presented by Lu and Kumar (2020): in this work, we focused on the bounding component of branch and bound.

2.6 Discussion

The vast majority of neural network bounding algorithms focuses on (solving or loosening) a popular triangle-shaped relaxation, referred to as the "convex barrier"

for verification. Relaxations that are tighter than this convex barrier have been recently introduced, but their complexity hinders applicability. We have presented Active Set, a sparse dual solver for one such relaxation, and empirically demonstrated that it yields significant formal verification speed-ups. Our results show that scalable tightness is key to the efficiency of neural network verification and instrumental in the definition of a more appropriate “convex barrier". We believe that new customised solvers for similarly tight relaxations are a crucial avenue for future research in the area, possibly beyond piecewise-linear networks. Finally, as it is inevitable that tighter bounds will come at a larger computational cost, future verification systems will be required to recognise a priori whether tight bounds are needed for a given property. A possible solution to this problem could rely on learning algorithms.

Acknowledgments

ADP was supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems, grant EP/L015987/1, and an IBM PhD fellowship. HSB was supported using a Tencent studentship through the University of Oxford.

3

Overcoming the Convex Barrier for Simplex Inputs

Contents

Abstract

Recent progress in neural network verification has challenged the notion of a *convex barrier*, that is, an inherent weakness in the convex relaxation of the output of a neural network. Specifically, there now exists a tight relaxation for verifying the robustness of a neural network to ℓ_∞ input perturbations, as well as efficient primal and dual solvers for the relaxation. Buoyed by this success, we consider the problem of developing similar techniques for verifying robustness to input perturbations within the probability simplex. We prove a somewhat surprising result that, in this case, not only can one design a tight relaxation that overcomes the convex barrier, but the size of the relaxation remains linear in the number of neurons, thereby leading to simpler and more efficient algorithms. We establish the scalability of our overall approach via the specification of ℓ_1 robustness for CIFAR-10 and MNIST classification, where our approach improves the state of the art verified accuracy by up to 14.4%. Furthermore, we establish its accuracy on a novel and highly challenging task of verifying the robustness of a multi-modal (text and image) classifier to arbitrary changes in its textual input.

3.1 Introduction

Verification refers to the challenging computational problem of determining whether a neural network satisfies a given specification. Perhaps the most popular specification is to prove or disprove that a neural network classifier is robust to perturbations of the input that lie within an ℓ_p ball (Wong and Kolter, 2018; Zhang et al., 2018; Weng et al., 2018; Mirman et al., 2018; Dvijotham et al., 2018), with impressive state of the art results recently achieved in Wang et al. (2021). The ability to verify neural networks would open the door to better understanding their nature, and allow us to apply deep learning in safety critical domains where errors can have a large cost. Given the importance of the problem, it is unsurprising that it has attracted considerable atttention from the machine learning and automated verification communities (Katz et al., 2017; Ehlers, 2017).

Early work in verification lead to the notion of a *convex barrier*, that is, a limitation in the tightness of the bounds obtainable by the convex relaxation of the output of a neural network (Salman et al., 2019). This weakness was seen as a primary reason for the slow convergence of branch-and-bound algorithms for verification, which rely on convex relaxations to compute the bounds, on a large number of specifications on standard datasets. However, recent work has been able to successfully overcome this barrier. Specifically, Anderson et al. (2018) proposed a tight relaxation that precisely defines the convex hull of a composition of a linear function of a vector within an ℓ_∞ ball with the ReLU non-linearity. While their relaxation has a large number of constraints (exponential in the size of the vector), they provide an efficient algorithm for identifying the most violated constraint at any infeasible point. This enables the use of efficient cutting plane algorithms to solve the primal (Anderson et al., 2018), active sets to solve the dual (de Palma et al., 2021), and even approximate linear bound propagation algorithms (Tjandraatmadja et al., 2020).

Buoyed by the possibility of realizing practical verification using tight relaxations, we consider an important specification, namely, robustness to perturbations that lie in a probability simplex. Examples of such a specification include robustness to ℓ_1

perturbations or to word substitutions (Huang et al., 2019). Previous approaches for addressing this specification relied on fairly loose relaxations based on interval bound propagation (Huang et al., 2019; Gowal et al., 2018a). To alleviate this deficiency, we derive the convex hull of the composition of a linear transformation of the probability simplex with a convex non-linearity such as ReLU or SoftPlus. Somewhat surprisingly, we show that, unlike the previously considered case of ℓ_∞ balls, the probability simplex helps greatly simplify the description of the convex hull. In fact, the number of constraints required is linear in the dimensionality of the simplex. By using a novel technique to propagate the simplex constraints through the hidden layers of the network, we derive a tight relaxation for the output of a network whose size scales linearly with the size of the network. Furthermore, we suitably extend a linear bound propopagation algorithm (Zhang et al., 2018) to solve the tight relaxation efficiently, thereby realizing practical verification over simplex inputs.

We demonstrate the scalability of our approach using the specification of robustness to ℓ_1 perturbations for MNIST and CIFAR-10 classification. Our method achieves 13.6% higher verified accuracy on MNIST and up to 14.4% higher verified accuracy on CIFAR-10 compared to the state of the art baselines on the same networks given the same computational budget.

To further demonstrate the benefits of our tight relaxation, we consider a novel and highly challenging specification of global robustness in multi-modal classification. Specifically, we consider the Food 101 dataset (Wang et al., 2015), where each sample consists of an image of a food item together with its recipe. Our specification requires a neural network trained to classify the food item to be robust to arbitrary changes in the text of a sample. This specification captures the scenario where the image is carefully curated while the text is crowd-sourced and can therefore be easily manipulated by adversarial actors. While similar specifications have been considered in previous works (Jia et al., 2019; Huang et al., 2019), they focus on substituting a very small subset of words with their synonyms, which leads to simpler verification problems. We show that, on our significantly more difficult

global verification task, our method achieves up to 25% higher verified accuracy in the same amount of time as the state of the art baseline.

3.2 Preliminaries

We provide a formal mathematical description of our verification problem, and briefly discuss the existing relaxations in order to contextualise our contributions.

3.2.1 Problem description

We denote the m-dimensional probability simplex as Δ_m, that is, $\Delta_m = \{\boldsymbol{x} \in \mathbb{R}^m \mid \boldsymbol{x} \geq 0, 1^\top \boldsymbol{x} \leq 1\}$. As mentioned in the previous section, we are interested in verifying specifications of a given neural network when its input is constrained to lie in a simplex. We focus on networks with a layered architecture to keep notation simple, but our ideas easily extend to any feed-forward network, including residual networks. We consider a neural network with n layers. Each layer is assumed to be composed of two operations: (i) an affine operation (fully connected layer or convolution layer), which we denote by $\mathbb{L}_k(\cdot)$; and (ii) a non-linear activation function, which we denote by $\sigma(\cdot)$. In other words, given its input $\boldsymbol{x}_{k-1} \in \mathbb{R}^{n_{k-1}}$, the k-th layer performs the operation $\hat{\boldsymbol{x}}_k = \mathbb{L}_k(\boldsymbol{x}_{k-1}) \in \mathbb{R}^{n_k}$, followed by $\boldsymbol{x}_k = \sigma(\hat{\boldsymbol{x}}_k) \in \mathbb{R}^{n_k}$. While we place no restriction on the linear operation, we make the following assumption on the activation function.

Assumption 1. We assume that the activation function σ is an element-wise convex function (for example, ReLU or SoftPlus).

Verification problem: Using the above notation, the verification problem we solve can be formulated as

$$\min_{\boldsymbol{x}} \quad \Psi\left(\hat{\boldsymbol{x}}_n\right) \tag{3.1a}$$

$$s.t. \quad \hat{\boldsymbol{x}}_k = \mathbb{L}_k\left(\boldsymbol{x}_{k-1}\right), \boldsymbol{x}_k = \sigma(\hat{\boldsymbol{x}}_k) \qquad k \in [n], \tag{3.1b}$$

$$\boldsymbol{x}_0 \in \Delta_{n_0}, \tag{3.1c}$$

where $[n]$ denotes the set $\{1, \cdots, n\}$. The objective function Ψ is a scalar-valued linear function of the output of the final layer of the the network. For example, say we wish to verify that a classification network is not vulnerable to any adversarial attacks. In this case, we define Ψ as the difference between the true logit and a target logit outputted by the network. The sign of the optimum value of problem (3.1) can be used to either prove or disprove the aforementioned specification.

We illustrate the practical importance of considering simplex inputs using two examples. These examples will also form the basis of our experimental setup.

ℓ_1 **perturbations:** Consider a network with a continuous valued input $\boldsymbol{x} \in \mathbb{R}^m$. We are interested in verifying the behavior of the network under input perturbations that lie within an ℓ_1 ball: $\{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^0\|_1 \leq \epsilon\}$. This input domain can be reformulated as a simplex as

$$\boldsymbol{x} = \boldsymbol{x}^0 + \epsilon M \boldsymbol{z},\ \boldsymbol{z} \in \Delta_{2m},\ \underset{(m \times 2m)}{M} = \begin{pmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 \end{pmatrix}. \tag{3.2}$$

Bag of words models: Consider a text classification network that takes text as input and makes predictions based on an embedding of the text. A commonly used embedding is the so-called "bag of words". Here, we first take an embedding for every word in the text (for example, using a precomputed set like GloVe (Pennington et al., 2014) or Word2Vec (Mikolov et al., 2013)). Next, we take the mean of all the word embeddings to obtain the final representation. We denote the word embeddings as a matrix $E \in \mathbb{R}^{d \times v}$, where each d-dimensional column represents an embedding of a putative word in a vocabulary of size v. Using the above matrix, the text embedding is given by $E\boldsymbol{x}$ where $\boldsymbol{x}$ represents the normalized counts of words from the vocabulary in the text. In other words, $\boldsymbol{x} \in \Delta_v$ assuming that arbitrarily long text with arbitrary numbers of repetitions of each word in the vocabulary are allowed.

3.2.2 Planet and disjunctive relaxations

Ehlers (2017) proposed a convex relaxation for the ReLU activation function, which is commonly referred to as Planet in the literature. The Planet relaxation has been

widely used in many verification algorithms (Bunel et al., 2020b; Dvijotham et al., 2018; Bunel et al., 2020a; Lu and Kumar, 2020). Briefly, it relaxes the sequence of two operations: $\hat{x} = \boldsymbol{w}^T\boldsymbol{x} + b$ where $\boldsymbol{x} \in \mathbb{R}^m$, followed by $y = \text{ReLU}(\hat{x})$. To this end, it utilizes lower and upper bounds on $\hat{x}$. In our case, since $\boldsymbol{x} \in \Delta_m$, we can compute the bounds as $\tilde{\mathbf{l}} = w_{\min} + b \leq \boldsymbol{w}^T\boldsymbol{x} + b \leq w_{\max} + b = \tilde{\mathbf{u}}$. Here, $w_{\min}$ and $w_{\max}$ denote the minimum and maximum entries of $\boldsymbol{w}$ respectively. Using the bounds on $\hat{x}$, the Planet relaxation for the set $\mathcal{S} = \{y, \boldsymbol{x} \mid y = \text{ReLU}(\boldsymbol{w}^T\boldsymbol{x} + b), \boldsymbol{x} \in \Delta_m\}$ is defined as

$$\mathcal{P}_\Delta : y \geq \boldsymbol{w}^T\boldsymbol{x} + b,\ y \geq 0,\ y \leq \frac{\tilde{\mathbf{u}}}{\tilde{\mathbf{u}} - \tilde{\mathbf{l}}}\left(\boldsymbol{w}^T\boldsymbol{x} + b - \tilde{\mathbf{l}}\right),\ \boldsymbol{x} \in \Delta_m. \tag{3.3}$$

In Anderson et al. (2019b, 2020), the authors propose a tighter relaxation and characterize the exact convex hull of the set

$$\{(x, \text{relu}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)) : \ell \leq x \leq u\}.$$

However, this characterization involves exponentially many inequalities. And efficient algorithms based on it need to resort to cutting plane methods, adding violated inequalities sequentially. Doing so is computationally challenging and requires significant effort to implement in a scalable manner. Further, this does not handle the simplex constraint on the input. Thus using this relaxation would require replacing the simplex constraint with the unit hypercube, a much weaker constraint on the inputs.

3.3 A Concise Convex Relaxation

In this section, we derive our novel tight convex relaxation for problem (3.1). In order to make the exposition clearer, we assume that we have access to simplex constraints for all $\boldsymbol{x}_k, k \in [n-1]$. As will be seen in the next section, such constraints can be derived by propagating the simplex constraint on the input $\boldsymbol{x}_0$ through the network.

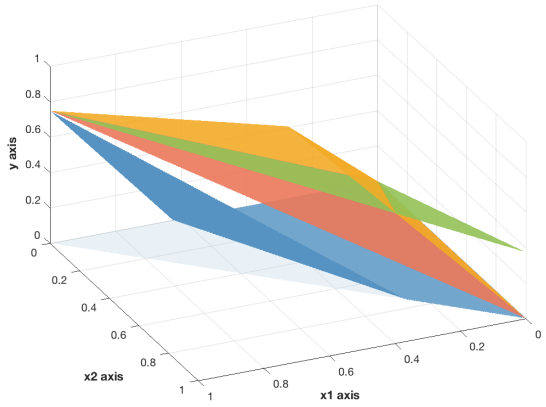


Figure 3.1: The input domain $\boldsymbol{x} \in \Delta_2$ is shown in light blue, while the output function $y = \text{ReLU}(\boldsymbol{w}^T\boldsymbol{x}+b)$ is shown in blue. It can be seen that the upper bound corresponding to the Planet relaxation (shown in green), is significantly looser in comparison to the upper bound corresponding to the proposed relaxation (shown in orange). Even the Anderson relaxation (shown in yellow) is much looser than our relaxation.

3.3.1 An exact convex relaxation for a single neuron

We begin by considering the simple case of a single neuron, which will form the building block of our final relaxation. Similar to the previous relaxations, we consider the set $\mathcal{S} = \{y, \boldsymbol{x} \mid y = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b), \boldsymbol{x} \in \Delta_m\}$, where σ is a convex activation function, for example ReLU. We aim to characterize the convex hull $\mathcal{CH}_\Delta$ of the set $\mathcal{S}$.

Theorem 2. *The convex hull $\mathcal{CH}_\Delta$ of $\mathcal{S}$ is defined by the following convex constraints*

$$y \geq \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right), \quad \boldsymbol{x} \in \Delta_m, \tag{3.4a}$$

$$y \leq \sum_i x_i \left(\sigma\left(\boldsymbol{w}^T\boldsymbol{e}^i + b\right) - \sigma\left(b\right)\right) + \sigma\left(b\right), \tag{3.4b}$$

where $\boldsymbol{e}^i \in \mathbb{R}^m, e^i_i = 1, e^i_j = 0\,\forall j \neq i$, denotes the i-th coordinate vector in $\mathbb{R}^m$.

By the definition of a convex hull, the proposed relaxation is the tightest possible relaxation for a single neuron with $\boldsymbol{x} \in \Delta_m$. Note that, when σ is the ReLU activation function, the only difference between the proposed relaxation and the Planet relaxation $\mathcal{P}_\Delta$ is the upper bound on y. Figure 3.1 compares the upper bound of the Planet relaxation $\mathcal{P}_\Delta$ with the proposed relaxation $\mathcal{CH}_\Delta$ for the case where $m = 2$. As can be seen, our relaxation is significantly tighter than Planet, which paves the way for tractable verification over simplex inputs. The

following proposition formally characterizes the difference between the tightness of $\mathcal{CH}_\Delta$ and $\mathcal{P}_\Delta$.

Proposition 3. *For any input dimension m, $\mathcal{CH}_\Delta$ is provably tighter than $\mathcal{P}_\Delta$. Specifically, we can characterize the gap between $\mathcal{CH}_\Delta$ and $\mathcal{P}_\Delta$ by the gap in the optimal value of the problem*

$$\max_{\boldsymbol{x}} y - y' \text{ subject to } (y, \boldsymbol{x}) \in \mathcal{P}_\Delta, (y', \boldsymbol{x}) \in \mathcal{CH}_\Delta.$$

The gap in the optimal value is proportional to the variance in the weight vector $\boldsymbol{w}$ (see supplementary material for details).

The proofs of the above theorem and proposition are provided in the supplementary material. The proof of Theorem 2 utilizes a fundamental result from convex analysis, that any linear function obtains the same maximum value over $\mathcal{S}$ as over its convex hull $\mathcal{CH}_\Delta$.

Note that our relaxation overcomes the convex barrier by providing bounds that are significantly tighter than the Planet relaxation. Convex barrier is a term introduced in Salman et al. (2019), which is defined as the gap between the optimal value of the original verification problem and the optimal convex relaxation of the non-linearity.

Comparison to Tjandraatmadja et al. (2020) Our relaxation requires only a linear number of inequalities to describe the convex hull for the composition of a linear function with a convex activation function for simplex inputs. In contrast, the Anderson relaxation (Tjandraatmadja et al., 2020; Anderson et al., 2020, 2019a) requires an exponential number of constraints. The submodularity based proof for deriving the relaxation from Tjandraatmadja et al. (2020) can help us understand the discrepancy. The method makes use of the Kuhn triangulation of $[0, 1]^n$ (Todd, 1976), which describes the collection of simplices whose union is $[0, 1]^n$, and requires an exponential number of simplices. Since a unique affine interpolation of the function needs to be constructed on each of these simplices, it requires an overall exponential number of inequalities. In contrast, our relaxation only requires a linear

number of inequalities. Note that the input simplex Δ_n is not one of the simplices from the Kuhn triangulation whose union is the unit hypercube. We provide a visualization for this intuition in the supplementary material.

3.3.2 Final relaxation

Using the convex hull for a single neuron, we can now describe our overall relaxation. Recall that we have assumed $\boldsymbol{x}_k \in \Delta_{n_k}$ for each $k \in [n-1]$. This is ensured through a simplex propagation algorithm described in the next section. In addition, we also compute interval bounds on the pre-activations $\hat{\boldsymbol{x}}_k \in [\ell_k, u_k]$. Using the interval bounds, we define $\theta_k = \frac{u_k}{u_k - \ell_k}, \gamma_k = -\ell_k$, $\mathbf{u}_k(\boldsymbol{x}) = \theta_k \odot (\mathbb{L}_k(\boldsymbol{x}) - \gamma_k)$ and $\mathbf{u}'_k(\boldsymbol{x}) = \sum_i \boldsymbol{x}_i (\sigma(\mathbb{L}_k(\boldsymbol{e}^i)) - \sigma(\mathbb{L}_k(\mathbf{0}))) + \sigma(\mathbb{L}_k(\mathbf{0}))$. Furthermore, since $\hat{\boldsymbol{x}}_n$ is itself a linear function of $\boldsymbol{x}_{n-1}$, with a slight overload of notation, we denote the objective function of our relaxation as $\Psi(\boldsymbol{x}_{n-1})$. Using the above notation, the overall proposed convex relaxation of problem (3.1) for the ReLU activation function can be written as

$$\begin{aligned}
\min_{\boldsymbol{x}} \quad & \Psi(\boldsymbol{x}_{n-1}) && && \text{(3.5a)}\\
s.t. \quad & \boldsymbol{x}_k \geq \mathbb{L}_k(\boldsymbol{x}_{k-1}), \boldsymbol{x}_k \geq 0 && k \in [n-1], \text{ (Planet lower bound)} && \text{(3.5b)}\\
& \boldsymbol{x}_k \leq \mathbf{u}_k(\boldsymbol{x}_{k-1}), && k \in [n-1], \text{ (Planet upper bound)} && \text{(3.5c)}\\
& \boldsymbol{x}_k \leq \mathbf{u}'_k(\boldsymbol{x}_{k-1}), && k \in [n-1], \quad \text{(from equation 3.4b)} && \text{(3.5d)}\\
& \boldsymbol{x}_k \in \Delta && k \in \{0\} \cup [n-1]. && \text{(3.5e)}
\end{aligned}$$

Similar relaxations can be derived for other convex activation functions by linearizing the convex function around a given point. However, we focus on the ReLU case for brevity of exposition, particularly since ReLU is the most commonly used convex activation function.

3.4 Algorithm

Our verification algorithm is composed of two phases: (i) propagating simplex constraints on the activations at every layer; and (ii) computing a lower bound on problem (3.1). We describe these in the following subsections.

3.4.1 Simplex propagation

We assume that the output of each layer is non-negative. If this is not the case, we can simply add a constant to the activation of each layer so that the output becomes non-negative. Let us denote $h_k(\boldsymbol{x}) = \sigma\left(\mathbb{L}_k\left(\boldsymbol{x}\right)\right)$. Since $h_k(.)$ is an element-wise convex function of its inputs, $\mathbf{1}^T h_k(.)$ is also a convex function. Using the fact that the maximum of a convex function over the simplex is attained at one of the vertices, it can be shown that the following inequality holds true

$$\sum_i x_{ki} \leq \max_{j \in \{0,\ldots,n_{k-1}\}} \mathbf{1}^T h_k\left(\boldsymbol{e}^j\right) = \alpha_k. \tag{3.6}$$

Here, x_{ki} denotes the i-th coordinate of the vector of activations x_k at the output of layer h_k.

Note that the above inequality can be rewritten in the form $\sum_i \tilde{x}_{ki} \leq 1$, where $\tilde{x}_{ki} = \frac{x_{ki}}{\alpha_k}$, so that we can propagate simplex-like constraint simply by rescaling activations at the layer appropriately. Details for conditioning the intermediate layers into simplex using inequalities of the above form are provided in the supplementary material.

3.4.2 Efficient solver

The optimization problem (3.5) is a Linear Program and as such can be solved easily by off-the-shelf solvers (Gurobi Optimization, 2020). However, given the size of modern deep networks, such an approach would struggle to scale. Hence, researchers have recently started developing scalable custom solvers for relaxations of neural network outputs.

Zhang et al. (2018); Singh et al. (2019b) developed efficient solvers for relaxations that rely on bounds on the output of a neuron that are expressed as linear functions of the input. Such solvers have been scaled to be extremely efficient and amenable to use within branch and bound frameworks (Wang et al., 2021). Inspired by their success, we extend their capabilities to solve our novel tight relaxations derived in the previous section. To this end, we first relax problem (3.5) to a form that can be solved efficiently using a single backward pass. Concretely, we combine the

lower bounds (3.5b) and the non-negativity constraint from (3.5e) into a single lower bound using weighting coefficients $\underline{\boldsymbol{a}}_k$. We also combine the upper bounds (3.5c and 3.5d) into a single upper bound with weighting coefficients $\bar{\boldsymbol{a}_k}$. The values of the weighting coefficients $\underline{\boldsymbol{a}}_k$ and $\bar{\boldsymbol{a}_k}$ are constrained to lie between 0 and 1. This leads to the following optimization problem

$$
\begin{aligned}
L(\underline{\boldsymbol{a}}, \overline{\boldsymbol{a}}) = \min_{\boldsymbol{x}} \quad & \Psi\left(\boldsymbol{x}_{n-1}\right) && \text{(3.7a)}\\
s.t. \quad & \boldsymbol{x}_0 \in \Delta\,, && \text{(3.7b)}\\
& \boldsymbol{x}_k \geq \underline{\boldsymbol{a}}_k \odot \mathbb{L}_k\left(\boldsymbol{x}_{k-1}\right) && k \in [n-1], \quad \text{(3.7c)}\\
& \boldsymbol{x}_k \leq \overline{\boldsymbol{a}}_k \odot \mathbf{u}_k\left(\boldsymbol{x}_{k-1}\right) + \left(1 - \overline{\boldsymbol{a}}_k\right) \odot \mathbf{u}'_k\left(\boldsymbol{x}_{k-1}\right) && k \in [n-1]. \quad \text{(3.7d)}
\end{aligned}
$$

It turns out that the above formulation can be solved efficiently by a single backward pass over the network. In order to derive the solution, it is useful to introduce the notion of decomposition of an affine function into monotone, anti-monotone and constant parts.

Definition 1. For any scalar-valued affine function $f\left(\boldsymbol{x}\right)$, define

$$
f^+\left(\boldsymbol{x}\right) = \left(\max\left(\frac{\partial f}{\partial \boldsymbol{x}}, 0\right)\right)^T \boldsymbol{x}, f^-\left(\boldsymbol{x}\right) = \left(\min\left(\frac{\partial f}{\partial \boldsymbol{x}}, 0\right)\right)^T \boldsymbol{x}, f^c = f\left(0\right), \quad \text{(3.8)}
$$

so that $f\left(\boldsymbol{x}\right) = f^+\left(\boldsymbol{x}\right) + f^-\left(\boldsymbol{x}\right) + f^c$.

Since f^+ only involves the positive coefficients in the gradient of f, it is monotonically increasing, and similarly, f^- is monotonically decreasing. Exploiting this and the fact that Ψ is a scalar valued linear function, we obtain

$$
\begin{aligned}
&\Psi\left(\boldsymbol{x}_{n-1}\right) = \Psi_0 + \Psi^-\left(\boldsymbol{x}_{n-1}\right) + \Psi^+\left(\boldsymbol{x}_{n-1}\right)\\
&\geq \Psi^c + \Psi^-\left(\overline{\boldsymbol{a}}_{n-1} \odot \mathbf{u}_{n-1}\left(\boldsymbol{x}_{n-2}\right) + \left(1 - \overline{\boldsymbol{a}}_{n-1}\right) \odot \mathbf{u}'_{n-1}\left(\boldsymbol{x}_{n-1}\right)\right)\\
&\qquad + \Psi^+\left(\underline{\boldsymbol{a}}_{n-1} \odot \mathbb{L}_{n-1}(\boldsymbol{x}_{n-2})\right). \qquad \text{(3.9)}
\end{aligned}
$$

The lower bound above is an affine function of $\boldsymbol{x}_{n-2}$ and hence leads to a recursive algorithm where we compute lower bounds on the specification expressed as a

function of $\boldsymbol{x}_{k-1}$ given a lower bound on the specification expressed as a linear function of $\boldsymbol{x}_k$.

Therefore, this leads to a recursive algorithm for computing a lower bound on the optimal value of (3.7), that is presented in the Subroutine SIMPLEX_BACKWARD function on line 11 in Algorithm 1. It can in fact be proven that this is the exact optimal value, using an argument similar to Salman et al. (2019), Section 3.

The computation required to express a lower bound on the specification given as a linear function of $\boldsymbol{x}_{k+1}$ to a lower bound expressed as a linear function of $\boldsymbol{x}_k$ is shown on line (14) of Algorithm 1. Note that this computation can be conveniently done in frameworks that support automatic differentiation, by computing the gradient of the expression on the right hand side of line 14 at 0 and its value at 0, which gives the vector of coeffients and the bias term of the affine function of $\boldsymbol{x}_k$. Thus, the cost of implementing line 14 of the algorithm is equal to the cost of performing backpropagation through a single layer of the network.

Denote the projection onto the unit hypercube as $\pi(\boldsymbol{x}) = \min(\max(\boldsymbol{x}, 0), 1)$. Our overall algorithm is presented in Algorithm 1. It consists of two main functions SIMPLEX_BACKWARD and SIMPLEX_VERIFY. The SIMPLEX_BACKWARD algorithm computes $L(\underline{\boldsymbol{a}}, \overline{\boldsymbol{a}})$ for a fixed value of $\underline{\boldsymbol{a}}, \overline{\boldsymbol{a}}$. However, this lower bound is valid for any choice of these parameters. Hence SIMPLEX_VERIFY performs projected gradient ascent on L with respect to $\underline{\boldsymbol{a}}, \overline{\boldsymbol{a}}$ to obtain the tightest possible lower bound on the (equation 3.1). The idea of optimizing the weighting coefficients $\underline{\boldsymbol{a}}, \overline{\boldsymbol{a}}$ to obtain tighter bounds is inspired from the algorithm of Xu et al. (2021), with suitable adaptations to the setting involving our novel relaxation.

Proposition 4. *Algorithm 1 computes a lower bound on the optimal value of (3.1).*

Proof. Follows by recursively applying the lower bound at each layer while defining f_k. □

Computational Complexity: The complexity of SIMPLEX_BACKWARD function 11 for computing bounds for a given value of $\underline{\boldsymbol{a}}^t, \bar{\boldsymbol{a}}^t$, is the same as the cost of two backward passes through the network. This is twice the cost of CROWN (Zhang et al., 2018), or one iteration of auto-lirpa (Xu et al., 2021), which uses a single backward pass.

Algorithm 1 Simplex Verify

function SIMPLEX_VERIFY(Ψ)
  Initialise $\underline{\boldsymbol{a}}$ and $\bar{\boldsymbol{a}}$ with values between $\mathbf{0}$ and $\mathbf{1}$
  for $t \in [\![0, t_{max} - 1]\!]$ **do**
    $L(\underline{\boldsymbol{a}}^t, \overline{\boldsymbol{a}}^t) =$ SIMPLEX_BACKWARD($\Psi, \underline{\boldsymbol{a}}^t, \overline{\boldsymbol{a}}^t$)
    Compute gradients $\frac{dL}{d\underline{\boldsymbol{a}}^t}, \frac{dL}{d\overline{\boldsymbol{a}}^t}$ via backpropagation
    $\underline{\boldsymbol{a}}^{t+1}, \bar{\boldsymbol{a}}^{t+1} \leftarrow$ update gradient ascent (or Adam)
    $\underline{\boldsymbol{a}}^{t+1}, \overline{\boldsymbol{a}}^{t+1} \leftarrow \pi\left(\underline{\boldsymbol{a}}^{t+1}\right), \pi(\overline{\boldsymbol{a}}^{t+1})$ (projection)
  end for
  return $L(\underline{\boldsymbol{a}}^{t+1}, \bar{\boldsymbol{a}}^{t+1})$
end function
function SIMPLEX_BACKWARD($\Psi, \underline{\boldsymbol{a}}, \overline{\boldsymbol{a}}$)
  $f_N \leftarrow \Psi$
  for $k \in [\![n-1, 0]\!]$ **do**
    Set $f_k(\boldsymbol{x}) \leftarrow f^-_{k+1}\left(\overline{\boldsymbol{a}}_k \odot \mathbf{u}_k(\boldsymbol{x}) + (1 - \overline{\boldsymbol{a}}_k) \odot \mathbf{u}'_k(\boldsymbol{x})\right) + f^c_{k+1} + f^+_{k+1}\left(\underline{\boldsymbol{a}}_k \odot \mathbb{L}_k(\boldsymbol{x})\right)$.
  end for
  $L(\underline{\boldsymbol{a}}, \overline{\boldsymbol{a}}) = \min_{\boldsymbol{x}_0 \in \Delta} f_0(\boldsymbol{x}_0)$
  return $L(\underline{\boldsymbol{a}}, \overline{\boldsymbol{a}})$
end function

3.5 Experiments

In this section, we demonstrate the effectiveness of the proposed method on two specifications: (i) robustness to ℓ_1 perturbations for image classifiers (Sec 3.5.1), and (ii) robustness of multi-modal classifiers to text perturbations (Sec 3.5.2).

3.5.1 ℓ_1 robustness verification

Experimental Setup We verify the ℓ_1 robustness of networks from (de Palma et al., 2021; Bunel et al., 2020b) and VNN-COMP (2020). We compute the lower bound on the robustness margin (difference between the ground truth logit and the other logits) using the verification methods. An image is said to be verified if the

lower bound across all possible labels is positive. We evaluate the effectiveness of various methods for incomplete verification on the MNIST (LeCun et al., 2010) and CIFAR-10 (Krizhevsky and Hinton, 2009) datasets. For MNIST, we evaluate on the entire test set, and for CIFAR-10 we evaluate on 1000 random images from the test set (Krizhevsky and Hinton, 2009). The MNIST and CIFAR-10 datasets are widely used in the machine learning community, and are available under the creator's consent and MIT license respectively. For both MNIST and CIFAR-10, we use the model architectures from (de Palma et al., 2021). The models are trained using the SLIDE attack (sparse ℓ_1-descent attack) from Tramer and Boneh (2019) with $\epsilon = 0.3$ for all networks except the VNN-comp big network, which is trained with $\epsilon = 0.05$. We used the publicly available training implementation of (Ding et al., 2019) (see supplementary material for details). The code is made available under the LGPL License online [1]. We also test on the SGD trained CIFAR Wide model from (de Palma et al., 2021). We verify robustness against input perturbations lying in ℓ_1 norm ball with $\epsilon = 0.35$ for the MNIST network, $\epsilon = 0.2$ for the VNN-comp big network and $\epsilon = 0.5$ for all the other CIFAR-10 networks.

Methods We compare against other propagation based solvers and Gurobi. Gurobi baselines employ the commercial black-box solver Gurobi (Gurobi Optimization, 2020). Gurobi solves the problems to optimality, giving the tightest possible bounds for the corresponding relaxations. Gurobi Planet corresponds to solving the Planet relaxation (Ehlers, 2017) of the network. Gurobi Simplex corresponds to solving our relaxation using Gurobi. Both the methods are run on 4 CPU threads on an Intel(R) Core(TM) i7-4960X CPU @ 3.60GHz processor. We also compare against an optimized LiRPA solver for the Planet relaxation (Ehlers, 2017), and refer to it as Opt-Lirpa Planet. The solver remains the same as our solver, with the only difference being that the upper bound corresponding to our relaxation is not present (see supplementary material for details). Note that the intermediate layer bounds in Opt-Lirpa Planet are not jointly optimized. Simplex Verify corresponds to our

[1] `https://github.com/BorealisAI/advertorch`.

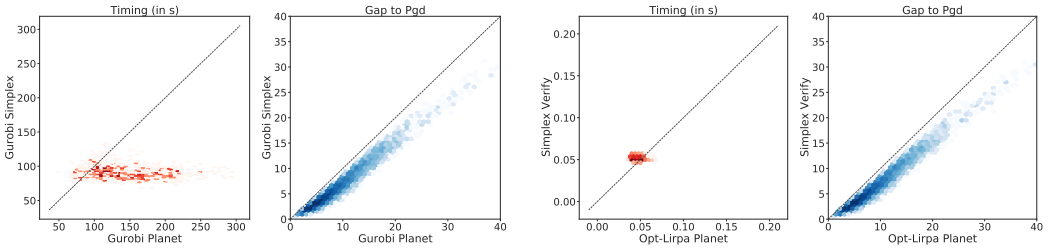
solver described in Sec 3.4.2. Both the LiRPA based solvers use Adam (Kingma and Ba, 2015) for updating the weighting vectors $\boldsymbol{a}$, and are run on a single Nvidia Titan Xp GPU. All the methods use the same intermediate bounds, which are computed using Opt-Lirpa Planet run for 20 iterations. We compare the effectiveness of the different methods for computing the final layer bounds. Further details about the baselines and experimental settings, including the hyper-parameters, are provided in the supplementary material.

Results Table 3.1 shows the verified accuracy and average verification time per sample of different methods. Gurobi Simplex achieves much higher verified accuracy (up to 16.5% higher) in comparison to Gurobi Planet. This is in line with Theorem 2 and Proposition 3 because our proposed relaxation is much tighter than Planet. This also shows the benefit of using tighter relaxations. For a fair comparison, the number of iterations of both LiRPA based methods were tuned such that each of them take the same amount of time. We tuned the number of iterations on a subset of images. It is worth noting that the proposed solver, Simplex Verify, achieves much higher verified accuracy than Opt-Lirpa Planet, in the same amount of time. Simplex Verify also achieves comparable accuracy to Gurobi Simplex, while being nearly 3 orders of magnitudes faster, which shows the effectiveness of the proposed solver in solving Problem 3.5.

We also compare the tightness of the bounds achieved by different methods. We obtained the lower bounds provided by different solvers. We can get the upper bounds using the sparse PGD attack, SLIDE (Tramer and Boneh, 2019). A smaller gap between the PGD upper bound and verified lower bound, indicates tighter verification. Figure 3.2a shows the pointwise comparison of the gap to PGD for Gurobi Planet and Gurobi Simplex, on the same data. PGD gap is much smaller for Gurobi Simplex in comparison to Gurobi Planet, thereby showing that our relaxation achieves much tighter verification. Figure 3.2b shows the pointwise comparison of the gap for Opt-Lirpa Planet and our Simplex Verify. Simplex Verify achieves much tighter verification.

Dataset		MNIST	CIFAR-10					
Model		OVAL Wide	OVAL Base	OVAL Wide	OVAL Deep	OVAL Wide	VNN Med	VNN Big
Training		SLIDE	SLIDE	SLIDE	SLIDE	SGD	SLIDE	SLIDE
Accuracy	Nominal	98.8%	75.1%	79.3%	72.1%	74.4%	81.4%	83.6%
	Pgd	98.2%	73.5%	77.0%	69.8%	73.3%	80.5%	82.3%
Verified Accuracy	Gurobi Planet	31.7%	34.1%	18.4%	11.1%	13.5%	-	-
	Gurobi Simplex	45.2%	48.6%	29.4%	13.4%	23.7%	-	-
	Opt-Lirpa Planet	31.0%	33.6%	17.9%	10.8%	13.5%	48.8%	60.0%
	Simplex Verify	44.6%	48.0%	28.8%	13.4%	22.4%	59.4%	66.4%
Verified Time/Sample	Gurobi Planet	74.61s	22.80s	114.92s	86.84s	114.70s	-	-
	Gurobi Simplex	72.47s	22.95s	72.17s	59.22s	70.42s	-	-
	Opt-Lirpa Planet	0.04s	0.04s	0.04s	0.06s	0.04s	0.05s	0.06s
	Simplex Verify	0.04s	0.04s	0.04s	0.05s	0.04s	0.05s	0.06s

Table 3.1: Verified accuracy and verification time of different solvers on MNIST and CIFAR-10 models. We test on the entire test set for MNIST, and random 1000 test images for CIFAR-10. Simplex Verify denotes our proposed solver. Our proposed method achieves much higher verified accuracy in comparison to the state of the art baseline, in the same amount of time. '-' denotes instances not solved within a 5 minute timeout.



(a) Comparison of runtime (left) and gap of lower bounds to PGD upper bounds (right), for Gurobi Simplex and Gurobi Planet.

(b) Comparison of runtime (left) and gap of lower bounds to PGD upper bounds (right), for Simplex Verify and Opt-Lirpa Planet.

Figure 3.2: Pointwise comparison of the lower bounds on CIFAR-10 test set on the adversarially trained Wide model, for a subset of the methods. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

3.5.2 Multi-modal classifier robustness verification

Experimental Setup In this section, we are interested in verifying the robustness of multi-modal classifiers to text-based attacks. The motivation are scenarios where image based models are augmented with text to improve the accuracy of the classifier. However, this makes the model vulnerable to text based attacks. The aim of this experiment is to show the adversarial vulnerability of such models and verify their robustness. For this experiment we verify the robustness of models on the UPMC FOOD-101 dataset (Wang et al., 2015), which is a commonly used dataset for multi-modal classification (see (Kiela et al., 2019)). This dataset consists of images and recipes of different food items. It is made available by the creators' consent online [2]. The specification that we are interested in verifying is as follows: for a given image and text pair, only the text is perturbed and any possible text from the given vocabulary is allowed in the attack. The aim of this specification is to characterise the worst-case sensitivity of the model. We are not aiming for perfect robustness to the noise in text, but aim to check its sensitivity. We use a ConcatBOW model for this task as proposed in Kiela et al. (2019). The model extracts an image embedding using a standard pretrained ResNet-152 model. It also extract a Bag of words embedding for the text. The model concatenates the

[2] `http://visiir.lip6.fr/`

embeddings and feeds them into a multilayer perceptron (MLP) classifier, which has 1,846,200 parameters. Arbitrary changes in the text can be modelled as a simplex as described in Section 3.2.1. The model follows the same architecture as the state-of-the-art model for this dataset (Ignazio Gallo and Grassa, 2020), except that we replace Bert embeddings with BOW embeddings. We reduce the dataset from 101 classes to 10 classes. More details are provided in the supplementary.

Related Work Huang et al. (2019) considered verification against synonym replacements or character flip perturbations on text classification models. The input specification was modeled as a simplex and they proposed to use Interval bound propagation (IBP) for the same. Concurrent to Huang et al. (2019), Jia et al. (2019) considered a specification where every word in the text can be replaced with a similar word and used IBP. Xu et al. (2020) use more recent LiRPA variants for verification of synonym-based word substitution with a maximum of 6 word substitutions. Concurrent to our work, Bonaert et al. (2021) proposed a new method for certifying Transformers to synonym based attacks. They provide a lifting from ℓ_∞ analysis techniques to other norms, and propose a new convex relaxation for a number of settings, but none of these characterize the convex hull exactly in any setting. In contrast to these works, we consider a much more challenging specification where any possible text from the vocabulary is allowed. This specification includes arbitrary length sentences. Further, even our baseline, Opt-Lirpa Planet, is much tighter than IBP. We also propose a tighter relaxation which characterises the convex hull for our setting of a composition over a convex activation function and an activation function where the input is constrained to be in the simplex, and propose an efficient algorithm for the relaxation. It is also important to note that we perform experiments on verification-agnostic networks unlike Huang et al. (2019).

Results We compare the lower bounds on the robustness margin for Opt-Lirpa Planet and Simplex Verify. We use PGD attack for computing the upper bound. The networks are trained to be robust to simplex perturbations using PGD training. Figure 3.3 shows the pointwise comparison of the gap to PGD upper bounds for

Opt-Lirpa Planet and Simplex Verify. Note here that a smaller gap indicates tighter verification. For a fair comparison, the iterations of the propagation algorithms were again tuned such that each of them takes the same amount of time (0.08s). It can be seen that Simplex Verify achieves much tighter bounds in the same amount of time.

In Table 3.2, we present the verified accuracy achieved by the different methods with the nominal and Pgd accuracy. We present results for networks trained with different weighting for nominal and PGD loss during the adversarial training. Network trained only on images achieves 87.82% accuracy. Note that our method achieves remarkably higher verified accuracy, up to 25% higher, as compared to Opt-Lirpa Planet in the same amount of time. This shows the benefit of our approach over the existing solvers.

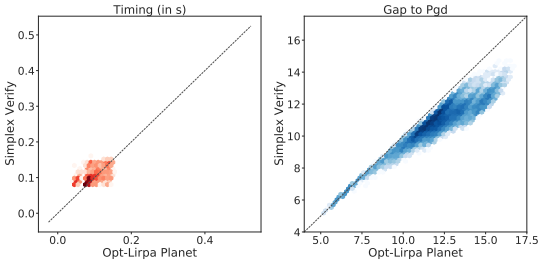


Figure 3.3: Pointwise comparison of runtime (left) and gap of lower bounds to PGD upper bounds (right) for the global specification on Food-101 dataset.

3.6 Discussion and Broader Impact

As machine learning continuously finds new application domains, the robustness of machine learning systems in the face of adversarsarial behavior becomes increasingly relevant. Neural network verification, which seeks to obtain provable guarantees on robustness of neural networks, has mostly focused on ℓ_∞ perturbations. However, real security threats often involve more drastic changes that could lead to arbitrary perturbations in the input. In this paper, we have we have proposed an efficient algorithm for verification of neural networks against simplex constrained perturbations. We have shown the practical importance of

Accuracy	Nominal	86.3%	87.3%	92.3%
	Pgd	84.7%	84.5%	19.2%
Verified Accuracy	Opt-Lirpa Planet	17.9%	08.0%	01.2%
	Simplex Verify	42.2%	32.8%	01.3%

Table 3.2: Verified accuracy achieved by different solvers on the 2164 test images of the reduced Food-101 dataset. Results are presented for networks trained with different weighting for nominal and PGD loss. Our method (Simplex Verify) achieves up to 24% higher verified accuracy, in the same amount of time as the state of the art baseline Opt-Lipra Planet.

this specification through two applications. Firstly, we verify the robustness of image classifiers to ℓ_1 perturbations, which could be an appropriate threat model when perturbations of different features are correlated. Secondly, we consider a challenging specification, whereby any arbitrary text perturbations are allowed on multi-modal (image and text input) classifiers. This highlights the vulnerability of multi-modal classifiers to text attacks. The proposed algorithm improves the state of the art verification accuracy by up to 25%.

Multi-modal data is used in various domains including social media, and the internet in general. We believe that our work could find application in verifying classifiers which label advertisements and social media posts as illegal or hateful. Although we tackle arbitrary perturbation in the text input, our work does not verify against all possible perturbations on the multi-modal input. Another limitation is that we have considered only a particular class of multi-modal architectures. It is also important to verify other commonly used architectures. Further, verification exposes flaws in neural network models, which on the one hand can help improve their robustness, but on the other hand, can be exploited by an adversary.

Acknowledgements and Disclosure of Funding

Harkirat was supported using a Tencent studentship through the University of Oxford. Philip H.S. Torr was supported by the EPSRC grant: Turing AI Fellowship: EP/W002981/1, EPSRC/MURI grant EP/N019474/1 and the Royal Academy of Engineering. We also thank Rudy Bunel for feedback on the draft.

4

AutoSimulate: (Quickly) Learning Synthetic Data Generation

Contents

Abstract

Simulation is increasingly being used for generating large labelled datasets in many machine learning problems. Recent methods have focused on adjusting simulator parameters with the goal of maximising accuracy on a validation task, usually relying on REINFORCE-like gradient estimators. However these approaches are very expensive as they treat the entire data generation, model training, and validation pipeline as a black-box and require multiple costly objective evaluations at each iteration. We propose an efficient alternative for optimal synthetic data generation, based on a novel differentiable approximation of the objective. This allows us to optimize the simulator, which may be non-differentiable, requiring only one objective evaluation at each iteration with a little overhead. We demonstrate on a state-of-the-art photorealistic renderer that the proposed method finds the optimal data distribution faster (up to 50×), with significantly reduced training data generation and better accuracy than previous methods.

4.1 Introduction

Massive amounts of data needs to be collected and labelled for training neural networks for tasks such as object detection Ren et al. (2017); He et al. (2017), segmentation Shelhamer et al. (2017) and machine translation Luong et al. (2015).

A tantalizing alternative to real data for training neural networks has been the use of synthetic data, which provides accurate labels for many computer vision and machine learning tasks such as (dense) optical flow estimation Dosovitskiy et al. (2015); Richter et al. (2016), pose estimation Varol et al. (2017); Doersch and Zisserman (2019); Xiang et al. (2018); Tekin et al. (2018); Kehl et al. (2017), among others Su et al. (2015); Gaidon et al. (2016); Hinterstoisser et al. (2018); Rad and Lepetit (2017); Dwibedi et al. (2017); Ros et al. (2016). Current paradigm for synthetic data generation involves human experts manually handcrafting the distributions over simulator parameters Le et al. (2017); Sakaridis et al. (2018), or randomizing the parameters to synthesize large amounts of data using game engines or photorealistic renderers Dosovitskiy et al. (2015); Richter et al. (2016); Ros et al. (2016). However, photorealistic data generation with these approaches is expensive, needs significant human effort and expertise, and can be sub-optimal. This raises the question, Has the full potential of synthetic data really been utilized?

Recent approaches Louppe and Cranmer (2019); Ganin et al. (2018); Ruiz et al. (2019); Kar et al. (2019) have formulated the setting of simulator parameters as a learning problem. A few of these methods Louppe and Cranmer (2019); Ganin et al. (2018) learn simulator parameters to minimize the distance between distributions of simulated data and real data. Ruiz *et al.* Ruiz et al. (2019) proposed to learn the optimal simulator parameters to directly maximise the accuracy of a model on a defined task. However these approaches Kar et al. (2019); Ruiz et al. (2019) are very expensive, as they treat the entire data generation and model training pipeline (Figure 4.1, outer loop) as a black-box, and use policy gradients Williams (1992), which require multiple expensive objective evaluations at each iteration. As a result, learning synthetic data generation with photorealistic renderers has remained a challenge.

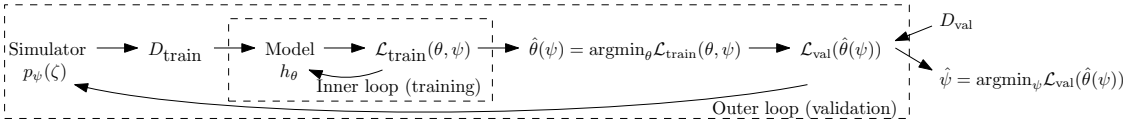


Figure 4.1: Overview of the bilevel optimization setup. A simulator $p_{\boldsymbol{\psi}}(\zeta)$ is used to generate a synthetic dataset D_{train}; **inner loop**: a model $h_{\boldsymbol{\theta}}$ is then trained on this dataset with training loss $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi})$ to obtain optimal model paramaters $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$; a real-data validation set D_{val} is used to evaluate the performance of this trained model with validation loss $\mathcal{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$, providing a measure of goodness of simulator parameter $\boldsymbol{\psi}$; **outer loop**: $\boldsymbol{\psi}$ is updated until we find optimal simulator parameters $\hat{\boldsymbol{\psi}}$.

In this work, we propose a fast optimization algorithm for learning synthetic data generation, which can quickly optimize state-of-the-art photorealistic renderers. We look at the problem of finding optimal simulator parameters as a bi-level optimization problem (Figure 4.1) of training (inner) and validation (outer) iterations, and derive approximations for their corresponding objectives. Our key contribution lies in proposing a novel differentiable approximation of the objective, which allows us to optimize the simulator requiring only one objective evaluation at each iteration, with improved speed and accuracy. We also propose effective numerical techniques to optimize the approximation, which can be used to derive terms depending on desired speed-accuracy tradeoff. The proposed method can be used with non-differentiable simulators and handle very deep neural networks. We demonstrate our method on two renderers, the Clevr data generator Johnson et al. (2017) and the state-of-the-art photorealistic renderer Arnold Georgiev et al. (2018).

4.2 Related Work

Expert Involvement and Random Generation

One of the initial successful work on training deep neural networks on synthetic data for a computer vision problem was done on optical flow estimation, where Dosovitskiy et al. Dosovitskiy et al. (2015) created a large dataset by randomly generating images of chairs using an OpenGL pipeline by pasting objects onto randomly selected real-world images. This strategy has been applied in other problems like object detection, instance segmentation and pose estimation Su et al. (2015); Hinterstoisser et al. (2018); Rad and Lepetit (2017); Tekin et al. (2018);

Dwibedi et al. (2017). Though this approach is simple to implement, the foreground objects are always pasted onto out-of-context background images, thereby requiring careful selection of the background images to achieve good accuracy as shown by Dvornik et al. Dvornik et al. (2018). Other issues with this technique include these images not being realistic, objects not having accurate shading, and shadows being inconsistent with the background.

Another line of work explores generation of photorealistic images with objects rendered within complete 3D scenes Richter et al. (2017, 2016); Hodaň et al. (2019); Ros et al. (2016); Gaidon et al. (2016); Tremblay et al. (2018); Handa et al. (2016); Zhang et al. (2017b). Though this is a well accepted approach for synthetic data generation, it suffers from several issues. First, given the data generation process is independent of the neural network training, these approaches synthesize a large set of redundant training images, as shown in experiments section. This might add a massive redundant burden on the rendering infrastructure. Second, this requires human expert involvement, e.g., to set the right scene properties, material and texture of objects, quality of rendering, among several other simulator parameters Hodaň et al. (2019). This hinders widespread adaptation of synthetic data to different tasks. Finally, some sub-optimal synthesized data can corrupt the neural network training.

Learning Simulator Parameters

In order to resolve these issues, recent research has focused on learning the simulator parameters. The non-differentiability of the simulators has posed a challenge for optimization. Louppe et al. Louppe and Cranmer (2019) proposed an adversarial variational optimization technique for learning parameters of non-differentiable simulators, by minimizing the Jensen–Shannon divergence between the distribution of the synthetic data and distribution of the real data. Ganin et al. Ganin et al. (2018) incorporated a non-differentiable simulator within an adversarial training pipeline for generating realistic synthetic images.

Ruiz et al. Ruiz et al. (2019) focused on optimization of simulator parameters with the objective of generating data that directly maximizes accuracy on downstream tasks such as object detection. They treated the entire pipeline of data generation and neural network training as a black-box, and used classical REINFORCE-based Williams (1992) gradient estimation. However, this approach suffers from scalability issues. A single objective evaluation involves generating a synthetic dataset, training a neural network for multiple epochs, and calculating the validation loss. And this method requires mutiple such expensive objective evaluations for taking a single step. Thus it has a very slow convergence and is difficult to scale to photorealistic simulators which have hundreds of parameters. In contrast, our method AutoSimulate, requires only a single objective evaluation at each iteration and works well with state-of-the-art photorealistic renderers. Making an assumption that a probabilistic grammar is available, Kar et al. Kar et al. (2019) proposed to learn to transform the scene graphs within this probabilistic grammar, with the objective of simulataneously optimizing performance on downstream task and matching the distribution of synthetic images to real images. They also use REINFORCE-based Williams (1992) gradient estimation for the first objective like Ruiz et al. (2019), whose limitations were discussed above.

In bi-level optimization, differentiating through neural network training is a challenge. MacKay et al. (2019) proposed to learn an approximation of inner loop using another network. Concurrent work Yang and Deng (2020) makes an assumption that the neural network is trained only for one or few iterations (not epochs), so they can store the computation graph in memory and back-propagate the derivatives, in a similar spirit as MAML Finn et al. (2017b); Behl et al. (2019). In constrast, we proposed a novel differentiable approximation of the inner loop using a Newton step which can handle many epochs without memory constraints. We also proposed efficient approximations which can be used for desired speed-accuracy tradeoff.

4.3 Problem Formulation

In supervised learning, a training set $D_{\text{train}} = \{z_1, ..., z_m\}$ of input–output pairs $z_i = (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{X} \times \mathbb{Y}$ is used to learn the parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ of a model $h_{\boldsymbol{\theta}}$ that maps the input domain $\mathbb{X}$ to the output codomain $\mathbb{Y}$. This is accomplished by minimizing the empirical risk $\frac{1}{n}\sum_{i=1}^{n} l(z_i, \boldsymbol{\theta})$, where $l(z, \boldsymbol{\theta}) \in \mathbb{R}$ denotes the loss of model $h_{\boldsymbol{\theta}}$ on a data point z.

Our goal is to generate synthetic training data using a simulator such that the model trained on this data minimizes the empirical risk on some real-data validation set D_{val}. The simulator defines a data generating distribution $p_{\boldsymbol{\psi}}(\zeta)$ given simulator parameters $\boldsymbol{\psi} \in \mathbb{R}^m$, from which we can sample training data instances $\zeta \sim p_{\boldsymbol{\psi}}(\zeta)$, where we use ζ to denote simulated data as opposed to real data z. The objective of finding optimal simulator parameters $\hat{\boldsymbol{\psi}}$ can then be formulated as the optimization problem

$$\min_{\boldsymbol{\psi}} \quad \mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big) \tag{4.1a}$$

$$s.t. \quad \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}) \in \arg\min_{\boldsymbol{\theta}} \mathbb{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}) \ , \tag{4.1b}$$

where $\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big) = \sum_{z_i \in D_{\text{val}}} l\Big(z_i, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big)$ is the validation loss, $\mathbb{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}) = \mathbb{E}_{\zeta \sim p_{\boldsymbol{\psi}}}\Big[l(\zeta, \boldsymbol{\theta})\Big]$ is the training loss, $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$ denote the optimum of model parameters after training on data generated from the simulator parameterised by $\boldsymbol{\psi}$, and $\hat{\boldsymbol{\psi}}$ denote the optimum simulator parameters that minimize $\mathbb{L}_{\text{val}}$. In this paper we will refer to Equations 4.1a and 4.1b as the outer and inner optimization problems respectively. This formulation is illustrated in Figure 4.1.

Equations 4.1a and 4.1b represent a bi-level optimization problem Colson et al. (2007); Franceschi et al. (2018); Bennett et al. (2008), which is a special kind of optimization where one problem is nested within another. To compute the gradient of the objective $\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big)$ with respect to $\boldsymbol{\psi}$, one needs to propagate derivatives through the training of a model and data generation from a simulator, which is often impossible due to the simulator being non-differentiable Louppe and Cranmer (2019). Even in the case of a differentiable simulator, backpropagating through

entire training sessions is impracticable because it requires keeping a large number of intermediate variables in memory Maclaurin et al. (2015). One technique to address this challenge is to treat the entire system as a black-box and use off-the-shelf hyper-parameter optimization algorithms such as REINFORCE Williams (1992), evolutionary algorithms Mitchell (1998) or Bayesian optimization Snoek et al. (2012), which require multiple costly evaluations of the objective in each iteration. An important distinction from neural network hyper-parameter optimization is that evaluating the objective $\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big)$ at a given $\boldsymbol{\psi}$ is much more expensive in our setting because it involves the expensive step of running the simulation for synthetic dataset generation along with neural network training.

In this paper we propose an efficient technique based on locally approximating the objective function $\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big)$ at a point $\boldsymbol{\psi}$, together with an effective numerical procedure to optimize this local model, enabling the efficient tuning of simulator parameters in state-of-the-art computer vision workflows.

4.4 AutoSimulate

We will derive differentiable approximations of the outer and inner optimization problems (Figure 4.1) using Taylor expansions of the objectives $\mathbb{L}_{\text{val}}$ and $\mathbb{L}_{\text{train}}$.

Outer Problem Our goal is to find $\hat{\boldsymbol{\psi}}$, the optimal simulator parameters which minimise $\mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$ in the outer (validation) problem, so we construct a Taylor expansion of $\mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$ around $\boldsymbol{\psi}_t$ at iteration t as

$$\begin{aligned}\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})\Big) &= \mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) + \Delta\boldsymbol{\psi}\frac{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)}{d\boldsymbol{\psi}}\frac{d\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big)}{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)} + ... \\ &= \mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) + \Delta\hat{\boldsymbol{\theta}}_{\boldsymbol{\psi}}\frac{d\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big)}{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)} + ... \,, \end{aligned} \tag{4.2}$$

where $\Delta\hat{\boldsymbol{\theta}}_{\boldsymbol{\psi}} = \Delta\boldsymbol{\psi}\frac{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)}{d\boldsymbol{\psi}} \approx \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + d\boldsymbol{\psi}) - \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)$.

Inner Problem To obtain parameter update $\Delta\hat{\boldsymbol{\theta}}_{\boldsymbol{\psi}}$ for the inner (training) problem, which requires retraining on the dataset generated with the new simulator

parameter $\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}$, we write the loss function $\mathbb{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})$ as its Taylor series approximation around the current $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$ as

$$\begin{aligned}\mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t) + \Delta\boldsymbol{\theta}, \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big) = \mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big) \\ + \Delta\boldsymbol{\theta}^\top \frac{\partial}{\partial\boldsymbol{\theta}} \mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big) \\ + \frac{1}{2}\Delta\boldsymbol{\theta}^\top \boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)\Delta\boldsymbol{\theta} + \dots ,\end{aligned} \tag{4.3}$$

where the Hessian $\boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big) \stackrel{\text{def}}{=} \frac{\partial^2}{\partial\boldsymbol{\theta}^2}\mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big) \in \mathbb{R}^{n\times n}$.

We are interested in our local model in the limit $\Delta\boldsymbol{\psi} \to 0$, implying that our initial point $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)$ will be in close vicinity to the optimal $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})$. Thus we utilize the local convergence of the Newton method Nocedal and Wright (2006) and approximate $\mathbb{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})$ by the quadratic portion. Assuming the $\boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)$ is positive definite, and minimizing the quadratic portion with respect to $\Delta\boldsymbol{\theta}$, we get

$$\begin{aligned}\Delta\hat{\boldsymbol{\theta}}_\psi &\approx \arg\min_{\Delta\boldsymbol{\theta}} \left(\Delta\boldsymbol{\theta}^\top \frac{\partial\mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)}{\partial\boldsymbol{\theta}} + \frac{1}{2}\Delta\boldsymbol{\theta}^\top \boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)\Delta\boldsymbol{\theta}\right) \\ &= -\boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)^{-1} \frac{\partial\mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)}{\partial\boldsymbol{\theta}} .\end{aligned} \tag{4.4}$$

Differentiable Approximation Putting this back into Equation 4.2, and ignoring higher-order terms in $\Delta\boldsymbol{\theta}$, we get the approximation for our objective as

$$\begin{aligned}\widetilde{\mathbb{L}}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})\Big) = \mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) - \\ \frac{\partial\mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)}{\partial\boldsymbol{\theta}}^\top \boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)^{-1} \frac{d\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big)}{d\boldsymbol{\theta}} ,\end{aligned} \tag{4.5}$$

which is equivalent to writing

$$\widetilde{\mathbb{L}}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big) = \mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) - \frac{\partial\mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\Big)}{\partial\boldsymbol{\theta}}^\top \boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\Big)^{-1} \frac{d\mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big)}{d\boldsymbol{\theta}}, \tag{4.6}$$

and is our local approximation of the objective. A visualization of this approximation is provided in Figure 4.2.

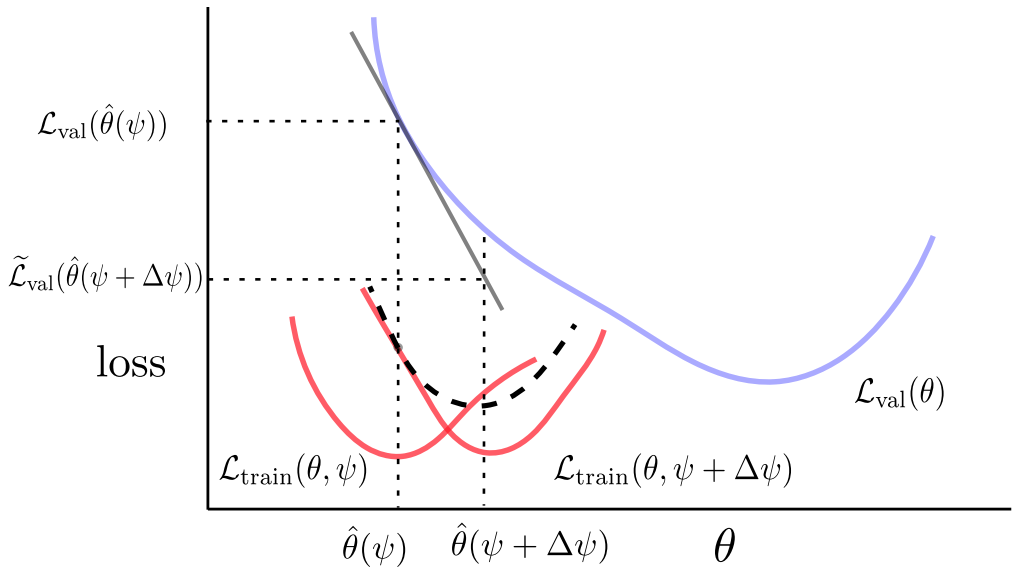


Figure 4.2: Visualization of proposed differentiable approximation of objective $\mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$ (blue). Red curves show the loss surface $\mathbb{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi})$ for the current training data and the loss surface $\mathbb{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi} + \Delta\boldsymbol{\psi})$ for data after a small update $\Delta\boldsymbol{\psi}$ in the simulator parameters. We assume $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi} + \Delta\boldsymbol{\psi})$ to be close to $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$, and use a single step of Newton's method to update $\boldsymbol{\theta}$. This gives us an approximation for updates in optimal $\hat{\boldsymbol{\theta}}$. We then use these to construct an approximation (black) for updates in optimal $\boldsymbol{\psi}$.

We propose to optimize our local model using gradient descent, and its derivative at point $\boldsymbol{\psi}_t$ (simulator parameter at iteration t) can be written as

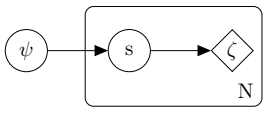
$$\frac{\partial \widetilde{\mathbb{L}}_{\text{val}}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\right)}{\partial \boldsymbol{\psi}}\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} = -\frac{\partial}{\partial \boldsymbol{\psi}}\left[\frac{\partial \mathbb{L}_{\text{train}}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\right)}{\partial \boldsymbol{\theta}}^{\top} \boldsymbol{H}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\right)^{-1} \frac{d\mathbb{L}_{\text{val}}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\right)}{d\boldsymbol{\theta}}\right]\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}. \tag{4.7}$$

Using the definition of $\mathbb{L}_{\text{train}}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\right)$ and ignoring the higher order derivative $\frac{\partial}{\partial \boldsymbol{\psi}} H\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\right)$, we get

$$\frac{\partial \widetilde{\mathbb{L}}_{\text{val}}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\right)}{\partial \boldsymbol{\psi}}\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} = -\frac{\partial}{\partial \boldsymbol{\psi}} \mathbb{E}_{\zeta \sim p_{\psi}}\left[\frac{\partial}{\partial \boldsymbol{\theta}} l\left(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\right)\right]^{\top}\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \boldsymbol{H}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t\right)^{-1} \frac{d}{d\boldsymbol{\theta}} \mathbb{L}_{\text{val}}\left(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\right). \tag{4.8}$$

Next we show how to approximate the term $\frac{\partial}{\partial \boldsymbol{\psi}} \mathbb{E}_{\zeta \sim p_{\psi}}\left[\frac{\partial}{\partial \boldsymbol{\theta}} l\left(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\right)\right] \in \mathbb{R}^{m \times n}$ which requires backpropagation through the dataset generation using a stochastic simulator.

4.4.1 Stochastic Simulator (Data Generating Distribution)



We assume a stochastic simulator that involves a deterministic renderer, which may be non-differentiable, and we make the stochasticity in the process explicit by separating the stochastic part of the simulator from the deterministic rendering. Given the deterministic renderer component $\zeta = \mathrm{r}(s)$, we would like to find the optimal values of simulator parameters $\boldsymbol{\psi}$ that parameterize $s \sim q_{\boldsymbol{\psi}}(s)$ representing the stochastic component, expressing the overall simulator as $\zeta \sim p_{\boldsymbol{\psi}}(\zeta)$. Thus we can write

$$p_{\boldsymbol{\psi}}(\zeta) = \int_{s \in \{s | r(s) = \zeta\}} q_{\boldsymbol{\psi}}(s) ds \ . \tag{4.9}$$

For example, lets say we want to optimize the location of an object in a scene. Then ψ could be the parameters of a Gaussian distribution $q_\psi(.)$ that is used to sample the location of the object in world coordinates, and s denotes the location of the object sampled as $s \sim q_\psi(s)$. Now this sampled location s is given as input to the renderer to generate an image ζ as $\zeta = \mathrm{r}(s)$. The overall simulator, including the stochastic sampling and the deterministic renderer, thus samples the images ζ as $\zeta \sim p_\psi(\zeta)$, where $p_\psi(\zeta)$ denotes the distribution over the images, parameterized by ψ. Therefore we get

$$\frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{\zeta \sim p_{\boldsymbol{\psi}}} \Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\Big(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) \Big] = \frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{s \sim q_{\boldsymbol{\psi}}} \Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\Big(r(s), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) \Big]. \tag{4.10}$$

The gradient of expectation term for continuous distributions can be computed using REINFORCE Williams (1992) as

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{s \sim q_{\boldsymbol{\psi}}} \Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\Big(r(s), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) \Big] &= \mathop{\mathbb{E}}_{s \sim q_{\boldsymbol{\psi}}} \Big[\frac{d}{d\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(s) . \Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\Big(r(s), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) \Big]^\top \Big] \\ &\approx \sum_{k=1}^{K} \frac{d}{d\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(s_k) . \Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\Big(r(s_k), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) \Big]^\top, \end{aligned} \tag{4.11}$$

where s_k denotes samples drawn from the distribution $q_{\boldsymbol{\psi}}$. This can be derived similarly for discrete distributions Schulman et al. (2015); Rezende et al. (2014), and we provide a derivation in the supplementary material. Therefore we can write the update rule as

$$\boldsymbol{\psi}_{t+1} \leftarrow \boldsymbol{\psi}_t + \alpha \frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{\zeta \sim p_{\boldsymbol{\psi}}} \Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\Big(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big) \Big]^\top \Big|_{\boldsymbol{\psi} = \boldsymbol{\psi}_t} \boldsymbol{H}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t\Big)^{-1} \frac{d}{d\boldsymbol{\theta}} \mathbb{L}_{\mathrm{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big). \tag{4.12}$$

Algorithm 2 AutoSimulate

for number of iterations **do**
Sample dataset of size K: $D_{\text{train}} \sim p_{\boldsymbol{\psi_t}}(\zeta)$
Fine-tune model for ϵ epochs on D_{train}
Compute $\boldsymbol{H}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t)^{-1} \frac{d}{d\boldsymbol{\theta}} \mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))$ using CG
Compute gradient of expectation as $\sum_{k=1}^{K} \frac{d}{d\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(s_k) . \left[\frac{\partial}{\partial \boldsymbol{\theta}} l\big(r(s_k), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\right]^{\top}$
Update simulator by descending the gradient $-\frac{\partial}{\partial \boldsymbol{\psi}} \mathbb{E}_{\zeta \sim p_{\psi}} \left[\frac{\partial}{\partial \boldsymbol{\theta}} l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))\right]^{\top} \Big|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \boldsymbol{H}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t)^{-1} \frac{d}{d\boldsymbol{\theta}} \mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))$
end for

It can be seen that we have transformed our original bi-level objective in Equation 4.1a, into iteratively creating a local model $\widetilde{\mathbb{L}}_{\text{val}}(\hat{\boldsymbol{\theta}}\Big(\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\Big)\Big)$ and minimizing it with respect to $\Delta\boldsymbol{\psi}$.

4.4.2 Efficient Numerical Computation

The benefit of the proposed approximation is that it enables us to use techniques from unconstrained optimization. The update rule in Equation 4.12 requires an inverse Hessian computation at each iteration, which is common in second-order optimization. We now discuss an efficient strategy for optimizing our model.

Regularization for Hessian The first challenge is that the Hessian might have negative eigenvalues. Thus the inverse of the Hessian may not exist. We regularize the Hessian using the Levenberg method Moré (1978) and use $\boldsymbol{H} + \lambda \boldsymbol{I}$, where λ is the regularization constant and $\boldsymbol{I}$ denotes the identity matrix. This is common in second-order optimization of Neural Networks Botev et al. (2019); Henriques et al. (2019); Martens and Grosse (2015).

Inverse Hessian–vector product computation Secondly, to compute the update term in Equation 4.12, we split it as follows: we first compute $\boldsymbol{v}_{\boldsymbol{\psi}_t} \stackrel{\text{def}}{=} \boldsymbol{H}^{-1}_{\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)} \boldsymbol{g}_{\text{val}}$ and then compute $\frac{\partial}{\partial \boldsymbol{\psi}} \widetilde{\mathbb{L}}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\Big)\Big|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} = -\boldsymbol{v}_{\boldsymbol{\psi}_t} . \nabla_{\boldsymbol{\psi}} \boldsymbol{g}_{\text{train}}$, where $\boldsymbol{g}_{\text{val}} \stackrel{\text{def}}{=} \frac{d}{d\boldsymbol{\theta}} \mathbb{L}_{\text{val}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\Big)$ and $\boldsymbol{g}_{\text{train}} \stackrel{\text{def}}{=} \frac{d}{d\boldsymbol{\theta}} \mathbb{L}_{\text{train}}\Big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\Big)$.

The inverse Hessian vector product $\boldsymbol{H}^{-1}\boldsymbol{g}$ can be computed by solving problem $\min_{\boldsymbol{v}}\{\frac{1}{2}\boldsymbol{v}^{\top}\boldsymbol{H}\boldsymbol{v} - \boldsymbol{g}^{\top}\boldsymbol{v}\}$ using the conjugate gradient method Shewchuk (1994). This

	Approximation ($\Delta\hat{\theta}_\psi$)	Derivative Term ($\frac{\partial}{\partial \boldsymbol{\psi}}\widetilde{\mathbb{L}}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$)
Quadratic	$-H(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})^{-1}\frac{\partial}{\partial \boldsymbol{\theta}}\mathbb{L}_{\text{tr.}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})$	$-\frac{\partial}{\partial \boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial \boldsymbol{\theta}} l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top \Big\vert_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \boldsymbol{H}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t)^{-1}\frac{d}{d\boldsymbol{\theta}}\mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))$
Approx. Quadratic	$-H(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})\frac{\partial}{\partial \boldsymbol{\theta}}\mathbb{L}_{\text{tr.}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})$	$-\frac{\partial}{\partial \boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial \boldsymbol{\theta}} l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top \Big\vert_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \boldsymbol{H}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t)\frac{d}{d\boldsymbol{\theta}}\mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))$
Linear	$-\frac{\partial}{\partial \boldsymbol{\theta}}\mathbb{L}_{\text{tr.}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})$	$-\frac{\partial}{\partial \boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial \boldsymbol{\theta}} l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top \Big\vert_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \frac{d}{d\boldsymbol{\theta}}\mathbb{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))$
No Val	$\mathbf{1}$	$-\frac{\partial}{\partial \boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial \boldsymbol{\theta}} l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top \Big\vert_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}$

Table 4.1: Proposed approximations for $\Delta\hat{\theta}_\psi$.

is common in second-order optimization. Thus the update term in Equation 4.12 can be obtained as

$$\boldsymbol{v}_{\boldsymbol{\psi}_t} \equiv \arg\min_{\boldsymbol{v}} Q(\boldsymbol{v}) \stackrel{\text{def}}{=} \{\frac{1}{2}\boldsymbol{v}^\top \boldsymbol{H}_{\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)}\boldsymbol{v} - \boldsymbol{g}_{\text{val}}^\top \boldsymbol{v}\}\ , \tag{4.13a}$$

$$\boldsymbol{\psi}_{t+1} \leftarrow \boldsymbol{\psi}_t + \alpha\, \boldsymbol{v}_{\boldsymbol{\psi}_t}.\nabla_{\boldsymbol{\psi}}\boldsymbol{g}_{\text{train}}\ . \tag{4.13b}$$

The CG approach only requires the evaluation of $\boldsymbol{H}_{\hat{\boldsymbol{\theta}}}\boldsymbol{v}$. Using automatic differentiation, Hessian-vector product requires only one forward and backward pass, same as a gradient, without explicitly forming $H_{\hat{\theta}}$. In practise, a good approximation for the inverse hessian vector product can be obtained with few iterations, as noted in Martens (2010).

Approximations for $\Delta\hat{\theta}_\psi$ We proposed a novel approximation for the solution of the inner problem. To further reduce the compute overhead, we propose approximations for $\Delta\hat{\theta}_\psi$. Table 4.1 shows some other alternative approximations for the inner problem, which can be obtained by using a linear approximation for the inner problem or using an approximate quadratic approximation. Using automatic differentiation, Hessian-vector product requires only one forward and backward pass, same as a gradient. Another baseline we try is a constant approximation for the inner problem where the method does not use the real validation set at all and just finds data which gives minimum model loss.

4.5 Experiments

In this section, we demonstrate the effectiveness of the proposed method in learning simulator parameters in two different scenarios. First we evaluate our method

on a simulator with the goal of performing a per-pixel semantic segmentation task. Second, we also conduct experiments with physically based rendering for solving object detection task on real world data. In supplementary material we provide more details about the data generation process from a simulator. In our experiments, we have used two physically based simulators: Blender-based CLEVR and the Arnold renderer.

Baselines In all our experiments, we compare our proposed method for learning simulator parameters against three state-of-the-art baseline algorithms. The main baseline is "learning to simulate" (LTS) Ruiz et al. (2019) which uses the REINFORCE gradient estimator. As the code for this is not public, we implemented it. Please note that Meta-sim Kar et al. (2019) also uses REINFORCE. In addition, we also compare against the two most established hyper-parameter optimization algorithms in machine learning; for Bayesian optimization we use the opensource Python package `bayesian-optimization` Nogueira (2014–) and for random search we used the Scikit-learn Python library Pedregosa et al. (2011). Please note that prior work Ruiz et al. (2019); Kar et al. (2019); Yang and Deng (2020) did not compare against these two approaches and in our results we found that out-of-the-box BO and Random search outperform REINFORCE Ruiz et al. (2019).

4.5.1 CLEVR Blender

In this experiment, we use the CLEVR simulator Johnson et al. (2017) which generates physically based images. The main task is semantic segmentation of the three classes present in the CLEVR benchmark, namely, Sphere, Cube and Cylinder. The images are generated using the CLEVR dataset generator. We optimize multiple CLEVR rendering parameters including the intensity of ambient light, back light, number of samples, number of bounces of light, image size, location of the objects in the scene, and materials of objects. The validation set is composed of synthetic images generated with a particular simulator configuration shown in Figure 4.5.

Task Network For the task network, we use a UNet Ronneberger et al. (2015) with eight convolutional layers. UNet is very common for segmentation and we

Method	Time	Images	Test mIoU
REINFORCE (LTS)	3h2m	3,750	61.0
Random search	2h38m	2,090	60.8
BO	**43m**	960	62.9
AutoSimulate	53m	**390**	**62.9**

Table 4.2: Segmentation on Clevr. Comparison of time, number of images, and test accuracy achieved by different methods.

have used an openly available Pytorch implementation[1] of UNet. The performance is measured in terms of mean IoU (intersection over union).

Results Quantitative results are shown in Table 4.2. BO and LTS methods to learn simulator parameters achieve similar test accuracy to ours. However, both these methods generate significantly more images to reach a similar accuracy. Essentially, to reach to the same level of test accuracy, the proposed approach requires 2.5× and 5× less data than BO and LTS methods respectively. This translates into saving time and resources required for the data generation and CNN training steps. Figure 4.5 shows some qualitative examples for this task.

4.5.2 Photorealistic Renderer Arnold

We next evaluate the performance of our proposed method on real-world data. For this, we use LineMod-Occluded (LM-O) dataset Hodan et al. (2018) for object detection task that consists of 3D models of objects. The dataset consists of eight object classes that includes metallic, non-Lambertian objects, e.g., metallic cans. The data has recently been used for benchmarking object detection problem Hodaň et al. (2019). We use the same test split for evaluating the performance of our method. Further, we use the same simulator as Hodan et al. Hodaň et al. (2019), based on Arnold Georgiev et al. (2018), to generate photo-realistic synthetic data for training an object detector model. Note that Hodan et al. Hodaň et al. (2019) heavily relied on human expert knowledge to correctly decide the distributions for different simulator parameters. In comparison, we show how our approach can

[1] `https://github.com/jvanvugt/pytorch-unet`

Figure 4.3: Synthetic images generated with Arnold renderer used for training.

be used to instead learn the optimal distribution over the simulator parameters without sacrificing accuracy.

In this experiment, we are given three scenes and nine locations within each scene. These locations signify the locations within the scene where objects can be placed. They can be arbitrarily chosen or can be selected by a human. Further, there are two rendering quality settings (high and low). The task is to optimize the categorical distribution for finding the fraction of data to be generated from each of these locations from different scenes under the two quality settings. Thus, the problem requires optimising 54 simulator parameters. Some of the images generated during simulator training have been shown in Figure 4.3.

Task Network For this task we use Yolo Redmon and Farhadi (2017), which is an established method for object detection and the code is freely available and easy to use[2]. We use Yolo-spp which has 112 layers with default parameter setting. The object detection performance is measured in terms of mean average precision (mAP@0.50).

Results Quantitative results are provided in Table 4.3 where we compare the presented method against the baselines. We evaluate these methods on three different criteria: mAP accuracy achieved on the object detection test set, total images generated during training of the simulator, and total time taken to complete simulator training. AutoSimulate provides significant benefit over the baseline

[2]`https://github.com/ultralytics/yolov3`

Method	Val. mAP	Images	Time(s)	Test mAP
REINFORCE (LTS)	40.2	86,150	114,360	37.2
Bayesian Optimization	39.3	9,200	83,225	37.5
Random Search	40.3	34,300	134,318	37.0
AutoSimulate	37.1	8,950	23,193	36.1
AutoSimulate(Approx Quad)	40.1	**2,950**	**2,321**	37.4
AutoSimulate (Linear)	**41.4**	17,850	30,477	**45.9**

Table 4.3: Object Detection. Comparison of methods. We run each method for a 1,000 epochs and report: *Val. mAP*: maximum validation mAP, *Images* and *Time*: number of images generated and time spent to reach maximum validation mAP, *Test mAP*: test mAP of the result.

Figure 4.4: Sample detections on real images from LM-O dataset, using a network trained on synthetic images generated by Arnold renderer, which is optimized with AutoSimulate using real-world images.

methods on all the criteria. Our method, AutoSimulate (Linear), achieves a remarkable improvement of almost 8 percent in mAP on test set. Further, it requires much lesser data (Figure 4.6) generation in comparison to baseline methods, and takes almost 2.5–4× less time to train simulator parameters compared to all the baselines including LTS, BO and random search. Our AutoSimulate (Approx Quad) is almost 35–60× faster than the baselines while also achieving the same mAP accuracy. This shows the effectiviness of the proposed method for learning simulator parameters. In Figure 4.4, we show qualitative examples of object detections in real-world images from the LM-O dataset, using a neural network trained on synthetic images generated by Arnold renderer optimized using AutoSimulate.

In supplementary material, we also show results on training simulator along with Faster-rcnn Ren et al. (2017) another popular object detection model.

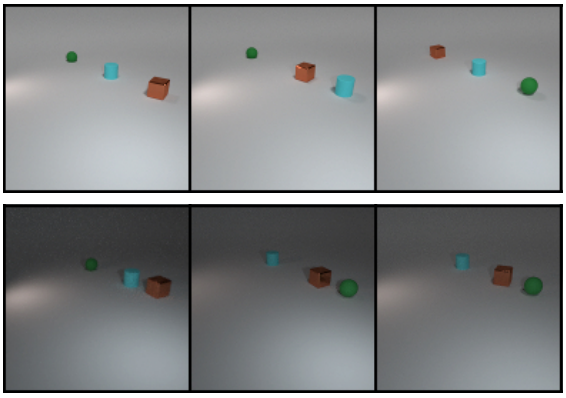


Figure 4.5: Images Rendered with the Clevr Simulator. Top: samples from validation set. Bottom: images rendered during the simulator training, showing variation in the quality of images, lighting in the scene, and location of objects.

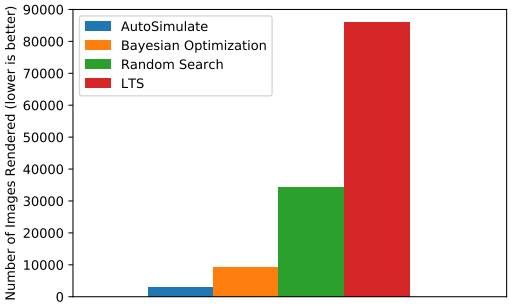


Figure 4.6: Comparison of the number of synthetic images required during training using the photorealistic renderer Arnold.

4.5.3 Additional Studies

Approximations for θ In this ablation, we analyse the different possible approximations for $\Delta\hat{\theta}$. The quantitative results are provided in Table 4.4. We observe that linear approximation of $\Delta\hat{\theta}$ achieves the best test accuracy (mAP). Further, our Approximate Quadratic takes the least time to converge and requires the least amount of data generation. Thereby giving the user freedom to select the approximation based on their speed–accuracy requirements.

Method	Test mAP	Time(s)	Images
Exact Quadratic (Ours)	36.1	2,3193	8,950
Approximate Quadratic (Ours)	37.4	**2,321**	**2,950**
Linear (Ours)	**45.9**	30,477	1,7850
No Validation	29.3	5,539	6,400

Table 4.4: Effect of Approximations

	0 frozen layers			98 frozen layers			104 frozen layers		
Method	mAP	Time(s)	Images	mAP	Time(s)	Images	mAP	Time(s)	Images
REINFORCE (LTS)	37.2	114,360	86,150	33.0	114,360	86,150	31.9	145,193	104,600
Bayesian Optimization	37.5	83,225	**9,225**	31.7	13,940	3,550	31.7	30,538	6,050
Random Search	36.8	134,137	34,300	30.2	8,913	3,500	28.9	73,411	21,650
Ours	**45.9**	**30,477**	17,850	**37.1**	**2,321**	**2,950**	**35.8**	**958**	**1,000**

Table 4.5: Effect of Freezing Layers

Effect of Freezing Layers It is a common practise to train on synthetic data with the initial layers of the network frozen and trained on real data. For this ablation, we use networks pretrained on COCO dataset. The effect of freezing different numbers of layers are shown in Table 4.5. In particular, we show the effect of freezing 0, 98 and 104 layers out of the total 112 layers. We observe that freezing no layers achieves better accuracy than freezing layers of the CNN model. However, it leads to higher convergence time. The faster convergence of frozen layers can be attributed to fast Hessian approximation computation.

Generelization and Effect of Network Size In this ablation, we study whether a simulator trained on a shallow network generalizes to a deeper network. We first examine the effect of network depth on simulator training. In particular, we show results of using two networks: YOLO-spp with 112 layers and YOLO-tiny with 22 layers in Table 4.6. Our approach on shallow network takes almost 7×, 15×, 135× less time to converge than LTS, random search and BO methods respectively. On the other hand, our method on deep network takes 4×, 2.5× and 4× less time than the three baseline methods. This highlights that the relative improvement of our method with the shallow network is much better than the deeper network. Further, in the supplementary material we also show the generalization of simulator parameters trained using shallow network on generating data for training deeper network. It gives users freedom to select size of network according to resources available for training the simulator.

	Yolo-spp			Yolo-Tiny		
Method	mAP	Time(s)	Images	mAP	Time(s)	Images
REINFORCE (LTS)	37.2	114,360	86,150	**24.7**	3,475	11,550
Bayesian Optimization	37.5	83,225	**9,225**	19.5	65,760	35,700
Random Search	36.8	134,137	34,300	20.6	7,319	11,620
Ours	**45.9**	**30,477**	17,850	21.2	**484**	**280**

Table 4.6: Effect of Network Size

4.6 Conclusion

Recent methods optimize simulator parameters with the objective of maximising accuracy on a downstream task. However these methods are computationally very expensive which has hindered the widespread use of simulator optimization for generating optimal training data. In this work, we propose an efficient algorithm for optimally generating synthetic data, based on a novel differentiable approximation of the objective. We demonstrate the effectiveness of our approach by optimising state-of-the-art photorealistic renderers using a real-world validation dataset, where our method significantly outperforms previous methods.

Acknowledgements Harkirat is wholly funded by a Tencent grant. This work was supported by ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC grant Seebibyte EP/M013774/1 and EPSRC/MURI grant EP/N019474/1. We would like to acknowledge the Royal Academy of Engineering, and also thank Ondrej Miksik, Tomas Hodan and Pawan Mudigonda for helpful discussions.

5

Meta-Learning Deep Visual Words for Fast Video Object Segmentation

Contents

Abstract

Personal robots and driverless cars need to be able to operate in novel environments and thus quickly and efficiently learn to recognise new object classes. We address this problem by considering the task of video object segmentation. Previous accurate methods for this task finetune a model using the first annotated frame, and/or use additional inputs such as optical flow and complex post-processing. In contrast, we develop a fast, causal algorithm that requires no finetuning, auxiliary inputs or post-processing, and segments a variable number of objects in a single forward-pass. We represent an object with clusters, or "visual words", in the embedding space, which correspond to object parts in the image space. This allows us to robustly match to the reference objects throughout the video, because although the global appearance of an object changes as it undergoes occlusions and deformations, the appearance of more local parts may stay consistent. We learn these visual words in an unsupervised manner, using meta-learning to ensure that our training objective matches our inference procedure. We achieve comparable accuracy to finetuning based methods (whilst being 1 to 2 orders of magnitude faster), and state-of-the-art in terms of speed/accuracy trade-offs on four video segmentation datasets. Code is available at `https://github.com/harkiratbehl/MetaVOS`.

Figure 5.1: Video object segmentation using a dictionary of deep visual words. Our proposed method represents an object as a set of cluster centroids in a learned embedding space, or "visual words", which correspond to object parts in image space (bottom row). This representation allows more robust and efficient matching as shown by our results (top row). The visual words are learned in an unsupervised manner, using meta-learning to ensure the training and inference procedures are identical. The t-SNE plot Maaten and Hinton (2008) on the right shows how different object parts cluster in different regions of the embedding space, and thus how our representation captures the multi-modal distribution of pixels constituting an object.

5.1 Introduction

Personal robots and driverless cars need to be able to operate in novel environments, and thus be able to quickly and efficiently learn to recognise object categories that they were not originally trained on. Furthermore, detailed segmentations of objects are also required for applications such as robot manipulation Kenney et al. (2009); Siam et al. (2019), grasping and learning object affordances Do et al. (2018); Siam et al. (2019). Finally, as live camera streams are processed in such applications, efficient and causal algorithms are required. This paper addresses these problems by considering the task of video object segmentation, following the protocol defined in the DAVIS datasets Caelles et al. (2018); Pont-Tuset et al. (2017). Here, the ground-truth object mask of one or more objects are provided only in the first frame, which must then be tracked at a pixel-level throughout the rest of the video. Since obtaining even a pixelwise segmentation for a single-frame may be too onerous in robotics applications, we also further extend the problem definition to only provide a bounding-box of each object in the first frame.

Accurate approaches to video segmentation trained a fully convolutional network (FCN) Long et al. (2015) for foreground/background segmentation on existing datasets, and then adapted it to the testing video by finetuning the network on the first, fully-annotated frame Caelles et al. (2017); Luiten et al. (2018); Voigtlaender

and Leibe (2017); Khoreva et al. (2017); Shin Yoon et al. (2017); Ci et al. (2018). Although these methods produce accurate results (and can be improved further by using optical flow Tsai et al. (2016); Bao et al. (2018); Cheng et al. (2017); Hu et al. (2017); Xiao et al. (2018) or post-processing with DenseCRF Krähenbühl and Koltun (2011); Caelles et al. (2017); Bao et al. (2018); Cheng et al. (2018)), they are extremely time consuming, taking between 700s to 3h to finetune per DAVIS video Caelles et al. (2017); Khoreva et al. (2017), rendering them unsuitable for real-life applications and robotics.

This paper, in contrast, considers the more challenging (and practical) scenario where the network is not finetuned at all, and uses no optical flow or extra post-processing, in order to develop a fast and causal algorithm. Our approach is inspired by metric-learning methods which embed pixels from the same object close to each other in a learned embedding space, and pixels from different objects far apart. Chen *et al.* Chen et al. (2018b) used this idea to formulate video segmentation as a pixel-level retrieval task, where each pixel of the ground-truth mask was embedded in the first frame to form an index, and pixels in subsequent frames were classified with nearest neighbours. Contrastingly, in the related context of few-shot learning, Prototypical networks Snell et al. (2017b) represent each class with the mean of their embeddings and classify subsequent queries with a softmax over distances to each prototype.

Prototypical networks, although simple and fast, do not have sufficient capacity to model complex, multi-modal data distributions such as an object in a video that undergoes deformations, occlusions and viewpoint changes. Nearest neighbour approaches Chen et al. (2018b); Li et al. (2018); Najafi et al. (2018); Hu et al. (2018) have greater modelling capacity, but are more computationally expensive as the time and memory cost to perform a lookup grows linearly with the size of the index. For pixel-level tasks, they also store many redundant pixels with similar appearance in the index. Furthermore, they are more prone to overfitting and noise, which becomes more prevalent during the "online adaptation" of video segmentation

models to account for variations throughout the video Chen et al. (2018b); Hu et al. (2018); Voigtlaender and Leibe (2017); Ci et al. (2018).

Our flexible approach interpolates the spectrum of metric learning approaches by representing an object with a fixed number of cluster centroids in the embedding space. We denote this as a dictionary of visual words, because each cluster centroid in the embedding space corresponds to a part of the object in the image space as shown in Fig. 5.1, even though these words are formed in an unsupervised manner.

The use of visual words enables more robust matching, because even though an object as a whole may be subject to occlusions, deformations, viewpoint changes, or disappear and reappear from the same video, the appearance of some of its more local parts may stay consistent. Moreover, the robustness of this approach allows us to easily extend it to the scenario where we only have weak bounding-box supervision in the first frame.

These visual words are learned without any explicit supervision by clustering our embedding space, and using meta-learning to ensure that our training objective matches our inference procedure. This is in contrast to related metric-learning based approaches Chen et al. (2018b); Najafi et al. (2018); Li et al. (2018) which are trained with surrogate, and sometimes unstable, losses. Furthermore, as our method requires only a single forward-pass to segment a variable number of objects per video, it naturally scales to the multi-object setting. Related methods Wug Oh et al. (2018), in contrast, segment each object independently before combining results and are thus slower for multiple objects.

The advantages of our simple and intuitive approach is reflected by its performance on multiple single- and multi-object video segmentation datasets (DAVIS 2016 Perazzi et al. (2016), DAVIS 2017 Pont-Tuset et al. (2017), SegTrack v2 Li et al. (2013), YouTube-Objects Prest et al. (2012); Jain and Grauman (2014)) where we achieve comparable accuracy to finetuning-based methods (whilst being 1 to 2 orders of magnitude faster), and lie on the Pareto front as no other published methods to our knowledge are both faster and more accurate.

5.2 Related Work

Fine-tuning based approaches The most accurate video segmentation methods using the DAVIS protocol Perazzi et al. (2016); Pont-Tuset et al. (2017) currently finetune models on the first frame of the video Luiten et al. (2018); Caelles et al. (2017); Voigtlaender and Leibe (2017); Khoreva et al. (2017) and/or use optical flow Hu et al. (2017); Xiao et al. (2018); Cheng et al. (2017); Tsai et al. (2016) or DenseCRF Caelles et al. (2017); Bao et al. (2018); Cheng et al. (2018) post-processing, or use self-paced learning Zhang et al. (2017a) to improve performance. Our proposed approach does not involve finetuning, or additional information such as optical flow, and still achieves comparable performance whilst being one to two orders of magnitude faster.

Fast approaches Fast approaches to video segmentation, that do not finetune on the first frame or use optical flow, can broadly be divided into methods performing mask propagation or metric learning. Mask propagation methods, such as Perazzi et al. (2017); Wug Oh et al. (2018); Jampani et al. (2017), use the segmentation mask from the one frame to guide the network to predict the mask in the next frame (*i.e.* pixel-level tracking). These methods use the prior that objects move smoothly and slowly over time, and thus struggle when there are temporal discontinuities like occlusion or rapid motion. Moreover, errors accumulate over time as the model "drifts", particularly if the algorithm loses track of the object. Li *et al.* Li and Change Loy (2018) addressed this issue using re-identification modules which traverse the video back-and-forth to recover any potential missed objects. However, this method is not causal as it looks at future frames. Oh *et al.* Wug Oh et al. (2018) do not only use the previous frame, but also the first reference frame, to guide the tracking. However, this does not completely alleviate the problem of model drift, as if the model loses track of the object, its appearance may have changed so much from the first frame that the reference frame is not effective in recovering it. Moreover, since these methods match the entire object as a whole, they struggle with occlusions. This is in contrast to our approach which represents objects by their

constituent parts to be more robust to appearance changes. Finally, Wug Oh et al. (2018) is designed for tracking a single object, and thus handling multiple objects require processing each object individually before heurstically merging results. Our method in comparison segments multiple objects in a single-forward pass.

Metric learning based approaches Our work is more similar to methods using pixel-to-pixel matching or metric learning Chen et al. (2018b); Hu et al. (2018); Najafi et al. (2018); Shin Yoon et al. (2017); Li et al. (2018). Chen *et al.* Chen et al. (2018b) formulated video segmentation as a pixel-level retrieval problem, where embeddings from the first reference frame are used to form an index for a nearest neighbour classifier. Note that the method of Chen et al. (2018b) is trained with a variant of the triplet loss, which though common for metric learning, does not optimise explicitly for the nearest neighbour search at inference time. The triplet loss is also difficult to train with, as it is very sensitive to triplet selection Schroff et al. (2015); Hermans et al. (2017). Siamese networks have also been employed in a similar manner Hu et al. (2018); Najafi et al. (2018); Shin Yoon et al. (2017), where one branch computes embeddings from the annotated first frame which are used to match to the embeddings computed by the other branch on the current frame. When classifying query images, these methods all effectively search all the pixels from the reference frame. This approach is not only expensive in terms of time and memory (as it retains redundant embeddings of similar pixels), but is also more susceptible to noise. This is an issue during the "online adaptation" Chen et al. (2018b); Ci et al. (2018); Hu et al. (2018); Voigtlaender and Leibe (2017) of the model which may introduce incorrectly labelled embeddings. Our method retains only cluster centroids in the embedding space (which correspond to exemplars of object parts), which enables faster and more robust matching.

Taking inspiration from classical computer vision, learned feature descriptors have been used in localization Sarlin et al. (2019), motion removal Sun et al. (2018) and feature matching Ono et al. (2018) because of their robustness and efficiency. Note that our main contributions are that we can learn to segment new object

classes in video from very few examples, and use a meta-learning technique to train our model so that the training and testing procedures match each other. The utility of object parts for more robust matching for video segmentation has been identified before by Cheng et al. (2018). However, Cheng *et al.* Cheng et al. (2018) use handcrafted heuristics to form object parts based on bounding boxes in the image space. In contrast, we cluster our embedding space in an unsupervised manner to obtain visual words which resemble object parts (as pixels with similar appearance cluster together). Moreover, the method of Cheng *et al.* Cheng et al. (2018) – which tracks bounding boxes of object parts and then merges foreground segmentations within these boxes – consists of two separately trained modules (using different datasets), whereas our method is trained via meta-learning with a single objective function that matches our inference procedure.

Meta-learning Finally, we note that meta-learning has not been explored much in the context of video segmentation. Yang *et al.* Yang et al. (2018) used meta-learning to adapt the weights of the final layer of a segmentation network at test time. This is in contrast to our method which adaptively computes the initial dictionary of visual words from the first labelled frame in the video. Note that our method can be viewed as a generalisation of Protoypical networks Snell et al. (2017b) and Matching networks Vinyals et al. (2016b) for few-shot classification. Prototypical networks represents the training data from each class as a single prototypical vector. Matching networks, on the opposite end of the spectrum, consider all training data samples of a particular class to make a classification regardless of how redundant or noisy these samples may be. Our method interpolates these two methods by representing an object class via a fixed number of cluster centroids, which correspond to exemplars of object parts in the case of video segmentation.

5.3 Proposed Approach

We first describe the formulation of video object segmentation as a meta-learning problem. This allows us to train our model in same way that it will be tested,

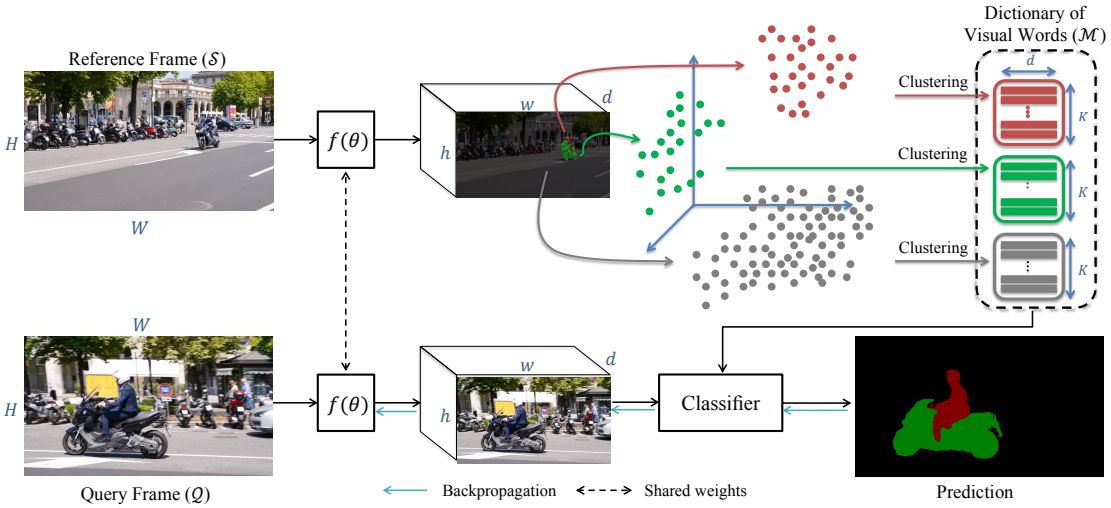


Figure 5.2: Overview of the proposed method. The first frame of the video (reference frame), which forms the support set $\mathcal{S}$ in our meta-learning setup, passes through a deep segmentation network $f(\theta)$ to compute a $d = 128$ dimensional embedding for each pixel. A dictionary of deep visual words are then learned by clustering these embeddings for each object in the reference frame (Eq. 5.2). Pixels of the query frame are classified as one of the objects based to their similarities to the visual words (Eq. 5.3 and Eq. 5.4). The model is meta-trained by alternately learning the visual words given model parameters θ, and learning model parameters given the visual words. During testing, the bottom path, without blue lines, is applied to all frames, whereas, the top path is only applied to the first frame and the online adaptation frames.

unlike other metric-learning based approaches to video segmentation Chen et al. (2018b); Hu et al. (2018).

5.3.1 Video Object Segmentation as Meta-Learning

Meta-learning, or learning to learn, is often defined as learning from a number of tasks in the training set, to become better at learning a new task in the test set Schmidhuber (1987b); Naik and Mammone (1992); Hochreiter et al. (2001b); Finn et al. (2017a). In the context of video object segmentation, the task is to learn from the ground-truth masks of the objects in the first frame of the video (support set) to segment and track them in rest of the video (query set). Our meta-learning objective is to learn model parameters θ on a variety of tasks (videos), which are sampled from the distribution $p(\mathcal{T})$ of training tasks (*i.e.* meta-training set), such that the learned model performs well on a new unseen task (test video). Denoting the loss of the model on the n^{th} task, $\mathcal{T}_n$, as $\mathcal{L}_{\mathcal{T}_n}(\theta)$, the meta-training objective is thus

$$\theta^* = \arg\min_{\theta} \sum_{\mathcal{T}_n \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_n}(\theta). \tag{5.1}$$

The support set $\mathcal{S}$ is the set of all labeled pixels in the first frame, $\mathcal{S} = \{x_i, y_i\}_{i=1}^{N}$. Here x_i represents the pixel i in the first frame, $y_i \in \mathcal{C} = \{1, ..., C\}$ is the ground truth class label of pixel x_i, N is the number of labeled pixels in the frame, and C is the number of object classes that need to be tracked and segmented in the video. Similarly the query set is defined by $\mathcal{Q} = \{x_j, y_j\}_{j=1}^{N_Q}$, where N_Q is the number of labelled pixels in the video after the first frame, j is the index. The output of each task $\mathcal{T}$ is the set of predicted class labels for the pixels in $\mathcal{Q}$, $\hat{\mathcal{Y}} = \{\hat{y}_j\}_{j=1}^{N_Q}$.

Next, we describe our model for estimating the outputs of each task, *i.e.* the object label for every pixel in the query frames of the video.

5.3.2 Model

In order to predict the label for each pixel in the query set $\mathcal{Q}$, we need to learn a representation for each object using information from the support set $\mathcal{S}$. We represent each object in the video using a dictionary of deep visual words. Each pixel in the query set is then labelled based on the deep visual word that it is assigned to. Our method is an online method does not use future-frames during runtime, i.e, to segment a new frame, only the information upto that point is used.

Learning visual words is a challenging task, as we do not have any ground truth information of the object parts that they correspond to. Consequently, as summarised in Fig. 5.2, we use a meta-training algorithm, where we alternate between the unsupervised learning of deep visual words (Sec. 5.3.2) and supervised learning of pixel classification given these visual words (Sec. 5.3.2). Our model thus learns to learn a better classifier, by optimising the visual words that it will produce at test-time.

Unsupervised Learning of Deep Visual Words

We initially pass the first frame of the video, which is the support set $\mathcal{S}$, through a deep neural network $f(\theta)$, which is an artificial neural network with multiple layers between input and output, to compute the embedding for each pixel x_i in $\mathcal{S}$, $f_\theta(x_i)$. We then compute a set of deep visual words for all the pixels in each object class.

Let $\mathcal{S}_c$ be the set of pixels in $\mathcal{S}$ with class label c. Each set $\mathcal{S}_c$ is partitioned into K clusters $\mathcal{S}_{c1}, ..., \mathcal{S}_{cK}$ using the k-means algorithm Arthur and Vassilvitskii (2007), with μ_{ck} being the respective centroids of the clusters, using the objective:

$$\mathcal{S}_{c1}, ..., \mathcal{S}_{cK} = \underset{\mathcal{S}_{c1},....,\mathcal{S}_{cK}}{\arg\min} \sum_{k=1}^{K} \sum_{x_i \in \mathcal{S}_{ck}} \|f_\theta(x_i) - \mu_{ck}\|_2^2 , \tag{5.2a}$$

$$\mu_{ck} = \frac{1}{|\mathcal{S}_{ck}|} \sum_{x_i \in \mathcal{S}_{ck}} f_\theta(x_i). \tag{5.2b}$$

In other words, we represent the distribution of the pixels within each set $\mathcal{S}_c$ in the learned embedding space with a set of deep visual words $\mathcal{M}_c = \{\mu_{c1}, ..., \mu_{cK}\}$. We can, in principle, use any clustering algorithm here and choose k-means as it is computationally efficient and simple.

Supervised Learning for Pixel Classification

Once the deep visual words for each object have been constructed, the probability of assigning a pixel $x_j \in \mathcal{Q}$ to the k^{th} visual word from the c^{th} object class is computed using a non-parametric softmax classifier,

$$p(c_k|x_j) = \frac{\exp\Big(cos(\mu_{ck}, f_\theta(x_j))\Big)}{\sum_{\mu_i \in \mathcal{M}} \exp\Big(cos(\mu_i, f_\theta(x_j))\Big)}, \tag{5.3}$$

where $\mathcal{M} = \bigcup_{c=1}^{C} \mathcal{M}_c$ is the dictionary of deep visual words for all objects present in the video, and *cos* is the cosine similarity function. We enable our model to account for intra-class variations by encouraging each pixel to resemble only one relevant visual word. As a result, the probability of pixel x_j taking the object class label c is defined as

$$p(\hat{y_j} = c|x_j) = \frac{\max_{k \in \{1,..,K\}} p(c_k|x_j)}{\sum_{c'=1}^{C} \max_{k \in \{1,..,K\}} p(c'_k|x_j)}. \tag{5.4}$$

This allows our model to learn meaningful visual words that correspond to the diverse object parts that constitute an object. Note from the T-SNE visualisation of our embeddings in Fig. 5.1 that pixels from different parts of the same object cluster in separate regions of the embedding space. Finally, our loss function for this pixel classification problem is the cross-entropy loss.

5.3.3 Meta-training procedure

Each iteration of our meta-training algorithm consists of an unsupervised learning process to construct a dictionary of visual words from the support set $\mathcal{S}$, followed by a supervised learning step where the segmentation network parameters, θ, are updated by minimsing the cross-entropy loss function according to Eq. 5.1. In other words, the model learns to learn deep visual words in the first frame of the video to minimise a pixel-level loss over the rest of the video.

Our method is a form of non-parameteric meta-learning, as described in Finn and Levine (2019). Note that the cluster centroids can be seen as the parameters of the final classification layer (Eq. 5.3). Prototypical networks Snell et al. (2017b) represent each class with a single prototypical vector (*i.e.* one visual word), whilst Matching networks Vinyals et al. (2016b) represent each class using all the samples of that class in the support set (*i.e.* the embedding of each pixel in $\mathcal{S}$ would form a visual word). Our method interpolates between these two approaches to build a more robust representation of the support set $\mathcal{S}$. Also note that previous metric-learning approaches to video object segmentation such as Chen et al. (2018b) learn an embedding using variants of the triplet loss and perform nearest neighbour classification at test time. This approach is thus similar to Matching networks Vinyals et al. (2016b), with the key difference being that the training objective (triplet loss) does not correspond to the inference procedure (nearest neighbour search), which is an essential component for meta-learning Finn and Levine (2019). Our method ensures that meta-train and meta-test setup match.

5.3.4 Online Adaptation

The objects of interest from the first frame, as well as the background, often undergo deformations, occlusions, viewpoint changes. As a result, adapting the model throughout the video is vital to achieve good performance and done by state-of-art video methods Voigtlaender and Leibe (2017); Chen et al. (2018b); Hu et al. (2018); Behl et al. (2018).

We adapt our model by simply updating the set of visual words that represent the object. Concretely, given a dictionary of visual words $\mathcal{M}$, captured up to the frame t_j, we predict the segmentation map in frame $t_{j+\delta}$, and treat it as a new support set $\mathcal{S}^\delta = \{x_i^\delta, y_i^\delta\}_{i=1}^N$, where y_i^δ is the predicted object class for pixel x_i^δ. Next, we compute an updated set of deep visual words $\mathcal{M}^\delta$ from the new support set using k-means as described in Sec. 5.3.2, and compute their corresponding cluster centroid representations by

$$\mu_{ck}^\delta = \frac{1}{|\mathcal{S}_{ck}^\delta|} \sum_{x_i \in \mathcal{S}_{ck}^\delta} f_\theta(x_i). \tag{5.5}$$

To filter out incorrect predictions and prevent errors from compounding, we only add new words that still resemble the existing ones. This is based on the assumption that within a time interval δ, where δ is chosen moderately, the objects will deform slowly and their pixel-level embeddings will also not vary greatly. Concretely, we update the main visual word set $\mathcal{M}$ with the new set $\mathcal{M}^\delta$, if there are $m^\delta \in \mathcal{M}_c^\delta$ and $m \in \mathcal{M}_c$, for which $\left\|\mu_m^\delta - \mu_m\right\| \leq \alpha$.

Additionally, to ensure that online adaptation uses reliable and confident pixel-level predictions to update the visual words, we apply a simple outlier removal process (that assumes spatio-temporal consistency of objects over time) to the pixel-level predictions. Specifically, we discard regions from the adaptation process if they have no intersection with the predicted object mask in the previous frame, using connected components. Note that during online adaptation, none of the existing words within $\mathcal{M}$ are discarded, because each object may revert to its original shape, appearance or viewpoint during a video. This is also why the Eq. 5.4 takes the maximum value to match to only the most relevant visual word. The effect of this online adaptation procedure, and other design choices, are experimentally validated next.

5.4 Experimental evaluation

5.4.1 Experimental setup

Model We use a Deeplab v2 architecture as the encoder Chen et al. (2018a), $f(\theta)$, which uses a ResNet-101 He et al. (2016) backbone with dilated convolutions. The

encoder maps an input frame of size $H \times W$ to a feature of size $H \times W \times 2048$. We add an additional convolutional layer to produce an embedding of $d = 128$ dimensions, and bilinearly upsample this to the original image size. These 128-dimensional embeddings are then clustered to form our visual words. Unless otherwise specified, we use $k = 50$ visual words for the foreground object classes. As the background typically contains more variation, we use four times as manyclusters for the background. For online adaptation, we set $\alpha = 0.5$. The ablation study in Sec. 5.4.4 shows the effect of the number of visual words, k.

Datasets We evaluate on standard video segmentation datasets for tracking both single objects (DAVIS-2016 Perazzi et al. (2016), YouTube-Objects Prest et al. (2012); Jain and Grauman (2014)) and multiple objects (DAVIS-2017 Pont-Tuset et al. (2017), SegTrack v2 Li et al. (2013)) given fully-annotated object masks in the first frame. DAVIS-2016 contains 30 training and 20 validation videos. DAVIS-2017 extends DAVIS-2016 to 60 training and 30 validation videos. Furthermore, multiple objects (ranging from 1 to 5, with an average of 2) are annotated in the first frame and must be tracked through the video, making it considerably more challenging than DAVIS-2016. YouTube-Objects and SegTrack v2 do not have a training split, so we evaluate our model trained on DAVIS-2017 on them.

Training Following competing methods which use a model pretrained on image segmentation datasets Hu et al. (2018); Wug Oh et al. (2018); Chen et al. (2018b); Yang et al. (2018); Maninis et al. (2018); Voigtlaender and Leibe (2017), we initialise the encoder of our network using the public Deeplab-v2 model Chen et al. (2018a) that has been trained on COCO Lin et al. (2014). Thereafter, we meta-train our model following the "episodic training" procedure, which is the standard practice Vinyals et al. (2016b); Snell et al. (2017b); Finn et al. (2017a); Behl et al. (2019). Each training episode is formed by sampling a support set $\mathcal{S}$ and a relevant query set $\mathcal{Q}$. The idea of episodic training is to, at each iteration, mimic the inference procedure. In other words, the query set should be classified given only the support set. Here, we build each episode by first randomly sampling a video from the

training dataset, treating the pixels of the first frame of the video as $\mathcal{S}$, and randomly selecting a set of query frames from the rest of the video and treating their pixels as $\mathcal{Q}$. Randomly selecting sets of frames in the video make the method robust to temporal discontinuties, occlusions and object complexity, thereby making it more generic. We sample from as low as 3 frames to the entire video length.

Evaluation metrics We report standard metrics defined by the DAVIS protocol Perazzi et al. (2016): The mean IoU ($\mathcal{J}$), the F-score along the boundaries of the object ($\mathcal{F}$) and the mean of these two values ($\mathcal{J}\&\mathcal{F}$). We also report the "decay" Perazzi et al. (2016) in $\mathcal{J}$. This is calculated by splitting a video temporally into four clips, and taking the difference of the IoU in the last clip to the first clip. Lower scores of "decay" are better, and was proposed by Perazzi et al. (2016) to measure whether a model is robust or if its predictions degrade over time.

Finally, we also report our runtime per-frame. Our runtime is measured on a desktop machine with a single Titan X (Pascal) GPU, and an Intel i7-6850K CPU with six cores. More details and experiments are available in the supplementary material[1].

5.4.2 Comparison to state-of-art

Table 5.1 shows our state-of-art results on DAVIS-2017, DAVIS-2016, YouTube-Objects and SegTrack-v2. On all these datasets, there is no method that is both faster and more accurate than us. This Pareto front is also visualised in Fig. 5.4 for DAVIS-2017, the most challenging dataset. The methods that are more accurate than us all finetune on the first frame, use optical flow or additional post-processing such as CRFs Krähenbühl and Koltun (2011) and thus have a runtime that is larger by a factor of at least 8 Li and Change Loy (2018); Caelles et al. (2017).

The only method that is close to us in terms of speed and accuracy is RGMP Wug Oh et al. (2018). However, the runtime of RGMP increases linearly with the number of objects being tracked from the first frame. This is because RGMP

[1]Supplementary materials `https://harkiratbehl.github.io/`

processes each object instance independently through the "encoder" part of their network Wug Oh et al. (2018), and combine their results together at the end. Therefore, even though RGMP is faster than our method on DAVIS-2016 (a single-object dataset), it is actually slower on DAVIS-2017 (a multi-object dataset). As the authors did not report the runtime of their method on DAVIS-2017, we ran their publicly available inference code, and obtain an average runtime of 0.30s per frame on DAVIS-2017 (Tab. 5.1). However, DAVIS-2017 only averages 2 objects per video. We measured RGMP to average 0.11s, 0.41s and 0.60s per frame for videos with 1, 3 and 5 objects respectively. The runtime of our method, in contrast, increases much slower, taking 0.25s, 0.38s and 0.53s respectively. Thus, the runtime of RGMP increases by 5.4× from 1 object to 5 objects, whilst our runtime only increases by 2.1×. This is because our model only requires a single forward-pass through the network, irrespective of the number of objects being tracked. Thus, there is only a minor increase in runtime from DAVIS-2016 to DAVIS-2017 as there are more visual words. Note that the runtime advantage of our method would increase over RGMP Wug Oh et al. (2018) if even more objects were to be tracked in a video. Moreover, our speed could also be improved by implementing CUDA kernels for k-means clustering.

Note that our method also achieves a lower $\mathcal{J}$ Decay Perazzi et al. (2016) than RGMP Wug Oh et al. (2018) indicating greater robustness. This is also shown qualitatively in Fig. 5.3 where our method overcomes occlusions and can recover from errors made in previous frames, unlike mask propagation methods like RGMP. We believe that representing objects with cluster centroids in the embedding space (visual words), which correspond to object parts in the image space, increases the robustness of our matching, as the appearance of more local parts typically stays consistent even though the object as a whole transforms. And as our online adapatation process retains a memory of previous visual words, our method can handle objects disappearing and reappearing (Fig. 5.3) unlike RGMP.

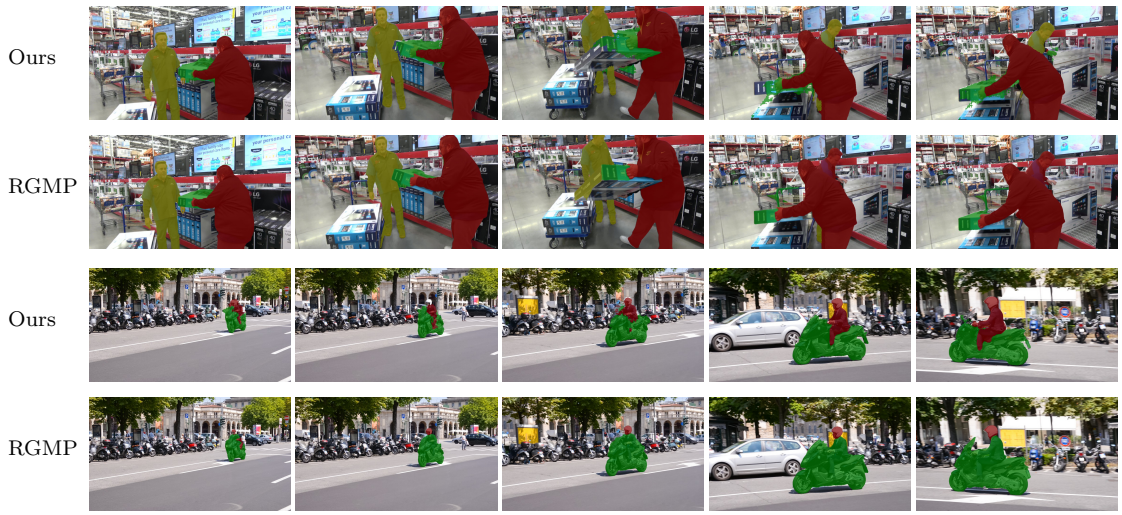


Figure 5.3: Qualitative comparison of our method to RGMP Wug Oh et al. (2018). RGMP obtains good results initially in the video (first two columns), but cannot recover after making errors (third column). Note how it misclassifies the yellow person (first example) and loses track of the rider (second example). In contrast, our method overcomes occlusions in all of these cases by robustly matching an object to its constituent parts.

5.4.3 Bounding box based initial mask

In this section, we do experiments when only bounding box supervision is available in the first frame. Hence the aim is to segment the object in the video, given only the bounding box in the first frame. To generate the dictionary of visual words for a given object, we use a mechanism similar to our online adaptation. We first take the embeddings of all pixels in the bounding box of the object and segment them into clusters. We then discard the ones that resemble the background, with the difference that here we use resemblance with the background to eliminate some regions of the bounding box. The remaining clusters are then used to construct the visual word dictionary for this object. The rest of the algorithm remains the same. The results are shown in Table 5.2. It can be seen that there is not a substantial drop in performance in comparison to the mask based case. We believe that our part-based approach makes our model more robust to the noise in the input in this scenario.

5.4.4 Ablation study

This section studies how different design choices in our algorithm impact overall performance on DAVIS-2017.

Effect of Meta-Learning To evaluate the efficacy of meta-learning, we evaluated our MS-COCO initialised network and obtained a mean IoU of ($\mathcal{J}$) of 50.7. Meta-training significantly improves our IoU to 63.9% (Tab. 5.1). Perhaps surprisingly, our initialisation already outperforms published work like Mask-RNN Hu et al. (2017) (Tab. 5.1).

Object representation We represent the object given in the first frame with a dictionary of k visual words in the embedding space. An alternative is to represent each object with a single vector, *i.e.* $k = 1$ (as in Prototypical networks Snell et al. (2017b)). In our case, this prototype is formed by taking the mean embedding of all pixels of the object labelled in the first frame. The other end of the spectrum is to represent each object with separate embeddings for all of its pixels, *i.e.* $k = n$ where n is the number of labelled pixels in the first frame, like Chen et al. (2018b) and Matching networks Vinyals et al. (2016b).

Table 5.3 compares these approaches for our MS-COCO pretrained network. It shows that using $k = 50$ clusters outperforms both nearest neighbour classification and a single prototypical vector per class. This motivates our reason for using k visual words to represent an object and suggests why we outperform methods such as Chen et al. (2018b) in Tab. 5.1.

Note how matching using our visual words representation has a similar runtime to a single prototype and is significantly faster than performing a nearest neighbour search. This is because the search time is linear in the number of pixels, $O(n)$. And like the other approaches we compare to in Tab. 5.3, we do the matching at full resolution. The runtime could be greatly reduced by doing the look-up on a subsampled image (for example, Chen et al. (2018b) do the look-up at 1/8 resolution which reduce the time by about a factor of 64).

Number of visual words Table 5.4 examines the effect that the number of visual words in the dictionary has on accuracy and runtime on DAVIS-2017. We can see that accuracy steadily increases as the number of clusters is increased from $k = 1$ (which corresponds to Prototypical networks Snell et al. (2017b)) and plateaus

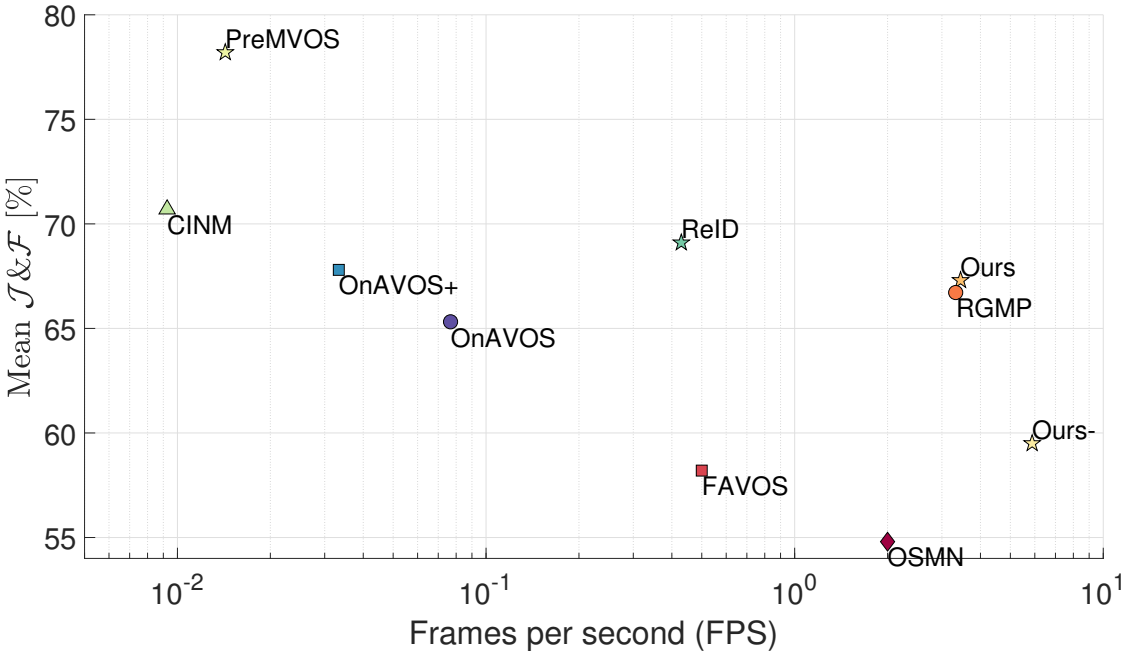


Figure 5.4: Comparison of speed and accuracy on DAVIS 2017. Entries on the Pareto front (*i.e.* no other method is both faster and more accurate) are marked by a star. Note the speed axis uses a logarithmic scale.

at $k = 50$. We believe that complex objects with high intra-object variations produce embeddings with multi-modal distributions, which is why they are better represented with multiple visual words. Although increasing k beyond 50 does not substantially change the accuracy, it does increase the runtime, which is why we use $k = 50$ when comparing to existing methods in Tab. 5.1. Setting the number of visual words to n, the number of pixels in the first frame, would amount to the nearest neighbour search done by Chen et al. (2018b).

5.5 Conclusion and Future Work

We proposed a novel representation of objects by their cluster centroids in the embedding space (visual words) which correspond to object parts. These visual words were learned without supervision, using meta-learning. Visual words enable robust matching, as the appearance of local parts may stay consistent whilst the object as a whole deforms or is occluded. Our novel representation, and meta-training procedure enabled our method to achieve state-of-art performance on four common datasets in terms of speed and accuracy trade-offs (with comparable accuracy to expensive finetuning-based methods that take at least 8 times longer).

Moreover, our method readily scales to multiple objects in videos, with its runtime only increasing slightly from single-object DAVIS-2016 to multi-object DAVIS-2017. Finally, the robustness of our part-based algorithm allows us to easily extend it to the scenario where we only have bounding-box supervision in first frame. Future work is to learn the number of clusters automatically, and learn to generate synthetic data Behl et al. (2020a) for video segmentation.

Acknowledgements Harkirat is wholly funded by a Tencent grant. This work was supported by the Royal Academy of Engineering, EPSRC/MURI grant EP/N019474/1 and FiveAI.

				DAVIS-2017					DAVIS-2016					YouTube-Objects	SegTrack-v2
Method	FT	PP	OF	$\mathcal{J}\&\mathcal{F}$(%)	$\mathcal{J}$(%)	$\mathcal{J}$ Decay(%)	$\mathcal{F}$(%)	Time(s)	$\mathcal{J}\&\mathcal{F}$(%)	$\mathcal{J}$(%)	$\mathcal{J}$ Decay(%)	$\mathcal{F}$(%)	Time(s)	$\mathcal{J}$(%)	$\mathcal{J}$(%)
OnAVOS Voigtlaender and Leibe (2017)	✓	✓		65.3	61.6	27.9	69.1	13	85.5	86.1	5.2	84.9	13	77.4	–
OSVOS^S Caelles et al. (2017)	✓	✓		68.0	64.7	15.1	71.3	–	86.5	**85.6**	5.5	87.5	4.50	**83.2**	65.4
PReMVOS† Luiten et al. (2018)	✓		✓	**78.2**	**74.3**	16.2	**82.2**	$\sim$ 70	**87.0**	85.5	8.8	**88.6**	$\sim$ 70	–	–
MaskRNN Hu et al. (2017)			✓	–	45.5	–	–	0.60	–	–	–	–	–	–	–
FAVOS Cheng et al. (2018)		✓		58.2	54.6	**14.1**	61.8	>1.80	80.9	**82.4**	**4.5**	79.5	1.80	–	–
CTN Jang and Kim (2017)			✓	–	–	–	–	–	71.4	73.5	15.6	69.3	1.33	–	–
FAVOS Cheng et al. (2018)				–	–	–	–	–	76.9	77.9	–	76.0	0.60	–	–
VPN Jampani et al. (2017)				–	–	–	–	–	67.8	70.2	12.4	65.5	0.63	–	–
BVS Maerki et al. (2016)				–	–	–	–	–	59.4	60.0	28.9	58.8	0.37	68.0	60.0
OSMN Yang et al. (2018)				54.8	52.5	21.5	57.5	0.50*	–	74.0	9.0	–	0.14	69.0	–
VideoMatch Hu et al. (2018)				–	56.5	–	–	0.35	–	81.0	–	–	0.32	79.7	–
RGMP Wug Oh et al. (2018)				66.7	**64.8**	18.9	68.6	0.30*	81.7	81.5	10.9	82.0	**0.13**	–	71.1
Ours$^-$				63.1	59.5	55.8	24.6	0.17	76.9	76.2	11.2	77.6	0.17	77.4	64.6
Ours				**67.3**	63.9	14.4	**70.7**	**0.29**	**82.1**	81.5	5.0	**82.7**	0.25	**81.1**	**72.0**

Table 5.1: State-of-art results among methods not performing finetuning on four common video object segmentation datasets. Legend: FT: Fine-Tuning on the first frame of the test video; PP: Post-Processing; OF: Optical Flow; Ours$^-$: Our model without online adaptation; †: An ensemble of models are used. *As the original authors did not report the runtime, we timed it using the public inference code. Evaluation metrics are detailed in Sec. 5.4.1.

	DAVIS-2017	DAVIS-2016	YouTube-Objects	SegTrack-v2
Ours	63.8	81.5	81.1	72.0
Ours-BB	51.5	77.5	75.8	66.5

Table 5.2: Results of our method ($\mathcal{J}\&\mathcal{F}$) with only bounding box based initialization. Ours-BB: Our model with only bounding box mask provided in first frame.

Model	$\mathcal{J}$(%)	Time(s)
Single prototype	32.9	**0.14**
5 Nearest neighbours	45.9	5.50
Visual words ($k = 50$)	**48.4**	0.17

Table 5.3: The effect of different object representations The same MS-COCO pretrained network is used, without any online adaptation. We use the 5 nearest neighbours, following Chen et al. (2018b).

Dictionary Size (k)	1	5	10	50	100	200	400	1500	3000
$\mathcal{J}(\%)$	49.9	54.5	54.8	55.8	56.3	56.3	56.4	56.2	54.5
Time (s)	0.140	0.168	0.170	0.173	0.199	0.254	0.373	0.97	1.42

Table 5.4: The effect of the size of visual word dictionary on model performance Results are on DAVIS-2017, without any online adaptation.

6

Alpha MAML: Adaptive Model-Agnostic Meta-Learning

Contents

Abstract

Model-agnostic meta-learning (MAML) is a meta-learning technique to train a model on a multitude of learning tasks in a way that primes the model for few-shot learning of new tasks. The MAML algorithm performs well on few-shot learning problems in classification, regression, and fine-tuning of policy gradients in reinforcement learning, but comes with the need for costly hyperparameter tuning for training stability. We address this shortcoming by introducing an extension to MAML, called Alpha MAML, to incorporate an online hyperparameter adaptation scheme that eliminates the need to tune meta-learning and learning rates. Our results with the Omniglot database demonstrate a substantial reduction in the need to tune MAML training hyperparameters and improvement to training stability with less sensitivity to hyperparameter choice.

6.1 Introduction

Meta-learning—or "learning to learn"—concerns machine learning models that can improve their learning quality by altering aspects of the learning process such as the model architecture, optimization rules, initialization, or learning hyperparameters (Thrun and Pratt, 2012; Schmidhuber, 1987a; Hochreiter et al., 2001a). An important application of meta-learning is in few-shot learning problems (Vinyals et al., 2016a; Behl et al., 2020b), where one is concerned with developing methods able to learn new concepts from one or only a few instances (Lake et al., 2015). In this paper we focus on the state-of-the-art *model-agnostic meta-learning* (MAML) (Finn et al., 2017b) method, which is a conceptually simple and general algorithm that has been shown to outperform existing approaches in tasks including few-shot image classification and few-shot adaptation in reinforcement learning (Antoniou et al., 2019). MAML aims to solve the few-shot learning problem by being just few gradient descent steps away from any new concepts, doing so by making the assumption that learning a new concept will just involve few parameter updates (Algorithm 3). In other words, MAML is based on learning an initial representation that can be efficiently fine-tuned for new tasks in a few steps.

The generality of MAML comes with the difficulty of choosing hyperparameters to achieve stable training in practice (Antoniou et al., 2019). MAML has two important hyper-parameters, namely the learning rate α and the meta-learning rate β, thus increasing any hyperparameter grid search computation by an order, and making it significantly more time and resource consuming than comparable methods. Another complication to this problem is the fact that it is currently not established whether the technique can benefit from a conventional decaying schedule for the inner learning rate α. Furthermore, a good value of α in MAML is even more important than for any conventional stochastic gradient descent (SGD) optimization, because only a handful of samples are available in the few-shot learning case. This has significant consequences, making it difficult to scale this algorithm to problems bigger than toy scales, due to the difficulty in assessing whether MAML is not suitable for a complex task or whether the hyperparameters are not sufficiently tuned.

In this paper, we provide a conceptually simple solution to this problem, by introducing an extension of the MAML algorithm to incorporate adaptive tuning of both the learning rate α and the meta-learning rate β. Our aim is to make it possible to use MAML without or with significantly less parameter tuning, and thus to reduce the need for grid search. We also aim to make the algorithm converge in fewer iterations. The solution we propose is based on the hypergradient descent (HD) algorithm (Baydin et al., 2018), which automatically updates a learning rate by performing gradient descent on the learning rate alongside original optimization steps. The proposed algorithm does not need any extra gradient computations, and just involves storing the gradients from the previous optimization step.

6.2 Related Work

Our work is primarily related with the subfields of hyperparameter optimization and meta-learning. In hyperparameter optimization one typically uses parallel runs to populate a selected grid of hyperparameter values (e.g., a range of learning rates), or use more advanced techniques such as Bayesian optimization (Snoek et al., 2012) and model-based approaches (Bergstra et al., 2013; Hutter et al., 2013). An interesting line of research, which also inspired our approach in this paper, is to use gradient-based optimization for the tuning of hyperparameters (Bengio, 2000). Recent work in this area include reversible learning (Maclaurin et al., 2015), which allows gradient-based optimization of hyperparameters through a training run consisting of multiple iterations, and hypergradient descent (Baydin et al., 2018), which achieves a similar optimization in an online, per-gradient-update, fashion.

Meta-learning is often referred to as "learning to learn" (Thrun and Pratt, 2012), meaning a learning procedure (most of the time gradient-based) is able to improve aspects of the learning process itself, such as the optimizer, hyperparameters like the learning rate, and initializations. In this sense, the "meta" concept of meta-learning has aspects in common with hyperparameter optimization. The MAML model (Finn et al., 2017b) on which we base our method, relies on meta-optimization through gradient descent in a model-agnostic way. Another recent method, Meta-SGD

(Li et al., 2017), performs online optimization of a per-parameter learning rate vector α, to which the authors refer as learning both the learning rate and update direction (because of the per-parameter nature being able to modify direction), and model parameters θ, using a single hyper-learning rate β. Our work differs from Meta-SGD as we perform simultaneous online optimization of both MAML learning rates α and β, which are both scalars.

6.3 Model-Agnostic Meta-Learning (MAML)

The MAML algorithm, given model parameters θ, aims to adapt to a new task $\mathcal{T}_t$ with SGD:

$$\theta'_t = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_{train(t)}}(f_\theta) \ , \tag{6.1}$$

where t is the task number and α is the learning rate. $T_{train(t)}$ and $T_{test(t)}$ denote the training and test set within task t. The tasks are sampled from a defined $p(\mathcal{T})$. The meta-objective is:

$$\min_\theta \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t}) = \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_{train(t)}}(f_\theta)}) \tag{6.2}$$

The model aims to optimize the parameters θ such that with just one SGD step it can adapt to the new task. For the optimization in Eq. 6.2, this looks as follows:

$$\theta_t = \theta - \beta \nabla_\theta \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t}) \ , \tag{6.3}$$

where β is the meta step size. This gives an algorithm that learns an initialization of θ that is useful for being adapted to new tasks efficiently with a small number of iterations. An important advantage is that this is achieved without making any assumptions on the form of the model. Another advantage is that the meta-learner, introduced in this way based on conventional gradient descent, does not introduce extra model parameters to learn as in model-based approaches in meta-learning. The full algorithm is shown in Algorithm 3. It can be seen that unlike usual SGD which has only one learning rate, MAML has two learning rates α and β, which require time-consuming hyperparameter tuning.

Algorithm 3 MAML

Input: $p(\mathcal{T})$: distribution over tasks.
Input: α, β : learning rates
randomly initialize θ.
while not done **do**
  Sample batch of tasks $\mathcal{T}_t \sim p(\mathcal{T})$
  for all $\mathcal{T}_t$ **do**
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_{train(t)}}(f_\theta)$ with respect to K examples.
    Compute adapted parameters with gradient descent: $\theta'_t = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_{train(t)}}(f_\theta)$
  end for
  Update $\theta = \theta - \beta \nabla_\theta \sum_{\mathcal{T}_t \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t})$
end while

Algorithm 4 Alpha MAML

Input: $p(\mathcal{T})$: distribution over tasks.
Input: α_0, β_0 : initial learning rates
Input: $\alpha_{\text{hyperlr}}, \beta_{\text{hyperlr}}$: hypergradient learning rates
randomly initialize θ.
while not done **do**
  Sample batch of tasks $\mathcal{T}_t \sim p(\mathcal{T})$
  for all $\mathcal{T}_t$ **do**
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_{train(t)}}(f_\theta)$ with respect to K examples.
    Compute adapted parameters with gradient descent: $\theta'_t = \theta - \alpha_i \nabla_\theta \mathcal{L}_{\mathcal{T}_{train(t)}}(f_\theta)$
  end for
  $\alpha_{i+1} = \alpha_i + \alpha_{\text{hyperlr}} \sum_{\mathcal{T}_t \sim p(\mathcal{T})} \nabla_{\theta'_t} \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t}) . \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{train(t)}}(f_{\theta_{i-1}})$
  $\beta_i = \beta_{i-1} + \beta_{\text{hyperlr}} \sum_{\mathcal{T}_t \sim p(\mathcal{T})} \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t}) . \nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_t})$
  $\theta_i = \theta_{i-1} - \beta_i \sum_{\mathcal{T}_t \sim p(\mathcal{T})} \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t})$
end while

6.4 Alpha MAML

Eq. 6.1 is regular gradient descent for adapting to the task t, with α being the task-level learning rate. Similarly, Eq. 6.3 is regular gradient descent for the meta update, with β being the meta-learning rate. In addition to the update rules in Equations 6.1 and 6.3, we would like to derive update rules for the learning rates α and β as well. Our algorithm, Alpha MAML, can simply be written in four update equations as shown in the following derivation. Here i is the iteration number. We first derive the algorithm for the simpler case when each batch has only one task, thus the iteration number is same as the task number $t = i$.

Firstly, our goal is to update the value of α towards the optimum value α_i^* that minimizes the value of the meta objective Eq. 6.2 in the next iteration. However,

we have not computed θ'_i yet. If we assume that the optimal value of α does not change much across iterations, we can estimate it by α^*_{i-1}. Therefore we perform one gradient descent step over the previous value of learning rate α_{i-1}, with gradient:

$$\begin{aligned}\frac{\partial \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}})}{\partial \alpha} &= \frac{\partial \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}})}{\partial \theta'_{i-1}}^T \frac{\partial \theta'_{i-1}}{\partial \alpha} \\ &= \nabla_{\theta'_{i-1}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}})^T \\ &\qquad \frac{\partial(\theta_{i-2} - \alpha_{i-1} \nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{train(i-1)}}(f_{\theta_{i-2}}))}{\partial \alpha} \\ &= \nabla_{\theta'_{i-1}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}}).(-\nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{train(i-1)}}(f_{\theta_{i-2}}))\end{aligned} \tag{6.4}$$

We can estimate α^*_{i-1} as follows:

$$\alpha_i = \alpha_{i-1} + \alpha_{\text{hyperlr}} \nabla_{\theta'_{i-1}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}}) . \nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{train(i-1)}}(f_{\theta_{i-2}}) \ , \tag{6.5}$$

where α_{hyperlr} is the hyper learning rate for α.

Secondly, we also want to derive an update rule for the meta learning rate β. Similar to α, we would like to update β towards its optimal value β^*_i that minimizes the value of the objective $\mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i})$ in the next iteration, making an assumption that the optimal value of β does not change much between two consecutive iterations. For this, we compute:

$$\begin{aligned}\frac{\partial \mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i})}{\partial \beta} &= \frac{\partial \mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i})}{\partial \theta_{i-1}} . \frac{\partial \theta_{i-1}}{\partial \beta} \\ &= \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i}).(-\nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}}))\end{aligned} \tag{6.6}$$

We can estimate β^*_{i-1} as follows:

$$\beta_i = \beta_{i-1} + \beta_{\text{hyperlr}} \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i}) . \nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}}) \ , \tag{6.7}$$

where β_{hyperlr} is the hyper learning rate for β. It is important to note that in Eq.6.4 and Eq.6.6, the gradients are computed with respect to the loss on the test set of a batch. This is done in accordance with the meta-objective. The final Alpha MAML algorithm can thus be written down into just 4 update equations:

$$\begin{aligned}\theta'_i &= \theta_{i-1} - \alpha_i \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{train(i)}}(f_{\theta_{i-1}}) \\ \alpha_{i+1} &= \alpha_i + \alpha_{\text{hyperlr}} \nabla_{\theta'_i} \mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i}) . \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{train(i)}}(f_{\theta_{i-1}}) \\ \beta_i &= \beta_{i-1} + \beta_{\text{hyperlr}} \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i}) . \nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_{i-1}}) \\ \theta_i &= \theta_{i-1} - \beta_i \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(i)}}(f_{\theta'_i})\end{aligned} \tag{6.8}$$

It can be seen that no extra gradient needs to be computed, as the gradients from the last iteration can be used, requiring only the extra memory storage of the gradient from the previous iteration. The full algorithm with the more general case of multiple tasks in one batch is shown in Algorithm 4. The derivation for this case is shown in the appendix E.1.

6.5 Experiments

To evaluate the performance of Alpha MAML in comparison to MAML, we perform experiments on the few-shot image recognition task on the Omniglot dataset (Lake et al., 2011), which is commonly used in related work (Finn et al., 2017b; Ravi and Larochelle, 2017; Snell et al., 2017a; Vinyals et al., 2016a). We keep the experimentation configuration the same as the original MAML work, for a fair comparison. Omniglot comprises 20 instances (drawn by 20 different people) of 1,623 characters from 50 alphabets. We follow the N-way Omniglot task setup, which was introduced by Vinyals et al. (2016a), and also used in MAML (Finn et al., 2017b). The N-way classification task is set up as follows: the model is shown K instances from N unseen classes, and then evaluated on some other instances from these N classes. To keep the procedure the same as MAML, we also randomly select 1,200 characters for training, and use the rest for testing. The network architecture follows the architecture used by Finn et al. (2017b), and more implementation details are in the appendix E.1.

Behaviour of Alpha MAML vs MAML: Here we choose a good case (where the MAML user picked a good pair of α_0 and β_0, i.e., the tuned case) and a bad case (where the user picked a bad pair of α_0 and β_0, i.e., an untuned case), and plot the evolution of α_i and β_i as a function of iterations, also showing the training and validation losses during training. Figure 6.1 shows the evolution of learning rate α_i, meta-learning rate β_i, and the training and validation losses. Results show that even the badly picked α_0 and β_0 values can be automatically tuned by the online learning rate adaptation scheme in Alpha MAML, tuned in the sense that the algorithm does

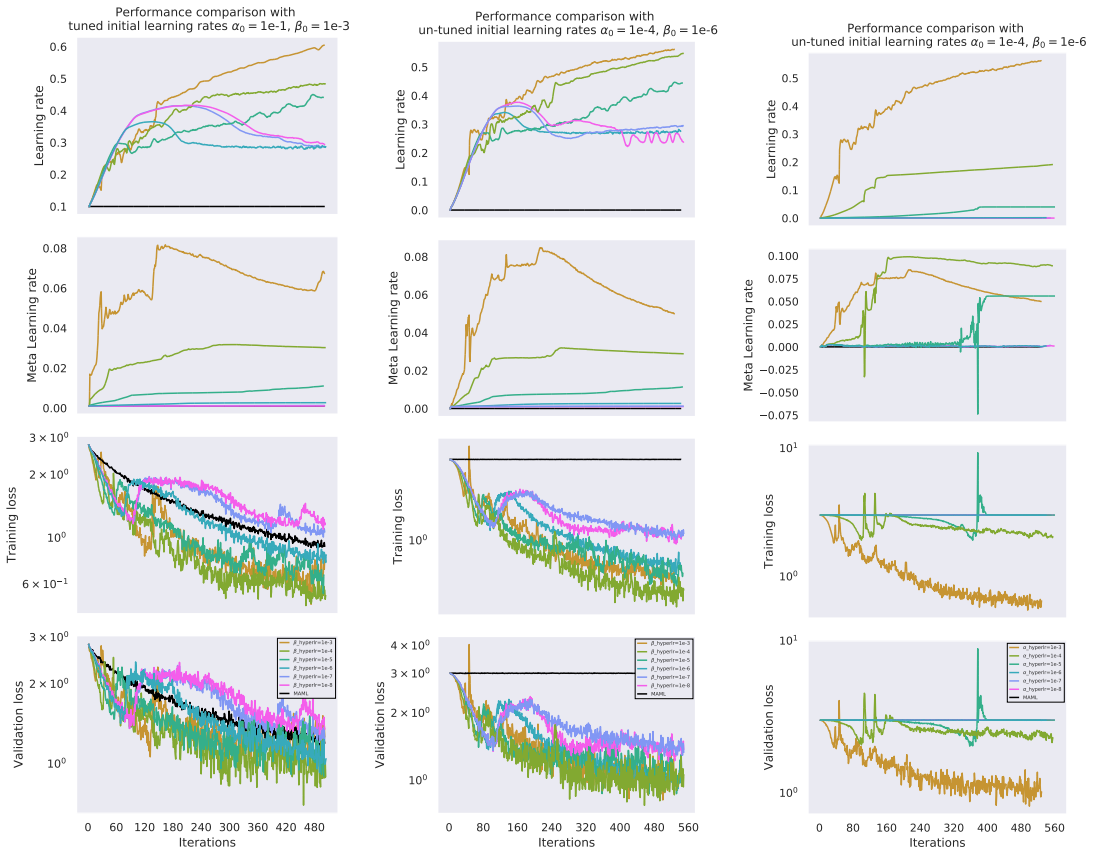


Figure 6.1: Convergence comparison of Alpha-MAML and MAML, with and without tuned initial learning rate hyper-parameters α_0 and β_0. Columns: *left:* tuned initial learning rates; *middle:* untuned initial learning rates, with varying β_{hyperlr} for Alpha-MAML; *right:* untuned initial learning rates, with varying α_{hyperlr} for Alpha-MAML. Rows: *first:* evolution of the learning rate α; *second:* evolution of the meta learning rate β; *third:* training loss; *fourth:* validation loss.

the necessary adjustments to α_i and β_i in each iteration to achieve a loss similar to the good case. This can be seen in Figure 6.1 *middle* and *right* columns.

Insensitivity with respect to hyperparameter choices: Here we run a series of training experiments to study the effect of initial learning rate values on the number of iterations needed for the algorithms to reach a particular chosen loss threshold. Figures 6.2 and E.1 show this grid search for tuning the learning rate hyper-parameters. It can be seen that MAML shows very slow convergence for a range of initial learning rates, more specifically for $\alpha_0 = 1e-4, \beta_0 = 1e-6$ and $\alpha_0 = 1e-5, \beta_0 = 1e-7$. In comparison, Alpha MAML shows comparatively faster convergence for these initial learning rates also, for a wide range of α_{hyperlr} and β_{hyperlr}.

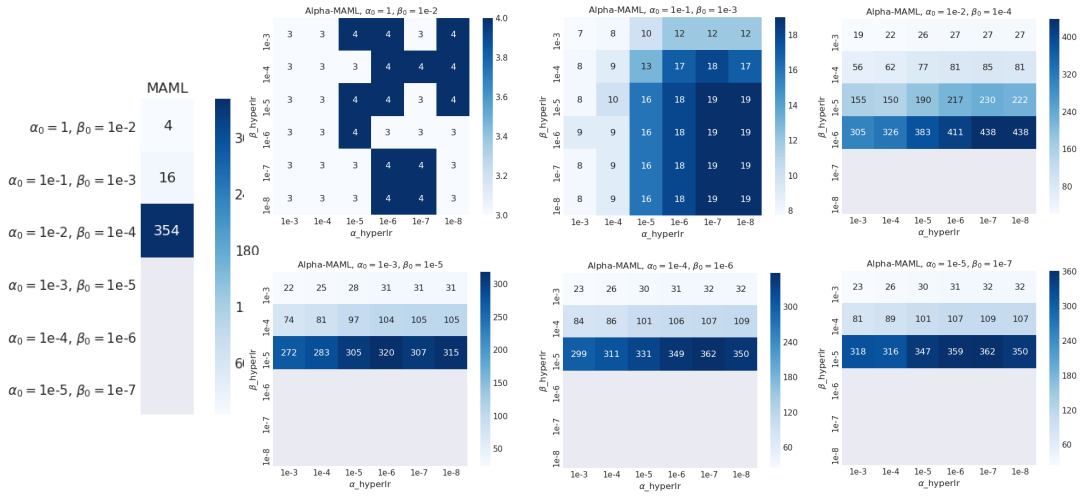


Figure 6.2: Grid Search for selecting the hyper-parameters in MAML and Alpha-MAML: shows the number of iterations to converge to a training loss of 2.5 for Omniglot dataset. Here the blank cells denote the cases where algorithm did not converge till 500 iterations. It can be seen that as we go far from the tuned hyper-parameter range, converge of Alpha-MAML is better than MAML. Specially for the case of $\alpha_0 = 1e-3, \beta_0 = 1e-5$ Alpha-MAML converged for some cases of α_{hyperlr} and β_{hyperlr}, whereas MAML does not converge till 500 iterations. Same is the case with $\alpha_0 = 1e-4, \beta_0 = 1e-6$, for half of the choices of α_{hyperlr} and β_{hyperlr}, Alpha-MAML converged but MAML does not converge in less than 500 iterations.

For the cases where MAML shows fast convergence, Alpha MAML also shows fast convergence for all values of α_{hyperlr} and β_{hyperlr}. This indicates that in practice no matter which values one chooses for the initial learning and meta-learning rates α_0 and β_0, Alpha MAML always shows faster, or in the worst case the same, convergence as MAML. In other words, Alpha MAML is less sensitive to the hyperparameter choice as compared to MAML, and hence needs significantly less tuning.

6.6 Conclusion

Meta-learning is an important and currently relevant approach with potential impact in solving hard problems in computer vision and reinforcement learning. The training instability of meta-learning algorithms as a function of hyperparameter choices is a known shortcoming and currently an active area of research. We have proposed an extension of the state-of-the-art MAML algorithm (Finn et al., 2017b) based on the application of the hypergradient descent technique (Baydin et al.,

2018) to make MAML training more robust to hyperparameter choices.

7 Summary and Discussion

Contents

7.1 Summary

There are many challenges that stand in the way of widespread deployment of deep learning. This thesis deals with two of these challenges, namely, neural network verification and automated machine learning. We formulated both these problems using the framework of *optimization*, and built on the existing knowledge of optimization to provide solutions to these challenges. We first summarise our contributions and then discuss extensions.

In chapter 2, we designed an efficient bounding algorithm for the convex relaxation of Tjandraatmadja et al. (2020). Since the relaxation has an exponential number of constraints, the existing algorithms are not directly applicable. We designed a sparse dual algorithm by keeping an active set of dual variables. This allowed us to attain better bounds than off-the-shelf solvers for the relaxation, in an order of magnitude less time.

In chapter 3, we proposed a tighter relaxation for the neural network verification problem with simplex inputs, which describes the convex hull of the composition of a linear function with an element-wise convex non-linearity. In contrast to a similar relaxation (Tjandraatmadja et al., 2020) for verification with ℓ_∞ inputs which requires an exponential number of constraints, our relaxation only requires constraints linear in the dimensionality of the input simplex. This allowed us to design more efficient bounding algorithms for the problem.

In chapter 4, we proposed a bi-level algorithm for optimizing simulators with the objective of generating data that is optimal for training models for a downstream task. Our algorithm can optimize non-differentiable renderers. Unlike black-box methods like Reinforce and Bayesian optimization, our algorithm exploits the structure of the problem explicitly. Thereby requiring only a single objective evaluation at each iteration and is much faster than these methods.

In chapter 5, we proposed a meta-learning based algorithm for video segmentation. Unlike previous methods which use surrogate losses, we ensure that the training objective matches the inference procedure. We demonstrated state-of-the-art performance on multiple video segmentation datasets.

In chapter 6, we proposed an extension of model-agnostic meta-learning (MAML) to overcome its shortcoming of training instability as a function of learning rates. We incorporated an online hyperparameter adaptation scheme that eliminates the need to tune meta-learning and learning rates. We demonstrated a substantial improvement in training stability.

7.2 Discussion

In this section, we present a critical discussion of the different methods presented in this thesis.

Scaling the Convex Barrier with Active Sets Convex barrier is a term introduced by Salman et al. (2019) to define the gap between the optimal value of the original verification problem and the optimal convex relaxation of the non-linearity. Anderson et al. (2019b) proposed a relaxation with the potential to overcome this barrier. We proposed a dual solver to realise this potential in an efficient manner in both incomplete and complete verification, thereby scaling the convex barrier.

After our work, Wang et al. (2021) showed that even propagation-based algorithms can be very useful and efficient for complete verification. While subsequent work (Palma et al., 2021) has improved the performance of dual algorithms, propagation-based algorithms still remain more effective. Another benefit of propagation based algorithms is that they require less hyper-parameter tuning than Lagrangian dual solvers.

It remains to explore whether a propagation-based approach for the tighter relaxation by Anderson et al. (2019b) can be helpful in the context of complete verification. This would involve utilizing the constraints from the Anderson relaxation (Anderson et al., 2019b) in the formulation of Wang et al. (2021), as we did in our next work (Behl et al., 2021).

Overcoming the Convex Barrier for Simplex Inputs Taking inspiration from the work of Anderson et al. (2019b), we proposed a tighter relaxation for verification with simplex inputs. We also proposed a technique to propagate simplex constraints through different layers to use the tighter relaxation. A promising direction is to write the tighter relaxation as an explicit function of the propagated simplex, and jointly optimise the simplex propagation along with the final layer bounds. This would also allow jointly optimising the intermediate bounds.

We believe that our work can be used as a component for building relaxations for self-attention layers, but not in isolation. The output of the self-attention layer will lie in the convex hull of the value vectors. Hence, our framework can be used as part of a tight relaxation for the self-attention layers. However, it would additionally require relaxations of other non-linearities such as the softmax function and the bilinear function. It would also be interesting to explore whether the same relaxation will provide some gains in the ℓ_∞ setting. This could be done by propagating simplex in that setting too.

AutoSimulate: Learning Synthetic Data Generation We tried to automate the pipeline of using synthetic data for training neural networks, by optimizing renderers. However, there are still certain components which require expert intervention. In the current pipeline, these factors include (i) choosing the distribution over the simulator parameters, (ii) human expertise needed to design 3d models of objects.

We have tried to address the second limitation in a recent work that was submitted to CVPR 2022. This involved using NeRF (Mildenhall et al., 2020) as the data generator instead of a traditional graphics renderer. However, the first challenge still remains open. We believe that a hybrid approach using a combination of deep generative models like VAEs/GANs and NeRF could be helpful for this.

Meta Learning Deep Visual Words for Fast Video Object Segmentation We formulated the problem of video segmentation as a meta learning problem and proposed a causal approach for the problem by constructing visual words. However, the number of visual words was kept as a hyperparameter which needs tuning. An

extension would involve exploring ways to remove the need for this tuning, for e.g., by using the Chinese restaurant process (Murphy, 2012).

More recently, transformer based architectures have been shown to be effective for segmentation (Chen et al., 2021). It would be interesting to explore using our framework with a transformer backbone. Another direction would be to try to back-propagate through the clustering or use an end-to-end alternative instead.

Alpha MAML: Adaptive Model-Agnostic Meta-Learning We proposed a scheme to automatically tune the learning rates of gradient based meta-learning algorithms by taking inspiration from Baydin et al. (2018). It would be interesting to explore other developments in adaptive learning methods for training neural networks (Berrada et al., 2019a,b). We can also leverage the recent research in deep bilevel optimization (Lorraine et al., 2020).

A significant limitation of the MAML algorithm itself is the assumption that the inner problem would only involve a few gradient steps. This approximation is quite loose for large scale problems and Rajeswaran et al. (2019), thus, proposed to use implicit gradients. This is related to our AutoSimulate work (Behl et al., 2020a) where we use a second order approximation for the inner problem. We had also proposed more efficient alternatives for this implicit gradient, which can be explored in this context.

Appendices

A

Scaling the Convex Barrier with Active Sets

Contents

A.1 Limitations of Previous Dual Approaches

In this section, we show that previous dual derivations (Bunel et al., 2020a; Dvijotham et al., 2018) violate Fact 1. Therefore, they are not efficiently applicable to problem equation 2.3, motivating our own derivation in section 2.3.

We start from the approach by Dvijotham et al. (2018), which relies on relaxing equality constraints equation 2.1b, equation 2.1c from the original non-convex problem equation 2.1. Dvijotham et al. (2018) prove that this relaxation corresponds to solving convex problem equation 2.2, which is equivalent to the Planet relaxation (Ehlers, 2017), to which the original proof refers. As we would like to solve tighter problem equation 2.3, the derivation is not directly applicable. Relying on intuition from convex analysis applied to duality gaps (Lemaréchal, 2001), we conjecture that relaxing the composition equation 2.1c $\circ$ equation 2.1b might tighten the primal problem equivalent to the relaxation, obtaining the following dual:

$$\begin{aligned} \max_{\boldsymbol{\mu}} \min_{\mathbf{x}} \quad & W_n \mathbf{x}_{n-1} + \mathbf{b}_n + \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T \left(\mathbf{x}_k - \max\left\{ W_k \mathbf{x}_{k-1} + \mathbf{b}_k, 0 \right\} \right) \\ \text{s.t.} \quad & \mathbf{l}_k \leq \mathbf{x}_k \leq \mathbf{u}_k \qquad\qquad k \in [\![1, n-1]\!], \\ & \mathbf{x}_0 \in \mathcal{C}. \end{aligned} \tag{A.1}$$

Unfortunately dual equation A.1 requires an LP (the inner minimisation over $\mathbf{x}$, which in this case does not decompose over layers) to be solved exactly to obtain a supergradient and any time a valid bound is needed. This is markedly different from the original dual by Dvijotham et al. (2018), which had an efficient closed-form for the inner problems.

The derivation by Bunel et al. (2020a), instead, operates by substituting equation 2.1c with its convex hull and solving its Lagrangian Decomposition dual. The Decomposition dual for the convex hull of equation 2.1c $\circ$ equation 2.1b (i.e., $\mathcal{A}_k$) takes the following form:

$$\begin{aligned} \max_{\boldsymbol{\rho}} \min_{\mathbf{x},\mathbf{z}} \; & W_n \mathbf{x}_{A,n-1} + \mathbf{b}_n + \sum_{k=1}^{n-1} \boldsymbol{\rho}_k^T \left(\mathbf{x}_{B,k} - \mathbf{x}_{A,k} \right) \\ \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\ & (\mathbf{x}_{B,k}, W_n \mathbf{x}_{A,n-1} + \mathbf{b}_n, \mathbf{z}_k) \in \mathcal{A}_{\text{dec},k} \qquad k \in [\![1, n-1]\!], \end{aligned} \tag{A.2}$$

where $\mathcal{A}_{\text{dec},k}$ corresponds to $\mathcal{A}_k$ with the following substitutions: $\mathbf{x}_k \to \mathbf{x}_{B,k}$, and $\hat{\mathbf{x}}_k \to W_n \mathbf{x}_{A,n-1} + \mathbf{b}_n$. It can be easily seen that the inner problems (the inner minimisation over $\mathbf{x}_{A,k}, \mathbf{x}_{B,k}$, for each layer $k > 0$) are an exponentially sized LP. Again, this differs from the original dual on the Planet relaxation (Bunel et al., 2020a), which had an efficient closed-form for the inner problems.

A.2 Dual Initialisation

Algorithm 5 Big-M solver

function BIGM_COMPUTE_BOUNDS($\{W_k, \mathbf{b}_k, \mathbf{l}_k, \mathbf{u}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k\}_{k=1..n}$)
  Initialise duals $\boldsymbol{\alpha}^0, \boldsymbol{\beta}^0_{\mathcal{M}}$ using interval propagation bounds (Gowal et al., 2018b)
  for $t \in [\![1, T-1]\!]$ **do**
    $\mathbf{x}^*, \mathbf{z}^* \in \arg\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^t, \boldsymbol{\beta}^t_{\mathcal{M}})$ using equation A.5-A.6
    Compute supergradient using equation A.7
    $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{M}} \leftarrow$ Adam's update rule (Kingma and Ba, 2015)
    $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{M}} \leftarrow \max(\boldsymbol{\alpha}^{t+1}, 0), \max(\boldsymbol{\beta}^{t+1}_{\mathcal{M}}, 0)$ (dual projection)
  end for
  return $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \boldsymbol{\beta}^T_{\mathcal{M}})$
end function

As shown in section 2.3, our Active Set solver yields a dual solver for the Big-M relaxation equation 2.2 if the active set $\mathcal{B}$ is kept empty throughout execution. As indeed $\mathcal{B} = \emptyset$ for the first Active Set iterations (see algorithm 6 in section A.4), the Big-M solver can be thought of as dual initialisation. Furthermore, we demonstrate experimentally in §2.5 that, when used as a stand-alone solver, our Big-M solver is competitive with previous dual algorithms for problem equation 2.2. The goal of this section is to explicitly describe the Big-M solver, which is summarised in algorithm 5. We point out that, in the notation of restricted variable sets from section 2.3.1, $\boldsymbol{\beta}_{\mathcal{M}} := \boldsymbol{\beta}_{\emptyset}$.

We now describe the equivalence between the Big-M and Planet relaxations, before presenting the solver in section A.2.3 and the dual it operates on in section A.2.2.

A.2.1 Equivalence to Planet

As previously shown (Bunel et al., 2018), the Big-M relaxation ($\mathcal{M}_k$, when considering the k-th layer only) in problem equation 2.2 is equivalent to the Planet relaxation by Ehlers (2017). Then, due to strong duality, our Big-M solver (section A.2.2) and the solvers by Bunel et al. (2020a); Dvijotham et al. (2018) will all converge to the bounds from the solution of problem equation 2.2. In fact, the Decomposition-based method (Bunel et al., 2020a) directly operates on the Planet relaxation, while Dvijotham et al. (2018) prove that their dual is equivalent to doing so.

On the k-th layer, the Planet relaxation takes the following form:

$$\mathcal{P}_k := \begin{cases} \text{if } \hat{\mathbf{l}}_k \leq 0 \text{ and } \hat{\mathbf{u}}_k \geq 0: & \\ & \mathbf{x}_k \geq 0, \quad \mathbf{x}_k \geq \hat{\mathbf{x}}_k, \\ & \mathbf{x}_k \leq \hat{\mathbf{u}}_\mathbf{k} \odot (\hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k) \odot \left(1/(\hat{\mathbf{u}}_\mathbf{k} - \hat{\mathbf{l}}_\mathbf{k})\right). \\ \text{if } \hat{\mathbf{u}}_k \leq 0: & \\ & \mathbf{x}_k = 0. \\ \text{if } \hat{\mathbf{l}}_k \geq 0: & \\ & \mathbf{x}_k = \hat{\mathbf{x}}_k. \end{cases} \tag{A.3}$$

It can be seen that $\mathcal{P}_k = \text{Proj}_{\mathbf{x},\hat{\mathbf{x}}}(\mathcal{M}_k)$, where $\text{Proj}_{\mathbf{x},\hat{\mathbf{x}}}$ denotes projection on the $\mathbf{x}, \hat{\mathbf{x}}$ hyperplane. In fact, as $\mathbf{z}_k$ does not appear in the objective of the primal formulation equation 2.2, but only in the constraints, this means assigning it the value that allows the largest possible feasible region. This is trivial for passing or blocking ReLUs. For the ambiguous case, instead, Figure A.1 (on a single ReLU) shows that $z_k = \frac{\hat{x}_k - \hat{l}_k}{\hat{u}_k - \hat{l}_k}$ is the correct assignment.

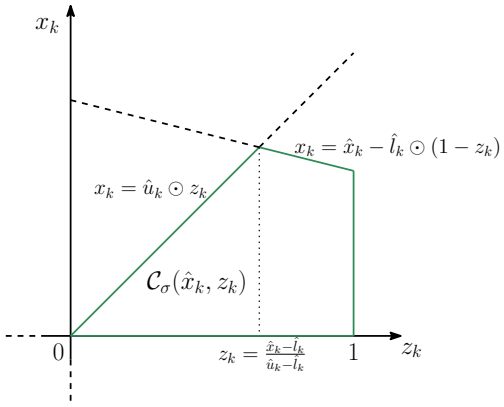


Figure A.1: $\mathcal{M}_k$ plotted on the $(\mathbf{z}_k, \mathbf{x}_k)$ plane, under the assumption that $\hat{\mathbf{l}}_k \leq 0$ and $\hat{\mathbf{u}}_k \geq 0$.

A.2.2 Big-M Dual

As evident from problem equation 2.3, $\mathcal{A}_k \subseteq \mathcal{M}_k$. If we relax all constraints in $\mathcal{M}_k$ (except, again, the box constraints), we are going to obtain a dual with a strict subset of the variables in problem equation 2.6. The Big-M dual is a specific instance of the Active Set dual equation 2.8 where $\mathcal{B} = \emptyset$, and it takes the following form:

$$\begin{aligned}
&\max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\geq 0} d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) \quad \text{where:} \qquad d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) := \min_{\mathbf{x},\mathbf{z}} \; \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}), \\
&\mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) = \begin{bmatrix} -\sum_{k=0}^{n-1} \left(\boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - (\boldsymbol{\beta}_{k,0} + \boldsymbol{\beta}_{k,1} - W_{k+1}^T \boldsymbol{\beta}_{k+1,1})\right)^T \mathbf{x}_k \\ +\sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k - \sum_{k=1}^{n-1} \left(\boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k\right)^T \mathbf{z}_k \\ +\sum_{k=1}^{n-1} (\hat{\mathbf{l}}_k - \mathbf{b}_k)^T \boldsymbol{\beta}_{k,1} \end{bmatrix} \\
&\text{s.t.} \qquad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad k \in [\![1, n-1]\!].
\end{aligned} \tag{A.4}$$

A.2.3 Big-M solver

We initialise dual variables from interval propagation bounds (Gowal et al., 2018b): this can be easily done by setting all dual variables except $\boldsymbol{\alpha}_n$ to 0. Then, we can maximize $d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ via projected supergradient ascent, exactly as described in section 2.3.1 on a generic active set $\mathcal{B}$. All the computations in the solver follow from keeping $\mathcal{B} = \emptyset$ in §2.3.1. We explicitly report them here for the reader's convenience.

Let us define the following shorthand for the primal coefficients:

$$\begin{aligned}
\boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) &= \left(\boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - (\boldsymbol{\beta}_{k,0} + \boldsymbol{\beta}_{k,1} - W_{k+1}^T \boldsymbol{\beta}_{k,1})\right) \\
\boldsymbol{g}_{\mathcal{M},k}(\boldsymbol{\beta}_{\mathcal{M}}) &= \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k.
\end{aligned}$$

The minimisation of the Lagrangian $\mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ over the primals for $k \in [\![1, n-1]\!]$ is as follows:

$$\mathbf{x}_k^* = \mathbb{1}_{\boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{M}})\geq 0} \odot \hat{\mathbf{u}}_k + \mathbb{1}_{\boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{M}})<0} \odot \hat{\mathbf{l}}_k \qquad \mathbf{z}_k^* = \mathbb{1}_{\boldsymbol{g}_{\mathcal{M},k}(\boldsymbol{\beta}_{\mathcal{M}})\geq 0} \odot \mathbf{1} \tag{A.5}$$

For $k = 0$, instead (assuming, as §2.3.1 that this can be done efficiently):

$$\mathbf{x}_0^* \in \arg\min_{\mathbf{x}_0} \quad \boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}})^T \mathbf{x}_0 \qquad \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}. \tag{A.6}$$

The supergradient over the Big-M dual variables $\boldsymbol{\alpha}, \boldsymbol{\beta}_{k,0}, \boldsymbol{\beta}_{k,1}$ is computed exactly as in §2.3.1 and is again a subset of the supergradient of the full dual problem equation 2.6. We report it for completeness. For each $k \in [\![0, n-1]\!]$:

$$\begin{gathered}\nabla_{\boldsymbol{\alpha}_k} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = W_k \mathbf{x}^*_{k-1} + \mathbf{b}_k - \mathbf{x}^*_k, \quad \nabla_{\beta_{k,0}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{x}_k - \mathbf{z}_k \odot \hat{\mathbf{u}}_k, \\ \nabla_{\beta_{k,1}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{x}_k - (W_k \mathbf{x}_{k-1} + \mathbf{b}_k) + (1 - \mathbf{z}_k) \odot \hat{\mathbf{l}}_k.\end{gathered} \tag{A.7}$$

A.3 Dual Derivations

We now derive problem equation 2.6, the dual of the full relaxation by Anderson et al. (2020) described in equation equation 2.3. The Active Set (equation equation 2.8) and Big-M duals (equation equation A.4) can be obtained by removing $\boldsymbol{\beta}_{k,I_k} \forall\ I_k \in \mathcal{E}_k \setminus \mathcal{B}_k$ and $\boldsymbol{\beta}_{k,I_k} \forall\ I_k \in \mathcal{E}_k$, respectively. We employ the following Lagrangian multipliers:

$$\begin{aligned}\mathbf{x}_k &\geq \hat{\mathbf{x}}_k \Rightarrow \boldsymbol{\alpha}_k, \\ \mathbf{x}_k &\leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k \Rightarrow \boldsymbol{\beta}_{k,0}, \\ \mathbf{x}_k &\leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k) \Rightarrow \boldsymbol{\beta}_{k,1}, \\ \mathbf{x}_k &\leq \begin{pmatrix} (W_k \odot I_k)\, \mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k \\ -\left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k) \\ +\left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \end{pmatrix} \Rightarrow \boldsymbol{\beta}_{k,I_k},\end{aligned}$$

and obtain, as a Lagrangian (using $\hat{\mathbf{x}}_k = W_k \mathbf{x}_{k-1} + \mathbf{b}_k$):

$$\begin{aligned}&\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \\ &\left[\begin{array}{l} \sum_{k=1}^{n-1} \boldsymbol{\alpha}_k^T \left(W_k \mathbf{x}_{k-1} + \mathbf{b}_k - \mathbf{x}_k\right) + \sum_{k=1}^{n-1} \boldsymbol{\beta}_{k,0}^T \left(\mathbf{x}_k - \mathbf{z}_k \odot \hat{\mathbf{u}}_k\right) \\ + \sum_{k=1}^{n-1} \sum_{I_k \in \mathcal{E}_k} \boldsymbol{\beta}_{k,I_k}^T \begin{pmatrix} \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k) - (W_k \odot I_k)\, \mathbf{x}_{k-1} \\ -\mathbf{b}_k \odot \mathbf{z}_k - \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k + \mathbf{x}_k \end{pmatrix} \\ + \sum_{k=1}^{n-1} \boldsymbol{\beta}_{k,1}^T \left(\mathbf{x}_k - (W_k \mathbf{x}_{k-1} + \mathbf{b}_k) + (1 - \mathbf{z}_k) \odot \hat{\mathbf{l}}_k\right) + W_n \mathbf{x}_{n-1} + \mathbf{b}_n \end{array}\right.\end{aligned} \tag{A.8}$$

Let us use $\sum_{I_k}$ as shorthand for $\sum_{I_k \in \mathcal{E}_k \cup \{0,1\}}$. If we collect the terms with respect to the primal variables and employ dummy variables $\boldsymbol{\alpha}_0 = 0$, $\boldsymbol{\beta}_0 = 0$, $\boldsymbol{\alpha}_n = I$, $\boldsymbol{\beta}_n = 0$, we obtain:

$$\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) =$$
$$\begin{bmatrix} -\sum_{k=0}^{n-1} \begin{pmatrix} \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \sum_{I_k} \boldsymbol{\beta}_{k,I_k} \\ +\sum_{I_{k+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}} \end{pmatrix}^T \mathbf{x}_k \\ -\sum_{k=1}^{n-1} \begin{pmatrix} \sum_{I_k \in \mathcal{E}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k \\ +\sum_{I_k \in \mathcal{E}_k} \left(W_k \odot I_k \odot \check{L}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} \\ +\sum_{I_k \in \mathcal{E}_k} \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} \end{pmatrix}^T \mathbf{z}_k \\ +\sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k + \sum_{k=1}^{n-1} \left(\sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \square \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k) \right) \end{bmatrix} \tag{A.9}$$

which corresponds to the form shown in problem equation 2.6.

A.4 Implementation Details for Active Set Method

From the high-level perspective, our Active Set solver proceeds by repeatedly solving modified instances of problem equation 2.6, where the exponential set $\mathcal{E}_k$ is replaced by a fixed (small) set of active variables $\mathcal{B}$. The full solver procedure is summarised in algorithm 6.

Algorithm 6 Active Set solver

function ACTIVESET_COMPUTE_BOUNDS($\{W_k, \mathbf{b}_k, \mathbf{l}_k, \mathbf{u}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k\}_{k=1..n}$)
  Initialise duals $\boldsymbol{\alpha}^0, \boldsymbol{\beta}_0^0, \boldsymbol{\beta}_1^0$ using Algorithm equation 5
  Set $\boldsymbol{\beta}_{k,I_k} = 0, \forall\ I_k \in \mathcal{E}_k$
  $\mathcal{B} = \emptyset$
  for `nb_additions` **do**
    for $t \in [\![1, T-1]\!]$ **do**
      $\mathbf{x}^*, \mathbf{z}^* \in \arg\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^t, \boldsymbol{\beta}_{\mathcal{B}}^t)$ using equation 2.9,equation 2.10
      if $t \leq$ `nb_vars_to_add` **then**
        For each layer k, add output of equation 2.4 called at $(\mathbf{x}^*, \mathbf{z}^*)$ to $\mathcal{B}_k$
      end if
      Compute supergradient using equation 2.11
      $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}_{\mathcal{B}}^{t+1} \leftarrow$ Adam's update rule (Kingma and Ba, 2015)
      $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}_{\mathcal{B}}^{t+1} \leftarrow \max(\boldsymbol{\alpha}^{t+1}, 0), \max(\boldsymbol{\beta}_{\mathcal{B}}^{t+1}, 0)$ (dual projection)
    end for
  end for
  return $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \boldsymbol{\beta}_{\mathcal{B}}^T)$
end function

We conclude this section by proving the primal interpretation of the selection criterion for adding a new set of variables to $\mathcal{B}$.

A.4.1 Active Set Selection Criterion

We map a restricted set of dual variables $\boldsymbol{\beta}_{\mathcal{B}}$ to a set of dual variables $\boldsymbol{\beta}$ for the full dual equation 2.6 by setting variables not in the active set to 0: $\boldsymbol{\beta}_{\bar{\mathcal{B}}} = 0$, and $\boldsymbol{\beta} = \boldsymbol{\beta}_{\mathcal{B}} \cup \boldsymbol{\beta}_{\bar{\mathcal{B}}}$.

Proposition 5. *Let $\boldsymbol{\beta}_{k,I_k^*}$ be the set of dual variables maximising the corresponding entries of the supergradient of the full dual problem equation 2.6: $\boldsymbol{\beta}_{k,I_k^*} \in \arg\max_{\boldsymbol{\beta}_{k,I_k}} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}\}$. $\boldsymbol{\beta}_{k,I_k^*}$ represents the Lagrangian multipliers associated to the most violated constraints from $\mathcal{A}_k$ at $(\mathbf{x}^*, \mathbf{z}^*) \in \arg\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$, the primal minimiser of the current restricted Lagrangian.*

Proof. In the following, $(\mathbf{x}^*, \mathbf{z}^*)$ denotes the points introduced in the statement. Recall the definition of $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ in equation equation 2.9, which applies beyond the current active set:

$$\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \begin{pmatrix} \mathbf{x}_k^* - (W_k \odot I_k)\, \mathbf{x}_{k-1}^* + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k^*) \\ -\mathbf{z}_k^* \odot \mathbf{b}_k + \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k^* \end{pmatrix} \; I_k \in \mathcal{E}_k.$$

We want to compute $I_k^* \in \arg\max_{I_k} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}\}$, that is:

$$I_k^* \in \arg\max_{I_k \in \mathcal{E}_k} \begin{pmatrix} \mathbf{x}_k^* - (W_k \odot I_k)\, \mathbf{x}_{k-1}^* + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k^*) \\ -\mathbf{z}_k^* \odot \mathbf{b}_k + \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k^* \end{pmatrix}^T \mathbf{1}.$$

By removing the terms that do not depend on I_k, we obtain:

$$\max_{I_k \in \mathcal{E}_k} \begin{pmatrix} -(W_k \odot I_k)\, \mathbf{x}_{k-1}^* + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k^*) \\ +\left(W_k \odot I_k \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k^* \end{pmatrix}^T \mathbf{1}.$$

Let us denote the i-th row of W_k and I_k by $\boldsymbol{w}_{i,k}$ and $\boldsymbol{i}_{i,k}$, respectively, and define $\mathcal{E}_k[i] = 2^{\boldsymbol{w}_{i,k}} \setminus \{0, 1\}$. The optimisation decomposes over each such row: we thus focus on the optimisation problem for the supergradient's i-th entry. Collecting the mask, we get:

$$\max_{\boldsymbol{i}_{i,k} \in \mathcal{E}_k[i]} \sum_j \left(\left((1 - \mathbf{z}_k^*[i]) \odot \check{L}_{k-1}[i,j] + \mathbf{z}_k^*[i] \odot \check{U}_{k-1}[i,j] - \mathbf{x}_{k-1}^*[i] \right) W_k[i,j] \right) I_k[i,j].$$

As the solution to the problem above is obtained by setting $I_k^*[i,j] = 1$ if its coefficient is positive and $I_k^*[i,j] = 0$ otherwise, we can see that the optimal I_k corresponds to calling oracle equation 2.4 by Anderson et al. (2020) on $(\mathbf{x}^*, \mathbf{z}^*)$. Hence, in addition to being the mask associated to $\boldsymbol{\beta}_{k,I_k^*}$, the variable set maximising the supergradient, I_k^* corresponds to the most violated constraint from $\mathcal{A}_k$ at $(\mathbf{x}^*, \mathbf{z}^*)$. □

A.5 Intermediate Bounds

A crucial quantity in both ReLU relaxations ($\mathcal{M}_k$ and $\mathcal{A}_k$) are intermediate pre-activation bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$. In practice, they are computed by solving a relaxation $\mathcal{C}_k$ (which might be $\mathcal{M}_k$, $\mathcal{A}_k$, or something looser) of equation 2.1 over subsets of the network (Bunel et al., 2020a). For $\hat{\mathbf{l}}_i$, this means solving the following problem (separately, for each entry $\hat{\mathbf{l}}_i[j]$):

$$\begin{aligned} \min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \ & \hat{\mathbf{x}}_i[j] \\ \text{s.t.} \ & \mathbf{x}_0 \in \mathcal{C} \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1}, \quad k \in [\![0, i-1]\!] \,, \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in \mathcal{C}_k \qquad\qquad k \in [\![1, i-1]\!] \,. \end{aligned} \tag{A.10}$$

As equation A.10 needs to be solved twice for each neuron (lower and upper bounds, changing the sign of the last layer's weights) rather than once as in equation 2.3, depending on the computational budget, $\mathcal{C}_k$ might be looser than the relaxation employed for the last layer bounds (in our case, $\mathcal{A}_k$). In all our experiments, we compute intermediate bounds as the tightest bounds between the method by Wong and Kolter (2018) and Interval Propagation (Gowal et al., 2018b).

Once pre-activation bounds are available, post-activation bounds can be simply computed as $\mathbf{l}_k = \max(\hat{\mathbf{l}}_k, 0), \mathbf{u}_k = \max(\hat{\mathbf{u}}_k, 0)$.

A.6 Pre-activation Bounds in $\mathcal{A}_k$

We now highlight the importance of an explicit treatment of pre-activation bounds in the context of the relaxation by Anderson et al. (2020). In §A.6.1 we will show through an example that, without a separate pre-activation bounds treatment, $\mathcal{A}_k$ could be looser than the less computationally expensive $\mathcal{M}_k$ relaxation. We then (§A.6.2) justify our specific pre-activation bounds treatment by extending the original proof by Anderson et al. (2020).

The original formulation by Anderson et al. (2020) is the following:

$$\left.\begin{array}{l} \mathbf{x}_k \geq W_k \mathbf{x}_{k-1} + \mathbf{b}_k \\ \mathbf{x}_k \leq \left(\begin{array}{l} (W_k \odot I_k)\, \mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k \\ -\left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k) \\ +\left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \end{array} \right) \forall I_k \in 2^{W_k} \\ (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \times [\mathbf{0},\ \mathbf{1}] \end{array}\right\} = \mathcal{A}'_k. \tag{A.11}$$

The difference with respect to $\mathcal{A}_k$ as defined in equation equation 2.3 exclusively lies in the treatment of pre-activation bounds. While $\mathcal{A}_k$ explicitly employs generic $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ in the constraint set via $\mathcal{M}_k$, $\mathcal{A}'_k$ implicitly sets $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ to the value dictated by interval propagation bounds (Gowal et al., 2018b) via the constraints in $I_k = 0$ and $I_k = 1$ from the exponential family. In fact, setting $I_k = 0$ and $I_k = 1$, we obtain the following two constraints:

$$\begin{array}{ll} & \mathbf{x}_k \leq \hat{\mathbf{x}}_k - M_k^- \odot (1 - \mathbf{z}_k) \\ & \mathbf{x}_k \leq M_k^+ \odot \mathbf{z}_k \\ \text{where:} & M_k^- := \min_{\mathbf{x}_{k-1} \in [\mathbf{l}_{k-1}, \mathbf{u}_{k-1}]} W_k^T \mathbf{x}_{k-1} + \mathbf{b}_k = W_k \odot \check{L}_{k-1} + \mathbf{b}_k \\ & M_k^+ := \max_{\mathbf{x}_{k-1} \in [\mathbf{l}_{k-1}, \mathbf{u}_{k-1}]} W_k^T \mathbf{x}_{k-1} + \mathbf{b}_k = W_k \odot \check{U}_{k-1} + \mathbf{b}_k \end{array} \tag{A.12}$$

which correspond to the upper bounding ReLU constraints in $\mathcal{M}_k$ if we set $\hat{\mathbf{l}}_k \to M_k^-, \hat{\mathbf{u}}_k \to M_k^+$. While $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ are (potentially) computed solving an optimisation problem over the entire network (problem A.10), the optimisation for M_k^-, M_k^+ involves only the layer before the current. Therefore, the constraints in equation A.12 might be much looser than those in $\mathcal{M}_k$.

In practice, the effect of $\hat{\mathbf{l}}_k[i], \hat{\mathbf{u}}_k[i]$ on the resulting set is so significant that $\mathcal{M}_k$ might yield better bounds than $\mathcal{A}'_k$, even on very small networks. We now provide a simple example.

A.6.1 Motivating Example

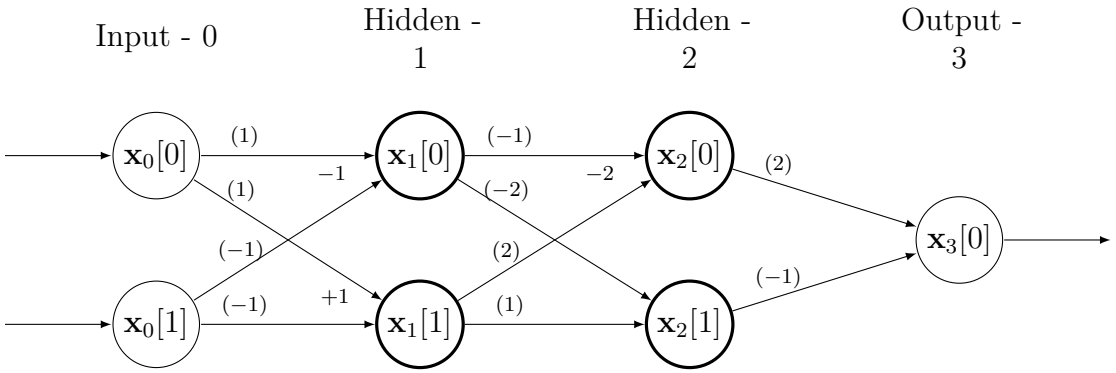


Figure A.2: Example network architecture in which $\mathcal{M}_k \subset \mathcal{A}'_k$, with pre-activation bounds computed with $\mathcal{C}_k = \mathcal{M}_k$. For the bold nodes (the two hidden layers) a ReLU activation follows the linear function. The numbers between parentheses indicate multiplicative weights, the others additive biases (if any).

Figure A.2 illustrates the network architecture. The size of the network is the minimal required to reproduce the phenomenon. $\mathcal{M}_k$ and $\mathcal{A}_k$ coincide for single-neuron layers (Anderson et al., 2020), and $\hat{\mathbf{l}}_k = M_k^-, \hat{\mathbf{u}}_k = M_k^+$ on the first hidden layer (hence, a second layer is needed).

Let us write the example network as a (not yet relaxed, as in problem equation 2.1) optimization problem for the lower bound on the output node $\mathbf{x}_3$.

$$\mathbf{l}_3 = \arg\min_{\mathbf{x},\hat{\mathbf{x}}} \quad \begin{bmatrix} 2 & -1 \end{bmatrix} \mathbf{x}_2 \tag{A.13a}$$

$$\text{s.t.} \quad \mathbf{x}_0 \in [-1, 1]^2 \tag{A.13b}$$

$$\hat{\mathbf{x}}_1 = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \mathbf{x}_0 + \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad \mathbf{x}_1 = \max(0, \hat{\mathbf{x}}_1) \tag{A.13c}$$

$$\hat{\mathbf{x}}_2 = \begin{bmatrix} -1 & 2 \\ -2 & 1 \end{bmatrix} \mathbf{x}_1 + \begin{bmatrix} -2 \\ 0 \end{bmatrix} \qquad \mathbf{x}_2 = \max(0, \hat{\mathbf{x}}_2) \tag{A.13d}$$

$$\mathbf{x}_3 = \begin{bmatrix} 2 & -1 \end{bmatrix} \mathbf{x}_2 \tag{A.13e}$$

Let us compute pre-activation bounds with $\mathcal{C}_k = \mathcal{M}_k$ (see problem equation A.10). For this network, the final output lower bound is tighter if the employed relaxation is $\mathcal{M}_k$ rather than $\mathcal{A}_k$ (hence, in this case, $\mathcal{M}_k \subset \mathcal{A}'_k$). Specifically: $\hat{\mathbf{l}}_{3,\mathcal{A}'_k} = -1.2857$, $\hat{\mathbf{l}}_{3,\mathcal{M}_k} = -1.2273$. In fact:

- In order to compute $\mathbf{l}_1$ and $\mathbf{u}_1$, the post-activation bounds of the first-layer, it suffices to solve a box-constrained linear program for $\hat{\mathbf{l}}_1$ and $\hat{\mathbf{u}}_1$, which at this layer coincide with interval propagation bounds, and to clip them to be non-negative. This yields $\mathbf{l}_1 = \begin{bmatrix}0 & 0\end{bmatrix}^T$, $\mathbf{u}_1 = \begin{bmatrix}1 & 3\end{bmatrix}^T$.

- Computing $M_2^+[1] = \max_{\mathbf{x}_1 \in [\mathbf{l}_1, \mathbf{u}_1]} \begin{bmatrix}-2 & 1\end{bmatrix} \mathbf{x}_1 = 3$ we are assuming that $\mathbf{x}_1[0] = \mathbf{l}_1[0]$ and $\mathbf{x}_1[1] = \mathbf{u}_1[1]$. These two assignments are in practice conflicting, as they imply different values for $\mathbf{x}_0$. Specifically, $\mathbf{x}_1[1] = \mathbf{u}_1[1]$ requires $\mathbf{x}_0 = \begin{bmatrix}\mathbf{u}_0[0] & \mathbf{l}_0[1]\end{bmatrix} = \begin{bmatrix}1 & -1\end{bmatrix}$, but this would also imply $\mathbf{x}_1[0] = \mathbf{u}_1[0]$, yielding $\hat{\mathbf{x}}_2[1] = 1 \neq 3$.

  Therefore, explicitly solving a LP relaxation of the network for the value of $\hat{\mathbf{u}}_2[1]$ will tighten the bound. Using $\mathcal{M}_k$, the LP for this intermediate pre-activation bound is:

$$\hat{\mathbf{u}}_2[1] = \arg\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \quad \begin{bmatrix}-2 & 1\end{bmatrix} \mathbf{x}_1 \tag{A.14a}$$

$$\text{s.t.} \quad \mathbf{x}_0 \in [-1,1]^2,\ \mathbf{z}_1 \in [0,1]^2,\ \mathbf{x}_1 \in \mathbb{R}^2_{\geq 0} \tag{A.14b}$$

$$\hat{\mathbf{x}}_1 = \begin{bmatrix}1 & -1\\ 1 & -1\end{bmatrix} \mathbf{x}_0 + \begin{bmatrix}-1\\ 1\end{bmatrix} \tag{A.14c}$$

$$\mathbf{x}_1 \geq \hat{\mathbf{x}}_1 \tag{A.14d}$$

$$\mathbf{x}_1 \leq \hat{\mathbf{u}}_1 \odot \mathbf{z}_1 = \begin{bmatrix}1\\ 3\end{bmatrix} \odot \mathbf{z}_1 \tag{A.14e}$$

$$\mathbf{x}_1 \leq \hat{\mathbf{x}}_1 - \hat{\mathbf{l}}_1 \odot (1 - \mathbf{z}_1) = \hat{\mathbf{x}}_1 - \begin{bmatrix}-3\\ -1\end{bmatrix} \odot (1 - \mathbf{z}_1) \tag{A.14f}$$

  Yielding $\hat{\mathbf{u}}_2[1] = 2.25 < 3 = M_2^+[1]$. An analogous reasoning holds for $M_2^-[1]$ and $\hat{\mathbf{l}}_2[1]$.

- In $\mathcal{M}_k$, we therefore added the following two constraints:

$$\begin{aligned} \mathbf{x}_2[1] &\leq \hat{\mathbf{x}}_2[1] - \hat{\mathbf{l}}_2[1](1 - \mathbf{z}_2[1]) \\ \mathbf{x}_2[1] &\leq \hat{\mathbf{u}}_2[1]\mathbf{z}_2[1] \end{aligned} \tag{A.15}$$

  that in $\mathcal{A}'_k$ correspond to the weaker:

$$\begin{aligned} \mathbf{x}_2[1] &\leq \hat{\mathbf{x}}_2[1] - M_2^-[1](1 - \mathbf{z}_2[1]) \\ \mathbf{x}_2[1] &\leq M_2^+[1]\mathbf{z}_2[1] \end{aligned} \tag{A.16}$$

  As the last layer weight corresponding to $\mathbf{x}_2[1]$ is negative ($W_3[0, 1] = -1$), these constraints are going to influence the computation of $\hat{\mathbf{l}}_3$.

- In fact, the constraints in equation A.15 are both active when optimizing for $\hat{\mathbf{l}}_{3,\mathcal{M}_k}$, whereas their counterparts for $\hat{\mathbf{l}}_{3,\mathcal{A}'_k}$ in equation A.16 are not. The only active upper constraint at neuron $\mathbf{x}_2[1]$ for the Anderson relaxation is $\mathbf{x}_2[1] \leq \mathbf{x}_1[1]$, corresponding to the constraint from $\mathcal{A}'_2$ with $I_2[1, \cdot] = [0\ 1]$. Evidently, its effect is not sufficient to counter-balance the effect of the tighter constraints equation A.15 for $I_2[1, \cdot] = [1\ 1]$ and $I_2[1, \cdot] = [0\ 0]$, yielding a weaker lower bound for the network output.

A.6.2 Derivation of $\mathcal{A}_k$

Having motivated an explicit pre-activation bounds treatment for the relaxation by Anderson et al. (2020), we now extend the original proof for $\mathcal{A}'_k$ (equation equation A.11) to obtain our formulation $\mathcal{A}_k$ (as defined in equation equation 2.3). For simplicity, we will operate on a single neuron $\mathbf{x}_k[i]$.

A self-contained way to derive $\mathcal{A}'_k$ is by applying Fourier-Motzkin elimination on a standard MIP formulation referred to as the *multiple choice* formulation (Anderson et al., 2019b), which is defined as follows:

$$\left.\begin{array}{l} (\mathbf{x}_{k-1}, \mathbf{x}_k[i]) = (\mathbf{x}^0_{k-1}, \mathbf{x}^0_k[i]) + (\mathbf{x}^1_{k-1}, \mathbf{x}^1_k[i]) \\ \mathbf{x}^0_k[i] = 0 \geq \boldsymbol{w}^T_{i,k}\mathbf{x}^0_{k-1} + \mathbf{b}_k[i](1 - \mathbf{z}_k[i]) \\ \mathbf{x}^1_k[i] = \boldsymbol{w}^T_{i,k}\mathbf{x}^1_{k-1} + \mathbf{b}_k[i]\mathbf{z}_k[i] \geq 0 \\ \mathbf{l}_{k-1}[i](1 - \mathbf{z}_k[i]) \leq \mathbf{x}^0_{k-1} \leq \mathbf{u}_{k-1}[i](1 - \mathbf{z}_k[i]) \\ \mathbf{l}_{k-1}[i]\mathbf{z}_k[i] \leq \mathbf{x}^1_{k-1} \leq \mathbf{u}_{k-1}[i]\mathbf{z}_k[i] \\ \mathbf{z}_k[i] \in [0,\ 1] \end{array}\right\} = \mathcal{S}'_{k,i} \tag{A.17}$$

Where $\boldsymbol{w}_{i,k}$ denotes the i-th row of W_k, and $\mathbf{x}^1_{k-1}$ and $\mathbf{x}^0_{k-1}$ are copies of the previous layer variables. Applying equation A.17 to the entire neural network results in a quadratic number of variables (relative to the number of neurons). The formulation can be obtained from well-known techniques from the MIP literature (Jeroslow, 1987) (it is the union of the two polyhedra for a passing and a blocking ReLU, operating in the space of $\mathbf{x}_{k-1}$). Anderson et al. (2019b) show that $\mathcal{A}'_k = \text{Proj}_{\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{z}_k}(\mathcal{S}'_k)$.

If pre-activation bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ (computed as described in section A.5) are available, we can naturally add them to equation A.17 as follows:

$$\left.\begin{array}{l}(\mathbf{x}_{k-1}, \mathbf{x}_k[i], \mathbf{z}_k[i]) \in \mathcal{S}'_{k,i} \\ \hat{\mathbf{l}}_k[i](1-\mathbf{z}_k[i]) \leq \boldsymbol{w}^T_{i,k}\mathbf{x}^0_{k-1} + \mathbf{b}_k[i](1-\mathbf{z}_k[i]) \leq \hat{\mathbf{u}}_k[i](1-\mathbf{z}_k[i]) \\ \hat{\mathbf{l}}_k[i] \odot \mathbf{z}_k[i] \leq \boldsymbol{w}^T_{i,k}\mathbf{x}^1_{k-1} + \mathbf{b}_k[i]\mathbf{z}_k[i] \leq \hat{\mathbf{u}}_k[i]\mathbf{z}_k[i]\end{array}\right\} = \mathcal{S}_{k,i} \quad \text{(A.18)}$$

We now prove that this formulation yields $\mathcal{A}_k$ when projecting out the copies of the activations.

Proposition 6. *Sets $\mathcal{S}_k$ from equation equation A.18 and $\mathcal{A}_k$ from problem equation 2.3 are equivalent, in the sense that $\mathcal{A}_k = Proj_{\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{z}_k}(\mathcal{S}_k)$.*

Proof. In order to prove the equivalence, we will rely on Fourier-Motzkin elimination as in the original Anderson relaxation proof (Anderson et al., 2019b). Going along the lines of the original proof, we start from equation A.17 and eliminate $\mathbf{x}^1_{k-1}$, $\mathbf{x}^0_k[i]$ and $\mathbf{x}^1_k[i]$ exploiting the equalities. We then re-write all the inequalities as upper or lower bounds on $\mathbf{x}^0_{k-1}[0]$ in order to eliminate this variable. As Anderson et al. (2019b), we assume $\boldsymbol{w}_{i,k}[0] > 0$. The proof generalizes by using $\check{L}$ and $\check{U}$ for $\boldsymbol{w}_{i,k}[0] < 0$, whereas if the coefficient is 0 the variable is easily eliminated. We get the following system:

$$\mathbf{x}^0_{k-1}[0] = \frac{1}{\boldsymbol{w}_{i,k}[0]}\left(\boldsymbol{w}^T_{i,k}\mathbf{x}_{k-1} - \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] - \mathbf{x}_k[i]\right) \quad \text{(A.19a)}$$

$$\mathbf{x}^0_{k-1}[0] \leq -\frac{1}{\boldsymbol{w}_{i,k}[0]}\left(\sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i](1-\mathbf{z}_k[i])\right) \quad \text{(A.19b)}$$

$$\mathbf{x}^0_{k-1}[0] \leq \frac{1}{\boldsymbol{w}_{i,k}[0]} \left(\boldsymbol{w}^T_{i,k} \mathbf{x}_{k-1} - \sum_{j>1} \boldsymbol{w}_{i,k}[j] \mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i] \mathbf{z}_k[i] \right) \tag{A.19c}$$

$$\mathbf{l}_{k-1}[0](1 - \mathbf{z}_k[i]) \leq \mathbf{x}^0_{k-1}[0] \leq \mathbf{u}_{k-1}[0](1 - \mathbf{z}_k[i]) \tag{A.19d}$$

$$\mathbf{x}^0_{k-1}[0] \leq \mathbf{x}_{k-1}[0] - \mathbf{l}_{k-1}[0] \mathbf{z}_k[i] \tag{A.19e}$$

$$\mathbf{x}^0_{k-1}[0] \geq \mathbf{x}_{k-1}[0] - \mathbf{u}_{k-1}[0] \mathbf{z}_k[i] \tag{A.19f}$$

$$\mathbf{x}^0_{k-1}[0] \leq \frac{1}{\boldsymbol{w}_{i,k}[0]} \left(\boldsymbol{w}^T_{i,k} \mathbf{x}_{k-1} - \sum_{j>1} \boldsymbol{w}_{i,k}[j] \mathbf{x}^0_{k-1}[j] + (\boldsymbol{b}_k[i] - \hat{\mathbf{l}}_k[i]) \mathbf{z}_k[i] \right) \tag{A.19g}$$

$$\mathbf{x}^0_{k-1}[0] \geq \frac{1}{\boldsymbol{w}_{i,k}[0]} \left(\boldsymbol{w}^T_{i,k} \mathbf{x}_{k-1} - \sum_{j>1} \boldsymbol{w}_{i,k}[j] \mathbf{x}^0_{k-1}[j] + (\boldsymbol{b}_k[i] - \hat{\mathbf{u}}_k[i]) \mathbf{z}_k[i] \right) \tag{A.19h}$$

$$\mathbf{x}^0_{k-1}[0] \geq \frac{1}{\boldsymbol{w}_{i,k}[0]} \left((\hat{\mathbf{l}}_k[i] - \boldsymbol{b}_k[i])(1 - \mathbf{z}_k[i]) - \sum_{j>1} \boldsymbol{w}_{i,k}[j] \mathbf{x}^0_{k-1}[j] \right) \tag{A.19i}$$

$$\mathbf{x}^0_{k-1}[0] \leq \frac{1}{\boldsymbol{w}_{i,k}[0]} \left((\hat{\mathbf{u}}_k[i] - \boldsymbol{b}_k[i])(1 - \mathbf{z}_k[i]) - \sum_{j>1} \boldsymbol{w}_{i,k}[j] \mathbf{x}^0_{k-1}[j] \right) \tag{A.19j}$$

where only inequalities equation A.19g to equation A.19j are not present in the original proof. We therefore focus on the part of the Fourier-Motzkin elimination that deals with them, and invite the reader to refer to Anderson et al. (2019b) for the others. The combination of these new inequalities yields trivial constraints. For instance:

$$\textit{equation A.19}i + \textit{equation A.19}g \implies \hat{\mathbf{l}}_k[i] \leq \boldsymbol{w}^T_{i,k} \mathbf{x}_{k-1} + \boldsymbol{b}_k[i] = \hat{\mathbf{x}}_k[i] \tag{A.20}$$

which holds by the definition of pre-activation bounds.

Let us recall that $\mathbf{x}_k[i] \geq 0$ and $\mathbf{x}_k[i] \geq \hat{\mathbf{x}}_k[i]$, the latter constraint resulting from *equation A.19a + equation A.19b*. Then, it can be easily verified that the only combinations of interest (i.e., those that do not result in constraints that are obvious by definition or are implied by other constraints) are those containing the equality equation A.19a. In particular, combining inequalities equation A.19g to equation A.19j with inequalities equation A.19d to equation A.19f generates constraints that are (after algebraic manipulations) superfluous with respect to those in equation A.21. We are now ready to show the system resulting from the elimination:

$$\mathbf{x}_k[i] \geq 0 \tag{A.21a}$$

$$\mathbf{x}_k[i] \geq \hat{\mathbf{x}}_k[i] \tag{A.21b}$$

$$\mathbf{x}_k[i] \leq \boldsymbol{w}_{i,k}[0]\mathbf{x}_{k-1}[0] - \boldsymbol{w}_{i,k}[0]\mathbf{l}_{k-1}[0](1 - \mathbf{z}_k[i]) + \sum_{j>1} \boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{A.21c}$$

$$\mathbf{x}_k[i] \leq \boldsymbol{w}_{i,k}[0]\mathbf{u}_{k-1}[0]\mathbf{z}_k[i] + \sum_{j>1} \boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{A.21d}$$

$$\mathbf{x}_k[i] \geq \boldsymbol{w}_{i,k}[0]\mathbf{x}_{k-1}[0] - \boldsymbol{w}_{i,k}[0]\mathbf{u}_{k-1}[0](1 - \mathbf{z}_k[i]) + \sum_{j>1} \boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{A.21e}$$

$$\mathbf{x}_k[i] \geq \boldsymbol{w}_{i,k}[0]\mathbf{l}_{k-1}[0]\mathbf{z}_k[i] + \sum_{j>1} \boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{A.21f}$$

$$\mathbf{l}_{k-1}[0] \leq \mathbf{x}_k[i] \leq \mathbf{u}_{k-1}[0] \tag{A.21g}$$

$$\mathbf{x}_k[i] \geq \hat{\mathbf{l}}_k[i]\mathbf{z}_k[i] \tag{A.21h}$$

$$\mathbf{x}_k[i] \leq \hat{\mathbf{u}}_k[i]\mathbf{z}_k[i] \tag{A.21i}$$

$$\mathbf{x}_k[i] \leq \hat{\mathbf{x}}_k[i] - \hat{\mathbf{l}}_k[i](1 - \mathbf{z}_k[i]) \tag{A.21j}$$

$$\mathbf{x}_k[i] \geq \hat{\mathbf{x}}_k[i] - \hat{\mathbf{u}}_k[i](1 - \mathbf{z}_k[i]) \tag{A.21k}$$

Constraints from equation A.21a to equation A.21g are those resulting from the original derivation of $\mathcal{A}'_k$ (see (Anderson et al., 2019b)). The others result from the inclusion of pre-activation bounds in equation A.18. Of these, equation A.21h is implied by equation A.21a if $\hat{\mathbf{l}}_k[i] \leq 0$ and by the definition of pre-activation bounds (together with equation A.21b) if $\hat{\mathbf{l}}_k[i] > 0$. Analogously, equation A.21k is implied by equation A.21b if $\hat{\mathbf{u}}_k[i] \geq 0$ and by equation A.21a otherwise.

By noting that no auxiliary variable is left in equation A.21i and in equation A.21j, we can conclude that these will not be affected by the remaining part of the elimination procedure. Therefore, the rest of the proof (the elimination of $\mathbf{x}^0_{k-1}[1]$, $\mathbf{x}^0_{k-1}[2]$, ...) proceeds as in (Anderson et al., 2019b), leading to $\mathcal{A}_{k,i}$. Repeating the proof for each neuron i at layer k, we get $\mathcal{A}_k = \text{Proj}_{\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{z}_k}(\mathcal{S}_k)$.

□

A.7 Masked Forward and Backward Passes

Crucial to the practical efficiency of our solvers is to represent the various operations as standard forward/backward passes over a neural network. This way, we can leverage the engineering efforts behind popular deep learning frameworks such as PyTorch (Paszke et al., 2017). While this can be trivially done for the Big-M solver (appendix A.2), the Active Set method (§2.3.1) requires a specialised operator that we call “masked" forward/backward pass. Here, we provide the details to our implementation.

The masked forward and backward passes respectively take the following forms: $(W_k \odot I_k)\,\mathbf{x}_k$, $(W_k \odot I_k)^T\,\mathbf{x}_{k+1}$ and they are needed when dealing with the exponential family of constraints from the relaxation by Anderson et al. (2020). A crucial property of the operator is that the I_k mask may take on a different value for each input/output combination. While this is straightforward to implement for fully connected layers, we need to be more careful when handling convolutional operators, which rely on re-applying the same weights (kernel) to many different parts of the image. A naive solution is to convert convolutions into equivalent linear operators, but this has a high cost in terms of performance, as it involves much redundancy.

A convolutional operator can be represented via a matrix-matrix multiplication if the input is *unfolded* and the filter is appropriately reshaped. The multiplication output can then be reshaped to the correct convolutional output shape. Given a filter $w \in \mathbb{R}^{c \times k_1 \times k_2}$, an input $\mathbf{x} \in \mathbb{R}^{i_1 \times i_2 \times i_3}$ and the convolutional output $\texttt{conv}_w(\mathbf{x}) = \mathbf{y} \in \mathbb{R}^{c \times o_2 \times o_3}$, we need the following definitions:

$$\begin{aligned} [\cdot]_{\mathcal{I},\mathcal{O}} &: \mathcal{I} \to \mathcal{O} \\ \{\cdot\}_j &: \mathbb{R}^{d_1 \times \cdots \times d_n} \to \mathbb{R}^{d_1 \times \cdots \times d_{j-1} \times d_{j+1} \times \cdots \times d_n} \\ \texttt{unfold}_w(\cdot) &: \mathbb{R}^{i_1 \times i_2 \times i_3} \to \mathbb{R}^{k_1 k_2 \times o_2 o_3} \\ \texttt{fold}_w(\cdot) &: \mathbb{R}^{k_1 k_2 \times o_2 o_3} \to \mathbb{R}^{i_1 \times i_2 \times i_3} \end{aligned} \tag{A.22}$$

where the brackets simply reshape the vector from shape $\mathcal{I}$ to $\mathcal{O}$, while the braces sum over the j-th dimension. `unfold` decomposes the input image into the

(possibly overlapping) $o_2 o_3$ blocks the sliding kernel operates on, taking padding and striding into account. `fold` brings the output of `unfold` to the original input space.

Let us define the following reshaped versions of the filter and the convolutional output:

$$W_R = [w]_{\mathbb{R}^{c \times k_1 \times k_2}, \mathbb{R}^{c \times k_1 k_2}}$$

$$\mathbf{y}_R = [\mathbf{y}]_{\mathbb{R}^{c \times o_2 \times o_3}, \mathbb{R}^{c \times o_2 o_3}}$$

The standard forward/backward convolution (neglecting the convolutional bias, which can be added at the end of the forward pass) can then be written as:

$$\begin{aligned} \texttt{conv}_w(\mathbf{x}) &= [W_R \ \texttt{unfold}_w(\mathbf{x})]_{\mathbb{R}^{c \times o_2 o_3}, \mathbb{R}^{c \times o_2 \times o_3}} \\ \texttt{back\_conv}_w(\mathbf{y}) &= \texttt{fold}_w(W_R^T \ \mathbf{y}_R). \end{aligned} \tag{A.23}$$

We need to mask the convolution with a different mask for each input-output pair. This means employing a mask $I \in \mathbb{R}^{c \times k_1 k_2 \times o_2 o_3}$. Therefore, assuming vectors are broadcast to the correct output shape[1] , we can write the masked forward and backward passes as:

$$\begin{aligned} \texttt{conv}_{w,I}(\mathbf{x}) &= [\{(W_R \ \odot I \odot \texttt{unfold}_w(\mathbf{x})\}_2]_{\mathbb{R}^{c \times o_2 o_3}, \mathbb{R}^{c \times o_2 \times o_3}} \\ \texttt{back\_conv}_{w,I}(\mathbf{y}) &= \texttt{fold}_w(\{W_R \ \odot I \odot \mathbf{y}_R\}_1). \end{aligned} \tag{A.24}$$

Owing to the avoided redundancy with respect to the equivalent linear operation (e.g., copying of the kernel matrix, zero-padding in the linear weight matrix), this implementation of the masked forward/backward pass reduces both the memory footprint and the number of floating point operations (FLOPs) associated to the passes computations by a factor $(i_1 i_2 i_3)/(k_1 k_2)$. In practice, this ratio might be significant: on the incomplete verification networks §2.5.1) it ranges from 16 to 64.

A.8 Stratified Bounding for Branch and Bound

As seen in the complete verification results in Figure 2.3 (§ 2.5.2), the use of a tighter bounding algorithm results in the verification of a larger number of properties. In

[1] if we want to perform an element-wise product $\mathbf{a} \odot \mathbf{b}$ between $\mathbf{a} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ and $\mathbf{b} \in \mathbb{R}^{d_1 \times d_3}$, the operation is implicitly performed as $\mathbf{a} \odot \mathbf{b}'$, where $\mathbf{b}' \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ is an extended version of $\mathbf{b}$ obtained by copying along the missing dimension.

general, tighter bounds come at a larger computational cost, which might negatively affect performance on “easy" verification properties, where a small number of domain splits with loose bounds suffices to verify the property (hence, the tightness is not needed). As a general complete verification system needs to be efficient a priori, on both easy and hard properties, we employ a *stratified* bounding system for use within Branch and Bound-like complete verification algorithms.

We design a set of heuristics to determine whether the children (i.e., the subproblems arising after splitting on it) of a given subproblem will require tight bounds or not. The graph describing parent-child relations between sub-problems is referred to as the BaB search tree. Given a problem p, we individually mark it as difficult (with all its future offspring) if it meets *all* the conditions in the following set:

- The lower bound on the minimum over the subproblem, p_{LB}, should be relatively “close" to the decision threshold (0, in the standard form by Bunel et al. (2018)). That is, $p_{LB} \leq c_{LB}$. We argue that tighter bounds are worth computing only if they are likely to result in a crossing of the decision threshold, thus cutting away a part of the BaB search tree.

- The depth p_{depth} in the BaB search tree should not be below a certain threshold c_{depth}. This is a way to ensure a certain number of splits is performed before adopting tighter bounds.

- The running average of the lower bound improvement from parent to child p_{impr} should fall below a threshold c_{impr}. This reflects the idea that if splitting seems to be effective on a given section of the BaB tree, it is perhaps wiser to invest computational budget in more splits (with cheaper bounding computations, more splits can be performed in a given time) than tighter bounds. Empirically, this is the criterion with the largest effect on performance.

Additionally, we do not employ tighter bounds unless the verification problem itself (in addition to the individual subproblems) is marked as “hard". We do so when the size of the un-pruned domains reaches a certain threshold c_{hard}.

The set of criteria above needs to address the following decision trade-off: one should postpone the use of tighter bounds until sure the difficulty of the verification task requires it. At the same time, if the switch to tighter bounds is excessively delayed, one might end up computing tight bounds on a large number of subproblems (as sections of the BaB tree were not pruned in time), hurting performance for harder tasks. In practice, we set the criteria as follows for all our experiments: $c_{depth} = 0$, $c_{LB} = 0.5$, $c_{impr} = 10^{-1}$, $c_{hard} = 200$. As seen in Figure 2.3, this resulted in a reasonable trade-off between losing efficiency on the harder properties (wide model) and gaining it on the easier ones (base and deep models).

A.9 Experimental Appendix

We conclude the appendix by presenting supplementary experiments and additional details with respect to the presentation in the main paper.

A.9.1 Experimental Setting, Hyper-parameters

All the experiments and bounding computations (including intermediate bounds) were run on a single Nvidia Titan Xp GPU, except Gurobi-based methods and "Active Set CPU". These were run on i7-6850K CPUs, utilising 4 cores for the incomplete verification experiments, and 6 cores for the more demanding complete verification experiments. The experiments were run under Ubuntu 16.04.2 LTS. Complete verification results for ERAN, the method by Singh et al. (2020), are taken from the recent VNN-COMP competition (VNN-COMP, 2020). These were executed by Singh et al. (2020) on a 2.6 GHz Intel Xeon CPU E5-2690 with 512 GB of main memory, utilising 14 cores.

Gurobi-based methods make use of LP incrementalism (warm-starting) when possible. In the experiments of §2.5.1, where each image involves the computation of 9 different output upper bounds, we warm-start each LP from the LP of the previous neuron. For "Gurobi 1 cut", which involves two LPs per neuron, we first solve all Big-M LPs, then proceed with the LPs containing a single cut.

Hyper-parameter tuning for incomplete verification was done on a small subset of the CIFAR-10 test set, on the SGD-trained network from Figures 2.1, 2.2. BDD+ is run with the hyper-parameters found by Bunel et al. (2020a) on the same datasets, for both incomplete and complete verification. For all supergradient-based methods (Big-M, Active Set), we employed the Adam update rule (Kingma and Ba, 2015), which showed stronger empirical convergence. For Big-M, replicating the findings by Bunel et al. (2020a) on their supergradient method, we linearly decrease the step size from 10^{-2} to 10^{-4}. Active Set is initialized with 500 Big-M iterations, after which the step size is reset and linearly scaled from 10^{-3} to 10^{-6}. We found the addition of variables to the active set to be effective before convergence: we add variables every 450 iterations, without re-scaling the step size again. Every addition consists of 2 new variables (see pseudo-code in appendix A.4), which was found to be a good compromise between fast bound improvement and computational cost. On complete verification, we re-employed the same hyper-parameters for both Big-M and Active Set, except the number of iterations. For Big-M, this was tuned to employ roughly the same time per bounding computation as BDD+.

As the complete verification problem is formulated as a minimisation (as in problem equation 2.1), in Branch and Bound we need a lower and an upper bound to the minimum of each sub-problem. The lower bound is computed by running the bounding algorithm, while the upper bound on the minimum is obtained by evaluating the network at the last primal output by the bounding algorithm in the lower bound computation (running the bounding algorithm to get an upper bound would result in a much looser bound, as it would imply having an upper bound on a version of problem equation 2.1 with maximisation instead of minimisation).

A.9.2 Dataset Details

We now provide further details on the employed datasets. For incomplete verification, the architecture was introduced by Wong and Kolter (2018) and re-employed by Bunel et al. (2020a). It corresponds to the “Wide" complete verification architecture, found in Table A.1 . Due to the additional computational cost of bounds obtained

Network Name	No. of Properties	Network Architecture
BASE Model	100	Conv2d(3,8,4, stride=2, padding=1) Conv2d(8,16,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 3172)
WIDE	100	Conv2d(3,16,4, stride=2, padding=1) Conv2d(16,32,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 6244)
DEEP	100	Conv2d(3,8,4, stride=2, padding=1) Conv2d(8,8,3, stride=1, padding=1) Conv2d(8,8,3, stride=1, padding=1) Conv2d(8,8,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 6756)

Table A.1: For each complete verification experiment, the network architecture used and the number of verification properties tested, a subset of the dataset by Lu and Kumar (2020). Each layer but the last is followed by a ReLU activation function.

via the tighter relaxation equation 2.3, we restricted the experiments to the first 3450 CIFAR-10 test set images for the experiments on the SGD-trained network (Figures 2.1, 2.2), and to the first 4513 images for the network trained via the method by Madry et al. (2018) (Figures A.3, A.4).

For complete verification, we employed a subset of the adversarial robustness dataset presented by Lu and Kumar (2020) and used by Bunel et al. (2020a), where the set of properties per network has been restricted to 100. The dataset provides, for a subset of the CIFAR-10 test set, a verification radius ϵ_{ver} defining the small region over which to look for adversarial examples (input points for which the output of the network is not the correct class) and a (randomly sampled) non-correct class to verify against. The verification problem is formulated as the search for an adversarial example, carried out by minimizing the difference between the ground truth logit and the target logit. If the minimum is positive, we have not succeeded in finding a counter-example, and the network is robust. The ϵ_{ver} radius was tuned to meet a certain “problem difficulty" via binary search, employing

a Gurobi-based bounding algorithm (Lu and Kumar, 2020). The networks are robust on all the properties we employed. Three different network architectures of different sizes are used. A "Base" network with 3172 ReLU activations, and two networks with roughly twice as many activations: a "Deep" network, and a "Wide" network. Details can be found in Table A.1.

A.9.3 Adversarially-Trained Incomplete Verification

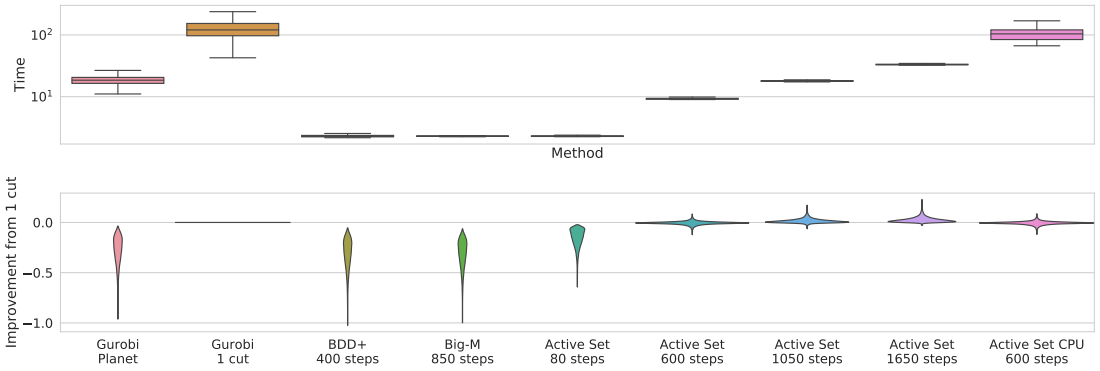
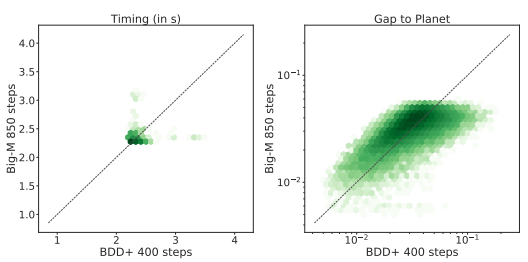
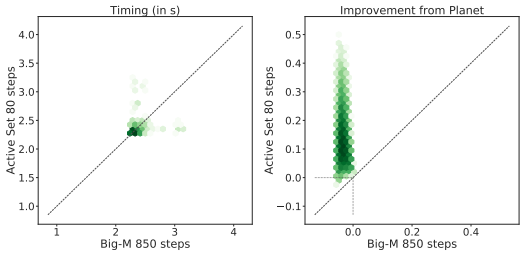


Figure A.3: Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the network adversarially trained with the method by Madry et al. (2018), from Bunel et al. (2020a). The width at a given value represents the proportion of problems for which this is the result.



(a) Comparison of runtime (left) and gap to Gurobi Planet bounds. For the latter, lower is better.



(b) Comparison of runtime (left) and improvement from the Gurobi Planet bounds. For the latter, higher is better.

Figure A.4: Pointwise comparison for a subset of the methods on the data presented in Figure A.3. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

In addition to the SGD-trained network in §2.5.1, we now present results relative to the same architecture, trained with the adversarial training method by Madry et al. (2018) for robustness to perturbations of $\epsilon_{train} = 8/255$. Each adversarial

sample for the training was obtained using 50 steps of projected gradient descent. For this network, we measure the robustness margin to perturbations with $\epsilon_{ver} = 12/255$.

Figures A.3, A.4 confirm most of the observations carried out for the SGD-trained network in §2.5.1, with fewer variability around the bounds returned by Gurobi cut. Big-M is competitive with BDD+, and switching to Active Set after 500 iterations results in much better bounds in the same time. Increasing the computational budget for Active Set still results in better bounds than Gurobi cut in a fraction of its running time, even though the performance gap is on average smaller than on the SGD-trained network.

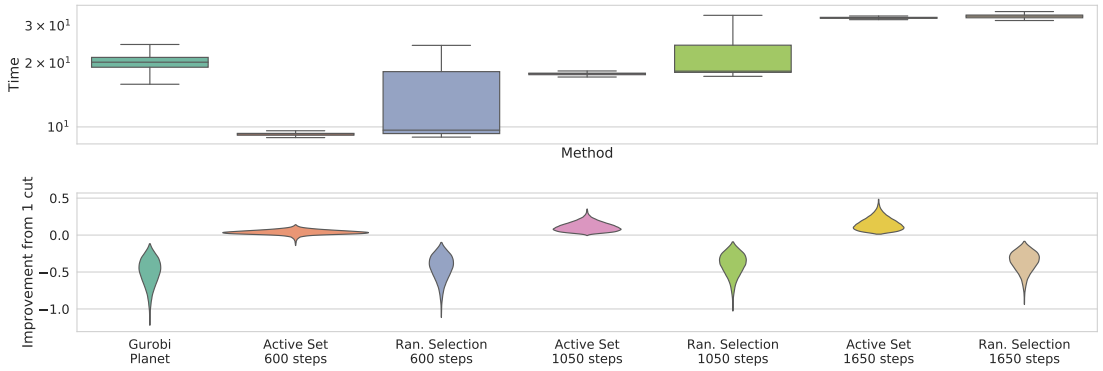


Figure A.5: Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the SGD-trained network from Bunel et al. (2020a). The width at a given value represents the proportion of problems for which this is the result. Sensitivity of Active Set to selection criterion (see §2.3.2).

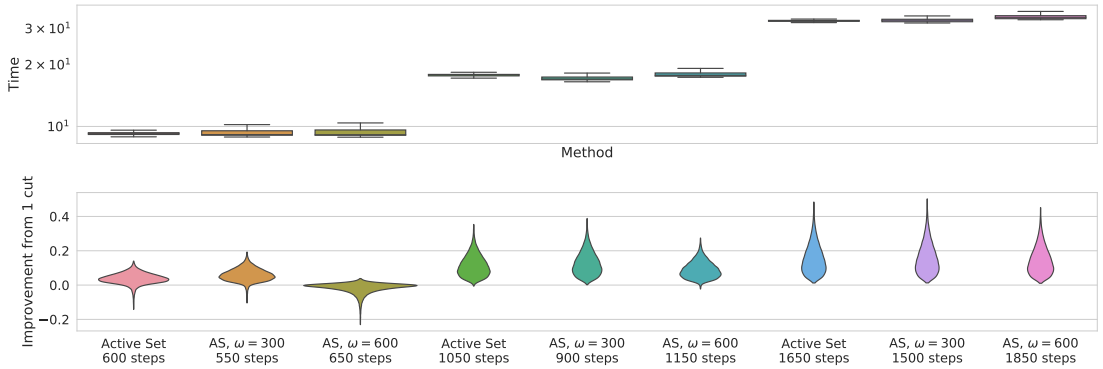


Figure A.6: Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the SGD-trained network from Bunel et al. (2020a). The width at a given value represents the proportion of problems for which this is the result. Sensitivity of Active Set to variable addition frequency ω, with the selection criterion presented in §2.3.2.

A.9.4 Sensitivity to selection criterion and frequency

In section 2.3.2, we describe how to iteratively modify $\mathcal{B}$, the active set of dual variables on which our Active Set solver operates. In short, Active Set adds the variables corresponding to the output of oracle equation 2.4 invoked at the primal minimiser of $\mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$, at a fixed frequency ω. We now investigate the empirical sensitivity of Active Set to both the selection criterion and the frequency of addition.

We test against **Ran. Selection**, a version of Active Set for which the variables to add are selected at random by uniformly sampling from the binary I_k masks. As expected, Figure A.5 shows that a good selection criterion is key to the efficiency of Active Set. In fact, random variable selection only marginally improves upon the Planet relaxation bounds, whereas the improvement becomes significant with our selection criterion from §2.3.2.

In addition, we investigate the sensitivity of Active Set (AS) to variable addition frequency ω. In order to do so, we cap the maximum number of cuts at 7 for all runs, and vary ω while keeping the time budget fixed (we test on three different time budgets). Figure A.6 compares the results for $\omega = 450$ (Active Set), which were presented in §2.5.1, with the bounds obtained by setting $\omega = 300$ and $\omega = 600$ (respectively **AS $\boldsymbol{\omega} = 300$** and **AS $\boldsymbol{\omega} = 600$**). Our solver proves to be relatively robust to ω across all the three budgets, with the difference in obtained bounds decreasing with the number of iterations. Moreover, early cut addition tends to yield better bounds in the same time, suggesting that our selection criterion is effective before subproblem convergence.

A.9.5 MNIST Incomplete Verification

We conclude the experimental appendix by presenting incomplete verification results (the experimental setup mirrors the one employed in section 2.5.1 and appendix A.9.3) on the MNIST dataset (LeCun et al., 1998).

We report results on the “wide" MNIST network from Lu and Kumar (2020), whose architecture is identical to the “wide" network in Table A.1 except for the

first layer, which has only one input channel to reflect the MNIST specification (the total number of ReLU activation units is 4804). As those employed for the complete verification experiments (§2.5.2), and differently from the incomplete verification experiments in section 2.5.1 and appendix A.9.3, the network was adversarially trained with the method by Wong and Kolter (2018). We compute the robustness margin to $\epsilon_{ver} = 0.15$ on the first 2960 images of the MNIST test set. All hyper-parameters are kept to the values employed for the CIFAR-10 networks, except the Big-M step size, which was linearly decreased from 10^{-1} to 10^{-3}, and the weight of the proximal terms for BDD+, which was linearly increased from 1 to 50.

As seen on the CIFAR-10 networks, Figures A.7, A.8 show that Active Set yields comparable or better bounds than Gurobi 1 cut in less average runtime. However, more iterations are required to reach the same relative bound improvement over

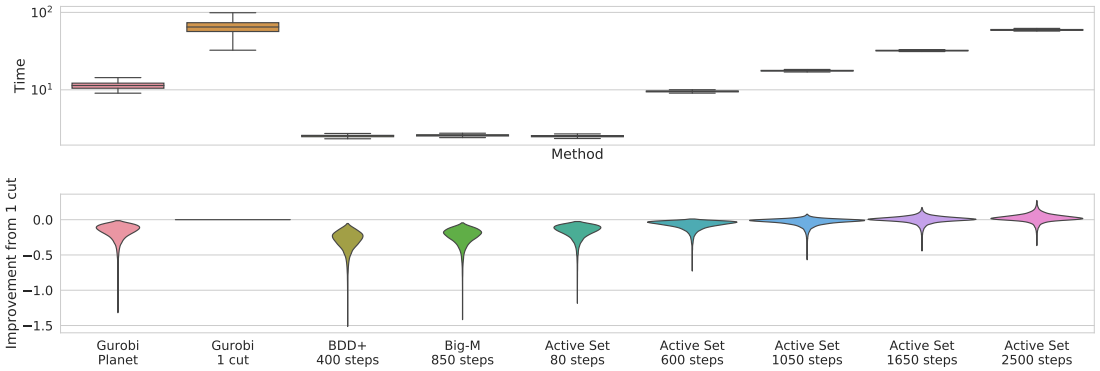


Figure A.7: Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. MNIST results for a network adversarially trained with the method by Wong and Kolter (2018), from Lu and Kumar (2020). The width at a given value represents the proportion of problems for which this is the result.

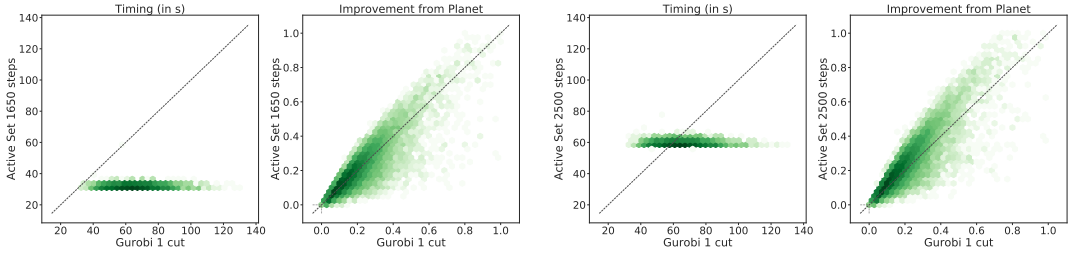


Figure A.8: Pointwise comparison for a subset of the methods on the data presented in Figure A.7. Comparison of runtime (left) and improvement from the Gurobi Planet bounds. For the latter, higher is better. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

Gurobi 1 cut (2500 as opposed to 600 in Figures 2.1, A.3). Furthermore, the smaller average gap between the bounds of Gurobi Planet and Gurobi 1 cut (especially with respect to Figure 2.1) suggests that the relaxation by Anderson et al. (2020) is less effective on this MNIST benchmark.

B

Overcoming the Convex Barrier for Simplex Inputs

Contents

B.1 Proofs

In this section, we present proofs for the theoretical results presented in the main paper Section 3.1.

Theorem 7. *The convex hull of set $\mathcal{S}$, $\mathcal{CH}(\mathcal{S})$, is the set of all $(y, \boldsymbol{x})$ satisfying the following convex constraints:*

$$y \geq \sigma\left(\boldsymbol{w}^T \boldsymbol{x} + b\right), \quad \boldsymbol{x} \in \Delta, \tag{B.1a}$$

$$y \leq \sum_i x_i \left(\sigma\left(\boldsymbol{w}^T \boldsymbol{e}^i + b\right) - \sigma\left(b\right)\right) + \sigma\left(b\right), \tag{B.1b}$$

where $\boldsymbol{e}^i \in \mathbb{R}^n, e^i_i = 1, e^i_j = 0 \, \forall j \neq i$, denotes the i-th coordinate vector in $\mathbb{R}^n$.

Proof. In this proof, for brevity, we write $\mathcal{CH}$ to denote $\mathcal{CH}(\mathcal{S})$. To begin with, we note that $\mathcal{CH}$ is clearly a convex set, since σ is a convex function.

The convex hull of the set

$$\mathcal{S} = \{(\boldsymbol{x}, y) : y = \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right), \boldsymbol{x} \in \Delta\}$$

can be characterized as the intersection of all the halfspaces that contain it (Boyd and Vandenberghe, 2004). Equivalently, if we demonstrate that any linear function obtains the same maximum value over $\mathcal{S}$ as over $\mathcal{CH}$, the proof is complete.

Let $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}$ denote coefficients of a linear function on $\mathcal{S}$.

If $\beta > 0$, the maximum is given by

$$\max_{(\boldsymbol{x},y)\in\mathcal{S}} \alpha^T\boldsymbol{x} + \beta y = \max_{\boldsymbol{x}\in\Delta} \alpha^T\boldsymbol{x} + \beta\sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right) = \max_{e\in\mathcal{V}} \alpha^T e + \beta\sigma\left(\boldsymbol{w}^T e + b\right)$$

where $\mathcal{V} = \{0, \boldsymbol{e}^1, \boldsymbol{e}^2, \ldots, \boldsymbol{e}^n\}$ denotes the set of vertices of the simplex Δ, and the second inequality follows from the fact that the objective is a convex function of $\boldsymbol{x}$. Since the constraint is invariant to scaling β, we can assume $\beta = 1$. Thus, we have that for any $(\boldsymbol{x}, y) \in \mathcal{S}$,

$$\max_{(\boldsymbol{x},y)\in\mathcal{S}} \alpha^T\boldsymbol{x} + y = \sigma(b) + \max\left(\max_i \alpha_i + \sigma\left(\boldsymbol{w}^T\boldsymbol{e}^i + b\right) - \sigma(b), 0\right)$$

If we optimize the same linear function over $\mathcal{CH}$, we obtain

$$\begin{aligned}
&\max_{(\boldsymbol{x},y)\in\mathcal{CH}} \alpha^T\boldsymbol{x} + y = \max_{\boldsymbol{x}\in\Delta} \alpha^T\boldsymbol{x} + \sum_i x_i\left(\sigma\left(\boldsymbol{w}^T\boldsymbol{e}^i + b\right) - \sigma(b)\right) + \sigma(b) \\
&= \max_{\boldsymbol{x}\in\Delta} \sum_i x_i\left(\alpha_i + \sigma\left(\boldsymbol{w}^T\boldsymbol{e}^i + b\right) - \sigma(b)\right) + \sigma(b) \\
&= \sigma(b) + \max\left(\max_i \left(\alpha_i + \sigma\left(\boldsymbol{w}^T\boldsymbol{e}^i + b\right) - \sigma(b)\right), 0\right)
\end{aligned}$$

Thus the linear functions obtain the same optimum over both sets when $\beta > 0$.

If $\beta < 0$, the maximum value of the linear function over $\mathcal{S}$

$$\max_{(\boldsymbol{x},y)\in\mathcal{S}} \alpha^T\boldsymbol{x} + \beta y = \max_{\boldsymbol{x}\in\Delta} \alpha^T\boldsymbol{x} + \beta\sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right) = \max_{(\boldsymbol{x},y)\in\mathcal{CH}} \alpha^T\boldsymbol{x} + \beta y$$

where the last equality follows from the fact that $\beta < 0$ and $\mathcal{CH}$ only contains one lower bound on y for any fixed $\boldsymbol{x}$, ie, $y \geq \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$ (equation B.1b) and the objective is optimized by setting y to its lower bound.

The only remaining case is when $\beta = 0$ and in this case, the optimization over both $\mathcal{S}, \mathcal{CH}$ reduce to

$$\max_{\boldsymbol{x} \in \Delta} \alpha^T \boldsymbol{x} = \max_{e \in \mathcal{V}} \alpha^T e$$

and are indeed identical.

Thus, we have demonstrated that the maximum value of any linear function over $\mathcal{S}$ and over $\mathcal{CH}$ are identical, and hence $\mathcal{CH}$ is the convex hull of $\mathcal{S}$. □

Proposition 8. *For any input dimension m, $\mathcal{CH}_\Delta$ is provably tighter than $\mathcal{P}_\Delta$. Specifically, we can characterize the gap between $\mathcal{CH}_\Delta$ and $\mathcal{P}_\Delta$ by the gap in the optimal value of the problem*

$$\max y - y' \text{ subject to } (y, \boldsymbol{x}) \in \mathcal{P}_\Delta, (y', \boldsymbol{x}) \in \mathcal{CH}_\Delta. \tag{B.2}$$

The gap in the optimal value is shown to be proportional to the variance in the weight vector $\boldsymbol{w}$.

Proof. Let $\boldsymbol{x} \in \Delta_n$, $h(\boldsymbol{x}) = \text{ReLU}(\boldsymbol{w}^T \boldsymbol{x} + b)$. Let $w_{\min} = \min_i w_i$. Note here that $w_{\min}$ includes comparison to 0 to take care of the origin point of the simplex. Let $i_{\min}$ or $i_{\max}$ denote the indices corresponding to $w_{\min}$ and $w_{\max}$ respectively. We use the pre-activation bounds computed using simplex, as per the definition of $\mathcal{P}_\Delta$, which is $l = w_{\min} + b$ and $u = w_{\max} + b$. We assume that $l \leq 0 \leq u$. The upper bounding cut in planet relaxation is

$$y^{\mathcal{P}}(\boldsymbol{x}) = \frac{u}{u - l}(\boldsymbol{w}^T \boldsymbol{x} + b - l).$$

We can write this as

$$y^{\mathcal{P}}(\boldsymbol{x}) = \frac{w_{\max} + b}{w_{\max} - w_{\min}}(\boldsymbol{w}^T \boldsymbol{x} - w_{\min}).$$

To compare the tightness, we compare the difference $y^{\mathcal{P}} - h(\boldsymbol{x})$ for the simplex vertices. We begin by noting that the value $y^{\mathcal{P}}$ at a vertex point $\boldsymbol{e}^i$ will be

$$y^{\mathcal{P}}(\boldsymbol{e}^i) = \frac{w_{\max} + b}{w_{\max} - w_{\min}}(w_i - w_{\min}). \tag{B.3}$$

By the definition and construction of the Planet relaxation, $y^{\mathcal{P}} = h(.)$ when $i = i_{\min}$ or $i = i_{\max}$. For $i \in \{0, .., n\}$ we have

$$y^{\mathcal{P}} - h\left(\boldsymbol{e}^i\right) = \begin{cases} -\, l\dfrac{w_{\max} - w_i}{w_{\max} - w_{\min}} & \text{if } w_i + b \geq 0, \quad \text{(B.4a)} \\ u\dfrac{w_i - w_{\min}}{w_{\max} - w_{\min}} & \text{if } w_i + b < 0. \quad \text{(B.4b)} \end{cases}$$

Since the difference $y^{\mathcal{P}} - h\left(\boldsymbol{e}^i\right)$ will be non-zero, we can conclude that the planet relaxation does not represent the convex hull. Since our relaxation represents the convex hull (see Theorem 7), thus it follows that our relaxation is tighter than the Planet relaxation.

Let $y^{\mathcal{CH}}$ denote the upper bound corresponding to our proposed relaxation. Note that at the simplex vertices $\boldsymbol{e}^i$, $y^{\mathcal{CH}} = h(\boldsymbol{e}^i)$. Thus at these vertices, the gap $y^{\mathcal{P}} - y^{\mathcal{CH}}$ can be given by Equation B.4. This difference provides a valid lower bound to the gap in Equation B.2. Note that this gap is proportional to the variance in the weight vector $\boldsymbol{w}$. □

B.2 Comparison to Anderson relaxation

To compare the tightness with respect to the Anderson relaxation (Anderson et al., 2020; Tjandraatmadja et al., 2020), we use a two dimensional example. The example corresponds to the weights in Figure 1 of the main paper. We use $\boldsymbol{x} \in \Delta_2$ with $w = [2, 1]$, $b = -1.25$ and $y = \text{ReLU}(w^T\boldsymbol{x} + b)$. To derive the anderson relaxation for this two dimensional setting, we take inspiration from Example 1 in Appendix of Tjandraatmadja et al. (2020). Figure 1 in our main paper compares the tightness between different relaxations. It can be seen that the Anderson relaxation does not describe the convex hull, and is much looser than our proposed relaxation.

For using the Anderson relaxation, we need to replace the input simplex with the unit hypercube. The relaxation first uses Kuhn triangulation of $[0, 1]^n$ (Todd, 1976), which is used to describe the collection of simplices whose union is $[0, 1]^n$. It requires an exponential number of simplices to describe the unit hypercube. The method then constructs a unique affine interpolation of the function $h(x)$ on each of these simplices. This gives an overall exponential number of inequalities. For our two dimensional example, we need to divide the unit square into two triangles

$\mathbf{T}_1$ and $\mathbf{T}_2$ as shown in Figure B.1. Then constraints r_1 and r_2 are constructed using these two triangles. In contrast, our relaxation works directly with the input simplex and only requires one upper constraint. Figure B.1 shows a visualization for this intuition behind why our relaxation only requires a linear number of constraints in comparison to the exponential constraints of the Anderson relaxation (Anderson et al., 2020). To show a direct correspondence, we construct this figure by modifying Figure 3 from Appendix of Tjandraatmadja et al. (2020).

B.3 Implementation Details

In this section, we provide implementation details for various components of the method.

Conditioning from ℓ_1 to simplex Here we show how ℓ_1 norm perturbations on images can be modelled as simplex perturbations. Let $\boldsymbol{x} \in \mathbb{R}^m$ denote the image input space and let the input perturbations lie within an ℓ_1 ball: $\{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^0\|_1 \leq \epsilon\}$. This input domain can be reformulated as a simplex as

$$\boldsymbol{x} = \boldsymbol{x}^0 + \epsilon M \boldsymbol{z},\ \boldsymbol{z} \in \Delta_{2m},\ \underset{(m \times 2m)}{M} = \begin{pmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 \end{pmatrix}. \tag{B.5}$$

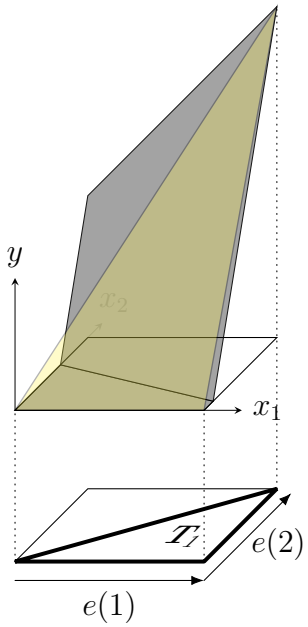
This transformation can be achieved by conditioning the first layer as

$$W\boldsymbol{x} + b = W(\boldsymbol{x}^0 + \epsilon M \boldsymbol{z}) + b \tag{B.6a}$$

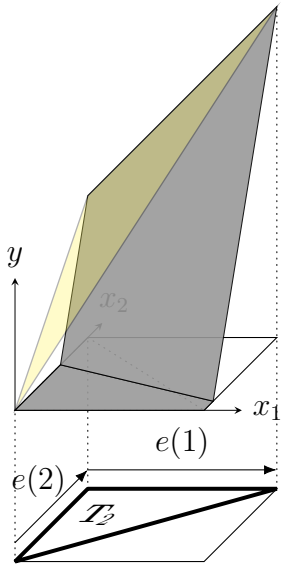
$$= \epsilon W M \boldsymbol{z} + (W\boldsymbol{x}^0 + b) \tag{B.6b}$$

$$= W'\boldsymbol{z} + b', \tag{B.6c}$$

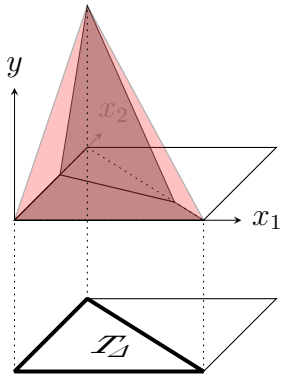
where W' and b' denote the weights and bias of the conditioned layer whose input lies in a simplex.



(a) Constructing r_1 (yellow) using $\mathbf{T}_1$ from the Anderson relaxation.



(b) Constructing r_2 (yellow) using $\mathbf{T}_2$ from the Anderson relaxation.



(c) Constructing r_Δ using the input simplex $\mathbf{T}_\Delta$. Our relaxation can directly model the input simplex $\mathbf{T}_\Delta$.

Figure B.1: Visualisation for the intuition behind why our relaxation requires only a linear number of inequalities, whereas the Anderson relaxation (Tjandraatmadja et al., 2020) requires an exponential number of constraints. Note that since the Anderson relaxation (Tjandraatmadja et al., 2020) relaxation cannot handle the simplex constraint on the input, we have to replace the input constraint with the unit hypercube. Sub-figures B.1a and B.1b show the construction of the simplices in Kuhn triangulation of the unit $[0, 1]^2$ cube, which requires an exponential number of simplices. The bottom figure in each sub-figure shows the simplex and the top figure shows the constraint corresponding to the simplex. Since there are an exponential number of simplices, an exponential number of constraints are required to describe the convex hull.
Sub-figure B.1c shows the input simplex $\mathbf{T}_\Delta$, and the upper bound (in red) corresponding to our relaxation. It can be noted that we only require one upper bound, and a total linear number of inequalities to describe the convex hull for the composition of a linear function with a convex activation function, when the input lies in a simplex.

Conditioning intermediate layers In Section 4.1 of the main paper, we proposed a technique to propagate simplex constraints throughout the network. We derived inequalities of the following form on the intermediate layers:

$$\sum_i x_{k,i} \leq \max_{j \in \{0,\ldots,n_{k-1}\}} \mathbf{1}^T h_k\left(\boldsymbol{e}^j\right) = \alpha_k. \tag{B.7}$$

Here, $x_{k,i}$ denotes the i-th coordinate of the vector of activations x_k at the output of layer h_k. Here we show how to condition the activations of the layers to propagate simplex using the above inequality, assuming that the non-linearity σ is ReLU.

It requires conditioning both layer $k, (h_k)$ and layer $k+1, (h_{k+1})$. We first condition layer k. Let the cut be $\sum_i x_{k,i} \leq \alpha$, where $\alpha_k > 0$. We can write it as

$$\sum_i \frac{x_{k,i}}{\alpha_k} \leq 1 \tag{B.8a}$$

$$\sum_i \frac{\sigma(\sum_j W_{k,ij} x_{k-1,j} + b_{k,i})}{\alpha_k} \leq 1 \tag{B.8b}$$

$$\sum_i \sigma(\sum_j \frac{W_{k,ij}}{\alpha_k} x_{k-1,j} + \frac{b_{k,i}}{\alpha_k}) \leq 1 \tag{B.8c}$$

$$\sum_i \tilde{x}_{k,i} \leq 1. \tag{B.8d}$$

We have achieved $\sum_i \tilde{x}_{k,i} \leq 1$ by down-scaling each $x_{k,i}$ by a factor α_k. Note that we also need to condition layer $k+1, (h_{k+1})$, such that the final output remains the same. This is achieved by up-scaling the weights of layer $k+1$ by a factor α_k.

B.4 Opt-Lirpa Planet Baseline

Algorithm 7 Opt-Lirpa Planet

function OPT-LIRPA_PLANET(Ψ)
  Initialise $\underline{\boldsymbol{a}}$ with values between $\mathbf{0}$ and $\mathbf{1}$
  for $t \in [\![0, t_{max} - 1]\!]$ **do**
    $L^P(\underline{\boldsymbol{a}}^t) =$ OPT-LIRPA_BACKWARD($\Psi, \underline{\boldsymbol{a}}^t$)
    Compute gradients $\frac{dL^P}{d\underline{\boldsymbol{a}}^t}$ via backpropagation
    $\underline{\boldsymbol{a}}^{t+1} \leftarrow$ update gradient ascent (or Adam)
    $\underline{\boldsymbol{a}}^{t+1} \leftarrow \pi\left(\underline{\boldsymbol{a}}^{t+1}\right)$ (projection)
  end for
  return $L^P(\underline{\boldsymbol{a}}^{t+1})$
end function
function OPT-LIRPA_BACKWARD($\Psi, \underline{\boldsymbol{a}}$)
  $f_N \leftarrow \Psi$
  for $k \in [\![n-1, 0]\!]$ **do**
    Set $f_k(\boldsymbol{x}) \leftarrow f^-_{k+1}(\mathbf{u}_k(\boldsymbol{x})) + f^+_{k+1}(\underline{\boldsymbol{a}}_k \odot \mathbb{L}_k(\boldsymbol{x})) + f^c_{k+1}$.
  end for
  $L^P(\underline{\boldsymbol{a}}) = \min_{\boldsymbol{x}_0 \in \Delta} f_0(\boldsymbol{x}_0)$
  return $L^P(\underline{\boldsymbol{a}})$
end function

As mentioned in the main paper, the Opt-Lirpa Planet baseline uses a similar algorithm as our proposed simplex verify algorithm, with the only difference being that it does not have the upper bound corresponding to our relaxation. More precisely, it solves the following optimization problem

$$L^P(\underline{\boldsymbol{a}}) = \min_{\boldsymbol{x}} \quad \Psi(\boldsymbol{x}_{n-1}) \tag{B.9a}$$

$$s.t. \quad \boldsymbol{x}_0 \in \Delta \tag{B.9b}$$

$$\boldsymbol{x}_k \geq \underline{\boldsymbol{a}}_k \odot \mathbb{L}_k(\boldsymbol{x}_{k-1}) \qquad k \in [n-1], \tag{B.9c}$$

$$\boldsymbol{x}_k \leq \mathbf{u}_k(\boldsymbol{x}_{k-1}) \qquad k \in [n-1]. \tag{B.9d}$$

Note that there is only one upper bound, and thus there is no need for an upper weighting coefficient $\overline{\boldsymbol{a}}$. The complete algorithm is shown in Algorithm 7.

B.5 Experimental Appendix

In this section, we present experimental details for the experiments presented in the main paper. We also present a comparison to other baselines.

B.5.1 Comparison Anderson Relaxation

In Anderson et al. (2019a, 2020), the authors proposed a tight relaxation that describes the convex hull of a composition of a linear function of a vector within an ℓ_∞ ball with the ReLU non-linearity. Although the relaxation has an exponential number of constraints, it admits a linear time separation oracle. More recently, de Palma et al. (2021) proposed an efficient dual algorithm for this relaxation. The dual algorithm uses Lagrangian relaxation, and maintains and updates a set of active constraints (Active Sets). Please see de Palma et al. (2021) for more details. In this section, we establish a comparison with the Active Set method (de Palma et al., 2021). This relaxation does not directly handle the simplex constraint on the input. Since it relies on ℓ_∞ constraints on the input to derive the upper bounds, we use the unit hypercube to derive the upper bounds for the first layer. For the intermediate layers, we can use the intermediate upper and lower bounds to form the relaxation. We also compare with the Bigm-adam solver presented in (de Palma et al., 2021), which is a Lagrangian relaxation based dual solver for the unprojected version (Bigm) of the planet relaxation.

The Verification accuracy and time taken are compared to other solvers presented in the main paper, in Table B.1. Note that Bigm-adam and Active Set methods also use the same intermediate bounds as the other methods. We compare the efficiency of the different methods in computing the final layer bounds. We use 850 iterations for Bigm-adam as is used in de Palma et al. (2021). Active set method is initialized with Bigm-adam run for 500 iterations, and the active set is then run for 100 iterations. The active set uses 2 inequalities, which is the same as is used in de Palma et al. (2021).

It can be seen that our Simplex Verify achieves better verified accuracy than the Active Set solver, while being 2 orders of magnitudes faster. One main limitation of the anderson relaxation (Anderson et al., 2020) is that it requires multiple upper bounds to define the convex hull. In comparison, our method only requires a single upper bound to describe the convex hull. In future work, it would be interesting to

explore a combination of upper bounds from the anderson relaxation (Anderson et al., 2020) and our proposed relaxation, for even tighter verification.

B.5.2 Experimental Setting, Hyper-parameters

Hyper-parameter tuning for ℓ_1 perturbation experiments was done on a small subset of the CIFAR-10 test set, on the adversarially-trained Wide network. All the methods use the same intermediate bounds. The intermediate bounds were computed using Opt-Lirpa planet run for 20 iterations. The weighting coefficients are optimized using the Adam optimizer (Kingma and Ba, 2015). The coefficients corresponding to the lower bounds are initialized using CROWN coefficients. The initial and final learning rates are 10^{-5} and 1 respectively.

We compare the efficiency of the different methods in computing the final layer bounds. For a fair comparison, the iterations for both the Lirpa style algorithms (Opt-Lirpa Planet and Simplex Verify) are tuned such that they take the same amount of time. The weighting coefficients corresponding to the lower bounds in both the methods are initialized with CROWN coefficients. Opt-Lirpa Planet is run for 6 iterations, and Simplex Verify is run for 3 iterations. The weighting coefficients are optimized using the Adam optimizer (Kingma and Ba, 2015). The initial and final learning rates for the weighting coefficients corresponding to the lower bounds are 10^-5 and 10 respectively, for Simplex Verify. The initial and final learning rates for the weighting coefficients corresponding to the upper bounds for Simplex Verify are 10^2 and 10^3 respectively.

Details about the network architectures used for the ℓ_1 experiments are presented in Table B.2. Note that the MNIST network uses the same architecture as CIFAR-Wide network, except that it has 1 input channel. These architectures are the same as used in de Palma et al. (2021).

The networks for multi-modal experiment are also trained with adversarial training. Both during training and verification, we allow arbitrary text perturbations from the vocabulary. For these attacks, the vocabulary comprises of the 1000 most frequent words from the training dataset. We also selected a subset of 10 classes

from the Food-101 dataset. These classes include donuts, pizza, french fries, ice cream, onion rings, chicken wings, pad thai, apple pie, chicken curry, waffles.

The models in both, ℓ_1 robustness verification and multi-modal classifier robustness verification, are trained using SLIDE (sparse ℓ_1-descent attack) from Tramer and Boneh (2019). Tuning of the sparsity constant for the SLIDE attack was crucial for training robust networks for the multi-modal classifier on the Food-101 dataset. We used sparsity of 0.3 for all the networks on this dataset. The same sparsity was used for computing the upper bounds for CIFAR networks. We noted that the SLIDE attack performed much better than the normal ℓ_1 PGD attack for training on Food-101 dataset. We also tried the EAD attack for the MNIST network, where SLIDE accuracy was 98.2% and the EAD accuracy was 98.3%. SLIDE performs at par with the EAD attack while being computationally much more efficient than the EAD attack. See Appendix C of Tramer and Boneh (2019) for an empirical comparison between the EAD attack and SLIDE. This was the motivation for choosing the SLIDE attack over the EAD attack.

Dataset		MNIST	CIFAR-10			
Model		Wide	Base	Wide	Deep	Wide
Training		SLIDE	SLIDE	SLIDE	SLIDE	SGD
Accuracy	Nominal	98.80%	75.1%	79.3%	72.1%	74.4%
	Pgd	98.23%	73.5%	77.0%	69.8%	73.3%
Verified Accuracy	Gurobi Planet	31.7%	34.1%	18.4%	11.1%	13.5%
	Gurobi Simplex	45.2%	48.6%	29.4%	13.4%	23.7%
	Bigm-adam (de Palma et al., 2021)	31.4%	33.6%	17.8%	10.6%	13.4%
	Active Set (de Palma et al., 2021)	43.0%	45.5%	26.8%	10.9%	20.9%
	Opt-Lirpa Planet	31.0%	33.7%	17.9%	10.8%	13.5%
	Simplex Verify	44.6%	48.0%	28.8%	13.4%	22.4%
Avg. Verified Time/Sample	Gurobi Planet	74.61s	22.80s	114.92s	86.84s	114.70s
	Gurobi Simplex	72.47s	22.95s	72.17s	59.22s	70.42s
	Bigm-adam (de Palma et al., 2021)	4.46s	4.36s	4.56s	6.80s	4.45s
	Active Set (de Palma et al., 2021)	4.45s	4.63s	4.60s	7.00s	4.63s
	Opt-Lirpa Planet	0.04s	0.04s	0.04s	0.06s	0.04s
	Simplex Verify	0.04s	0.04s	0.04s	0.05s	0.04s

Table B.1: Verified accuracy and verification time of different solvers on MNIST and CIFAR-10 models. We test on the entire test set for MNIST, and random 1000 test images for CIFAR-10. Simplex Verify denotes our proposed solver. Our proposed method achieves much higher verified accuracy in comparison to the state of the art baseline, in the same amount of time.

Network Name	No. of Properties	Network Architecture
OVAL-BASE	1000	Conv2d(3,8,4, stride=2, padding=1) Conv2d(8,16,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 3172)
OVAL-WIDE	1000	Conv2d(3,16,4, stride=2, padding=1) Conv2d(16,32,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 6244)
OVAL-DEEP	1000	Conv2d(3,8,4, stride=2, padding=1) Conv2d(8,8,3, stride=1, padding=1) Conv2d(8,8,3, stride=1, padding=1) Conv2d(8,8,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 6756)
VNN-COMP-Med	1000	Conv2d(3, 32, 5, stride=2, padding=2) Conv2d(32, 128, 4, stride=2, padding=1) linear layer of 250 hidden units linear layer of 10 hidden units (Total ReLU activation units: 16634)
VNN-COMP-Big	1000	Conv2d(3, 32, 3, stride=1, padding=1) Conv2d(32, 32, 4, stride=2, padding=1) Conv2d(32, 128, 4, stride=2, padding=1) linear layer of 250 hidden units linear layer of 10 hidden units (Total ReLU activation units: 49402)

Table B.2: For each incomplete verification experiment, the network architecture used and the number of verification properties tested, a subset of the CIFAR-10 test dataset. Each layer but the last is followed by a ReLU activation function.

C

AutoSimulate: Learning Synthetic Data Generation

Contents

C.1 CLEVR Blender

The different simulator parameters optimized for the Clevr experiment and the underlying distributions are shown in Table C.1. The validation set of 30 images is composed of synthetic images generated with a particular simulator configuration which involved a particular lighting configuration, quality of images, image size, location and material of images. This is shown in the right column of the last row of Fig.C.1. This configuration is hidden during the simulator training and only the validation set is available.

The test set and validation set are generated from the same hidden simulator. The simulator is initialized from a different configuration than the validation

Parameter	Distribution	Range
Number of Samples (Quality)	Categorical	2, 128, 512
Number of Bounces (Quality)	Bernoulli	8, 128
Image Size	Categorical	32x32, 128x128, 256x256
Ambient Light Intensity	Gaussian	0 to 100
Back Light Intensity	Gaussian	0 to 100
Each Object Location (3 Objects)	Gaussian	-10 to 10
Each Object Material (3 Objects)	Bernoulli	Rubber, Metal

Table C.1: Distributions over Clevr simulator parameters

simulator. During training, the simulator learns to generate data which is optimal for training a neural network for perforimng well on the validation set. Figure C.1 shows the evolution of the simulator with training. It can be seen that the simulator starts from poor lighting, quality and location of objects and improves itself during training, along with improving the segmentation performance on the validation set.

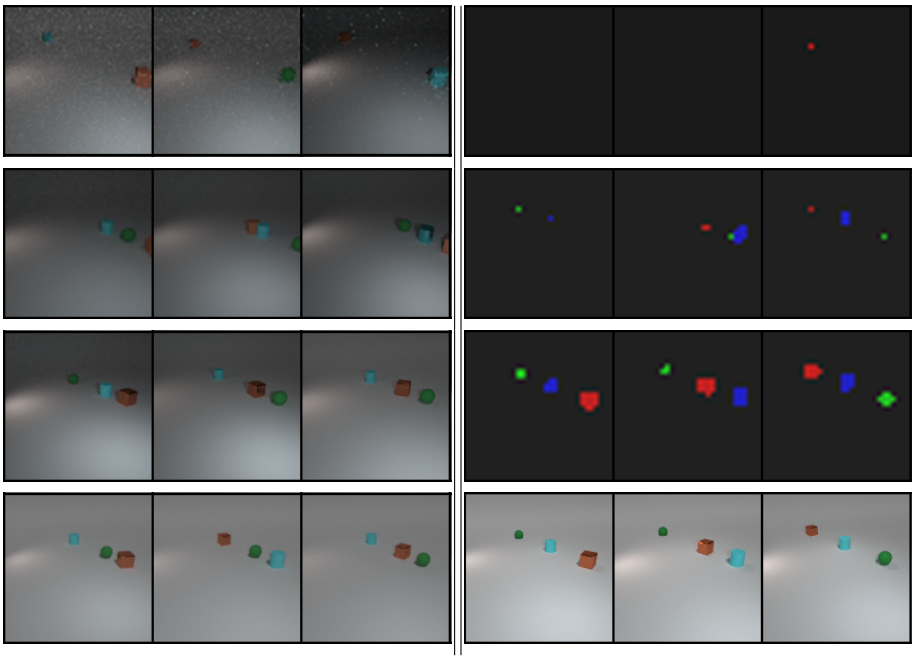


Figure C.1: Evolution of CLEVR simulator during training using AutoSimulate. `Left`: rows show the images generated by the simulator at different iterations during simulator training. It can be seen that the simulator evolves from poor lighting and low quality (Row 1 and 2) to good lighting and quality (Row 4) and also learns the object positions, as are in the validation set. Row 4 shows images generated from the final trained simulator. These images are used to train a semantic segmentation model. `Right`: semantic segmentation model output on 3 validation images (Row 4) across simulator training iterations. Row 3 provides outputs from the final trained segmentation model.

C.2 Photorealistic Renderer

We briefly describe the rendering technique used to generate photo-realistic images in the main set of experiments. There are three essential components to the rendering system. First component is acquiring accurate 3D CAD models of objects; which involves assigning accurate geometric properties, along with material, texture and color to the objects. Then, these 3D objects models are rendered within a realistic scene, which is essential for capturing the context around these objects. A realistic scene should also have accurate geometric shapes, background objects, along with accurate materials, texture and lights.

Second component is the proper placement of these objects and camera within a scene such that the images renderer are plausible. Objects are placed in the scene such that rigid body properties are not violated, for example, they should not intersect with each other. A physics engine (Nvidia PhysX) is used to generate possible object arrangements. A set of camera positions is randomly generated for each object arrangement.

Finally, in order to generate photorealistic images, accurate simulation of reflectance and light properties, including shadows, soft shadows, inter-reflections etc, is necessary. Physically based rendering method, for example Arnold Georgiev et al. (2018), is then used to generate highly photorealistic images. This system uses ray tracing which relies on shooting rays through within a scene. The quality of rendered images depends on the number of rays sampled and the number of bounces of lights from (diffused and specular) surfaces. A user can control the quality of images by specifying the values for these parameters. More details about data generation with the simulator can be found in the work of Hodaň et al. (2019) and Arnold Georgiev et al. (2018).

Implementation Details

For calucation of the inverse hessian vector product using the Conjugate Gradient (CG) algorithm, we used the `fmin_ncg` function from python library `scipy.optimize`

Method	mAP (Faster-rcnn)	mAP (Yolo)
REINFORCE (LTS)	51.8	37.2
Bayesian Optimization	46.0	37.5
Random Search	50.3	36.8
Ours	**55.1**	**45.9**

Table C.2: Effect of Network Architecture

[1]. Furthermore, a standard implementation of Grid Search from Scikit-learn Python library Pedregosa et al. (2011) was used to tune the hyper-parameters for all the baselines and our algorithm, and the best results are reported for every algorithm. The size of the dataset K generated in each iteration is tuned over the values $20, 50$ and 70. The simulator learning rate α is tuned over the values $0.01, 0.1$ and 10. For LTS, the number of datasets generated at each iteration is tuned over the values $5, 10, 40$ and 50. The weight decay regularization λ of the hessian is tuned over 0.1 and 10. We observed a standard deviation of around 3% in the Test mAP with this hyper-parameter tuning.

C.2.1 More Ablation Studies

We now show another experiments to compare and evaluate the robustness and performance of different algorithms.

Effect of Network Architecture

In this experiment we evaluate the effect of the network architecture on simulator training for different methods. In particular, we show results using two networks: Faster-rcnn with Resnet50-fpn backbone and YOLO with 112 layers in Table C.2. We observe that our methods is much better than all the other baseline methods across different network architectures, thereby demonstrating the generality of the method.

C.3 Extended Related Work

Kar et. al. Kar et al. (2019) proposed an interesting method that learns to transform scene graphs sampled from the given probabilistic grammar of scenes

[1] `https://het.as.utexas.edu/HET/Software/Scipy/generated/scipy.optimize.fmin_ncg.html`

with the goal of simultaneously optimizing performance on downstream task (using REINFORCE-like gradient estimator) and matching the distribution of simulated images to real images. The method aims to reduce the content gap between simulated and real images, however it is still limited by the slow REINFORCE-like gradient estimator. In contrast, our contribution lies in making a more efficient algorithm for optimizing performance on a downstream task. Furthermore, we also focus on optimizing the parameters that impact realism of rendered images such as rendering quality, material of objects, illumination in the environment, while also optimizing content parameter such as location of objects in the scene. We believe both the *autosimulate* and *meta-sim* approaches are orthogonal and can benefit from each other. Efficient optimization method proposed in the autosimulate method can replace REINFORCE in the meta-sim approach. Similarly, distribution matching loss between simulated and real images from the meta-sim work can be used within autosimulate framework. These are interesting directions. However, a complete validation of these ideas is beyond the scope of this paper.

C.4 Gradient of Expectation for different distributions

We discuss a technique to compute the term $\frac{d}{d\psi}\,\mathbb{E}_{s\sim p_\psi}[f(s)]$ which is the gradient of expectation of a function over a distribution, with respect to parameters of the distribution. For continuous distributions we can directly use REINFORCE Williams (1992), as follows:

$$\frac{d}{d\psi}\,\mathop{\mathbb{E}}_{s\sim p_\psi}[f(s)] = \mathop{\mathbb{E}}_{s\sim p_\psi}[f(s)\frac{d}{d\psi}\log p_\psi(s)] \tag{C.1}$$

Gaussian Distribution

For the mean of Gaussian distribution, we have
$\frac{\partial}{\partial\psi}\log p_\psi(x) = \Sigma^{-1}(x-\psi)$, where ψ is the mean and Σ is the covariance matrix. Putting this into Eq. C.1 we get:

$$\frac{d}{d\psi}\,\mathop{\mathbb{E}}_{s\sim p_\psi}[f(s)] = \mathop{\mathbb{E}}_{s\sim p_\psi}[f(s)\,\Sigma^{-1}(s-\psi)] \tag{C.2}$$

For discrete distributions we can derive it similarly Schulman et al. (2015); Rezende et al. (2014). Here we give the derivations for Bernoulli and Categorical distributions which we used in our experiments.

Bernoulli Distribution

Let s take value 1 with probability ψ and value 0 with probability $1-\psi$, defined as $p_\psi(s) = \psi^s(1-\psi)^{(1-s)}$. The gradient over expecation term can be computed as:

$$\begin{aligned}\frac{\partial}{\partial\psi}\mathop{\mathbb{E}}_{s\sim p_\psi}[f(s)] &= \frac{\partial}{\partial\psi}(1-\psi)f(0) + \frac{\partial}{\partial\psi}\psi f(1)\\ &= f(1) - f(0)\end{aligned} \tag{C.3}$$

Categorical Distribution

Let the parameter be $\psi = (\psi_1, \psi_2, ..., \psi_K)$ s.t $\psi_1 + \psi_2 + ... + \psi_K = 1$ where ψ_i is the probability of seeing element i. Let $[s = i]$ be 1 if $s = i$ and 0 otherwise, then:

$$\begin{aligned}p_\psi(s=i) &= \prod_{i=1}^{K}\psi_i^{[s=i]}\\ \mathop{\mathbb{E}}_{s\sim p_\psi}[f(s)] &= \sum_{i=1}^{K}[s=i]\psi_i f(s)\\ \frac{\partial}{\partial\psi_i}\mathop{\mathbb{E}}_{s\sim p_\psi}[f(s)] &= \frac{\partial}{\partial\psi_i}[s=i]\psi_i f(s)\\ &= f(i)\end{aligned} \tag{C.4}$$

D

Meta Learning Deep Visual Words for Fast Video Object Segmentation

Contents

D.1 Additional ablation studies

In this section, we present an additional experiment on the temporal consistency of our visual words, the effect of online adaptation and the size of the visual word dictionary.

D.1.1 Temporal consistency of our visual words

In this experiment we aim to infer whether our visual words remain in the same region/part on the object throughout the video. We use the Physical Parts Discovery Dataset Del Pero et al. (2016) for this experiment. This dataset contains videos

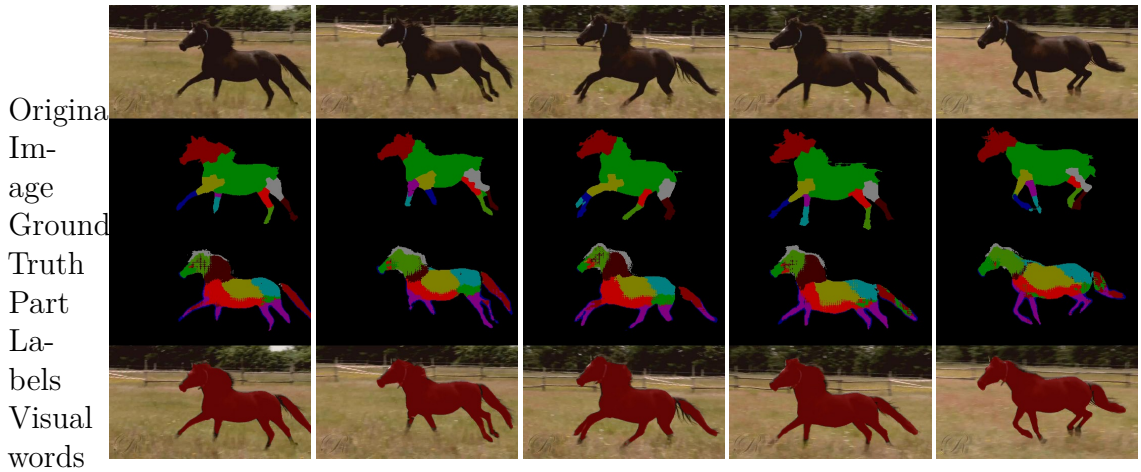


Figure D.1: Qualitative examples of our experiment on the temporal consistency of visual words on the Physical Parts Discovery (PPD) Dataset Del Pero et al. (2016) Note how the visual words that our model learns in an unsupervised manner (third row) are consistent over time. The PPD dataset contains ground-truth annotations for animal classes (second row) which we leverage for our experiment. Note that our visual words will not correspond exactly to the ground truth parts as we learn our visual words in an unsupervised manner.

for two object classes, namely, "tiger" and "horse". There are 16 videos for each class, with 8 videos with the animal facing right and the other 8 videos with the animal facing left. Ground truth annotation for 10 body parts of the object is provided for a subset of frames in the video.

We use this dataset, as it is the only one that we are aware of that annotates the parts of an object throughout the video. However, as our method produces object parts in an unsupervised manner, we first form a mapping from each of our visual words to a ground truth object part. Thereafter, we evaluate the consistency of these visual-word-to-object-part mappings over time.

Mapping visual words to ground truth object parts We initially run our algorithm on the first frame of the video, to obtain k visual words for the object. We then map each visual word to a ground truth object part. This assignment is done based on the majority vote from all the pixels in that visual word, *i.e.* a visual word is assigned to the part which it has the maximum spatial overlap with it. This is then the ground truth assignment of the visual words to the object parts

Dictionary Size (k)	10	15	20	30	40	50	60	80	100
Part consistency score (%)	77.4	75.1	74.1	71.9	71.1	71.1	70.9	69.9	69.6

Table D.1: Part consistency score as a function of the number of visual words, k. The relatively high score (the maximum is 100) suggests that our method indeed forms visual words that consistently correspond to the same parts of the tracked object throughout the video. This is performed on the Physical Parts Discovery Dataset Del Pero et al. (2016).

and will be used next for evaluation.

Evaluation For all other frames of the video, we run our method and match each pixel to a visual word of the object. We then find the part that these visual words now belong to, which is following the previous paragraph performed according to the maximum spatial overlap. We then calculate how many visual words still belong to the part that they were assigned to in the first annotated frame. The percentage of these consistent mappings, averaged over the entire dataset, is our performance metric, which we denote the "Part consistency score". This metric measures the consistency of our visual words over time, as it indicates if a visual word is still in the same region of the object as it was in the first frame.

Results Table D.1 performs this experiment for different numbers of visual words, k. For $k = 10$, the accuracy is 77.4%, showing that our method is indeed consistently tracking parts of the object. This is also shown visually in Fig. D.1 and in the supplementary video. Table D.1 also shows that the part tracking accuracy decreases as k is increased. We believe this is because each of the regions corresponding to a visual word decreases as k is increased, and thus there is higher variance in visual-word-to-object-part mapping.

D.1.2 Effect of Online Adaptation

Online adapation, as described in Sec. 5.3.4, updates the dictionary of visual words to account for appearance changes in the object throughout the video. Figure D.2 shows an example of the online adaptation of our model on the DAVIS-2017 dataset.

Figure D.2: The effect of online adaptation on the representation of dynamic objects. As the object pose changes over time, newer visual words (denoted by gray and light-blue colors) are learned and added to the dictionary of visual words.

Interval (δ)	NA	30	20	10	5	2	1
$\mathcal{J}(\%)$	55.8	58.6	59.2	61.3	**63.9**	63.7	62.8

Table D.2: The effect of online adaptation on model performance. δ denotes the frame interval before every step of online adaptation. Small values of δ help the model to adapt to quickly changing scenes, but too small a δ can introduce noise into the visual words. NA: No online adaptation.

Table D.2 shows how updating the dictionary, $\mathcal{M}$, improves performance. Smaller values of the update interval, δ, means $\mathcal{M}$ is updated more frequently and thus helps the system smoothly adapt to dynamic scenes and fast-moving objects. However, very small values of δ, such as $\delta = 1$ also increase the chance of adding noisy visual

Outlier Removal	Online Adaptation	$\mathcal{J}(\%)$
✗	✗	55.8
✗	✓	60.4
✓	✓	63.9

Table D.3: The effect of outlier removal and online adaptation

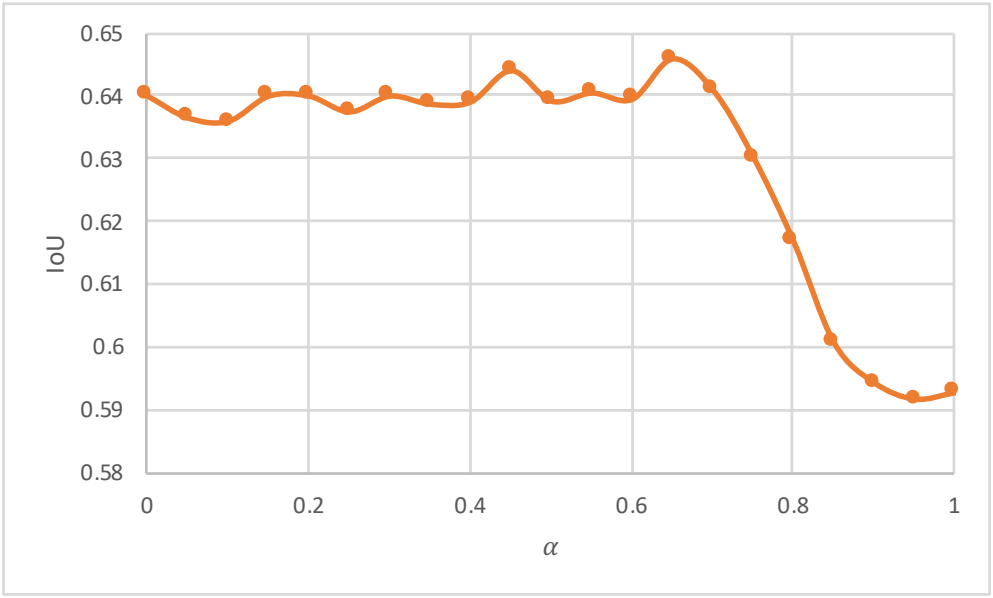


Figure D.3: The effect of the α hyper-parameter used in the online adaptation of on our method. Note how our IoU on the DAVIS-2017 validation set barely changes for $\alpha \in [0, 0.7]$ showing that our algorithm is not very sensitive to this hyper-parameter.

words, which may explain why $\delta = 5$ performs the best. Table D.3 also shows that our simple "outlier removal" step improves results as well, by encouraging spatio-temporal consistency from one frame to the next.

Note that our online adaptation system adds little overhead as it simply updates the existing dictionary of visual words. No additional forward or backward passes through the network are required, unlike methods such as Voigtlaender and Leibe (2017).

Effect of α on online adaptation Figure D.3 shows the effect of distance threshold in online adaptation, α, on our accuracy on DAVIS-2017. It shows that our algorithm is robust to the choice of alpha, performing similarly in a wide range of values when α lies in the interal $[0, 0.7]$, thus motivating our decision to use $\alpha = 0.5$ in our experiments. Although our method is tolerant to a wide range of α values, setting it too high ($\alpha \geq 0.8$) introduces noisy visual words into online adaptation process which harms performance.

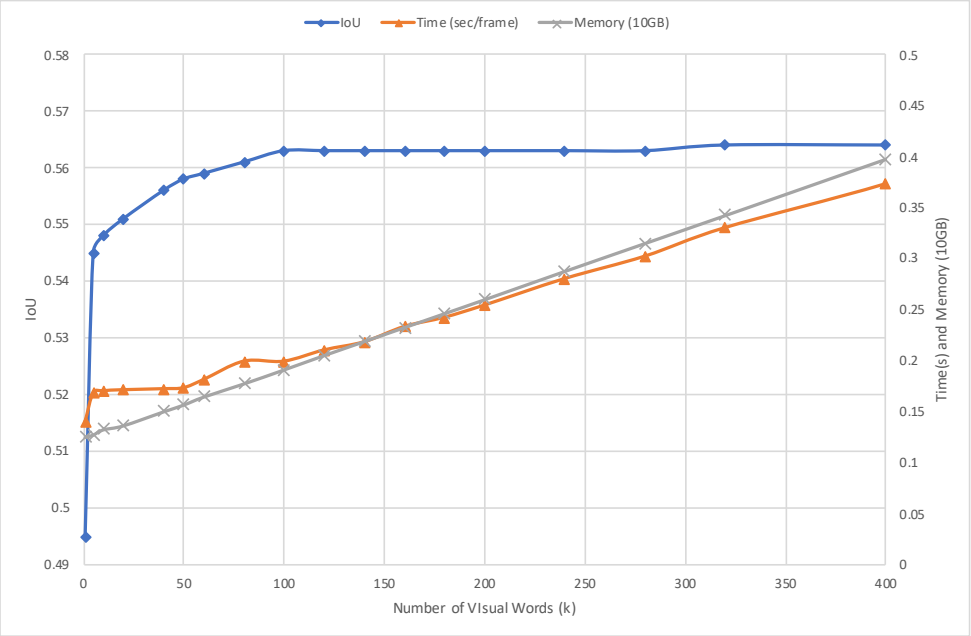


Figure D.4: The effect of visual word dictionary size hyper-parameter on our algorithm's accuracy (IoU), runtime (s) and memory consumption (GB). The accuracy, in terms of the IoU ($\mathcal{J}$) starts saturating around $k = 50$. However, the runtime and memory increase linearly with the number of clusters, k.

D.1.3 Effect of visual word dictionary size

Table 3 of the main paper already showed the effect of the visual word dictionary size on performance. Figure D.4 examines this in more detail by showing the accuracy (measured by the IoU, $\mathcal{J}$), the runtime in seconds and the memory consumption as a function of the size of the visual word dictionary. We can see that the accuracy starts saturating after around $k = 50$ clusters. However, the runtime and memory consumption increase linearly as the number of visual words is increased. So we use $k = 50$ clusters, as it provides a good balance between accuracy and speed.

Additionally, Fig. D.5 presents some of the visual words, corresponding to object parts, that are automatically learned by our model. Finally, Fig. D.6 shows the qualitative effect on our final segmentation results by increasing the number of visual words in the dictionary.

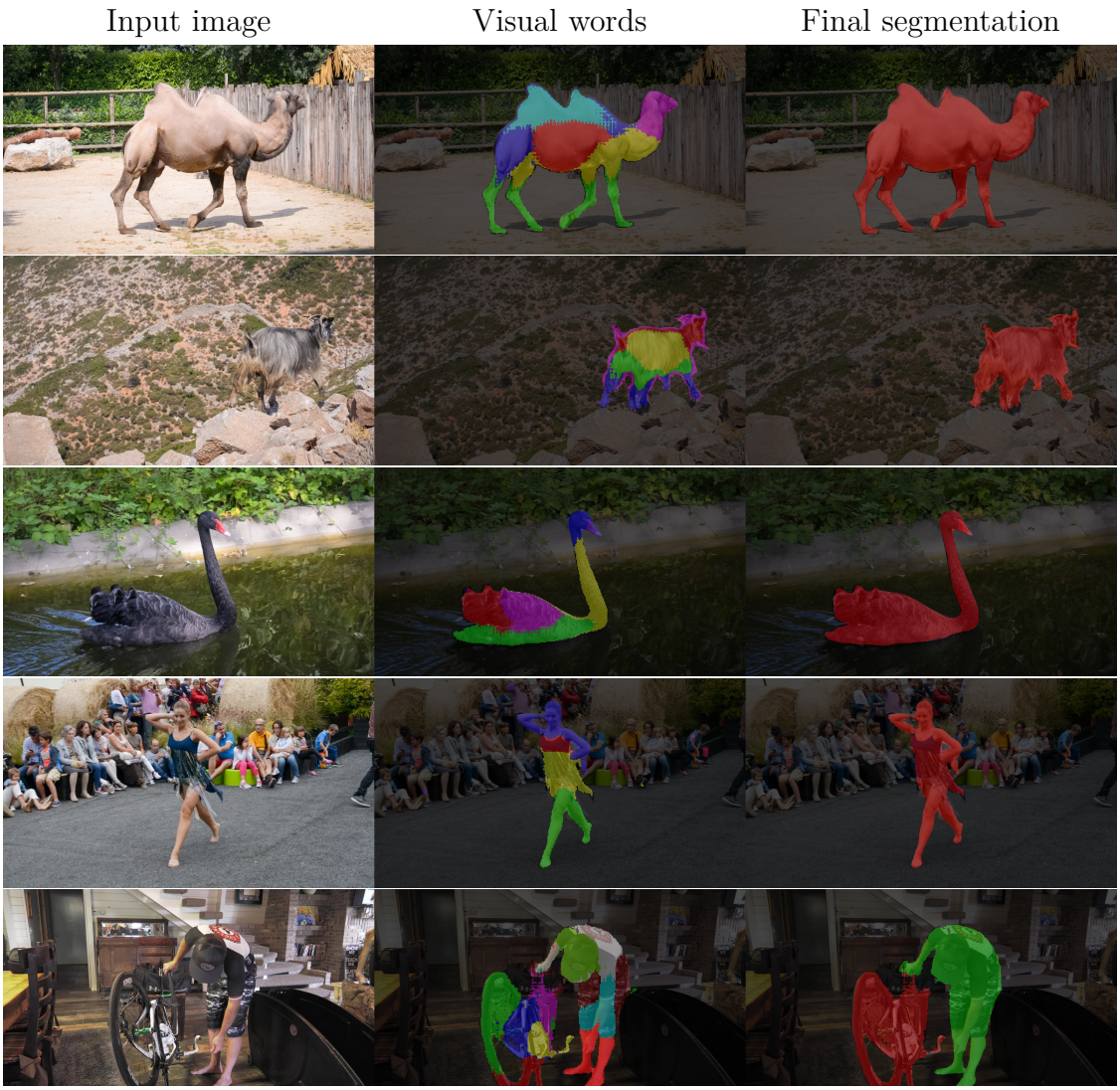


Figure D.5: Visual words formed by our model. Each row represents a video from DAVIS-2017 dataset, with the original image (left), the object parts formed by our model in different colors (middle), and the segmentation output (right) obtained using our model. It can be seen that our model forms meaningful visual words which represent body parts in objects.

D.2 Qualitative Results

Figures D.7 and D.8 present further qualitative comparisons of our method with RGMP Wug Oh et al. (2018).

D.3 Additional quantitative results

Tables D.4, D.5 and D.6 present per-sequence results on the DAVIS-2016, DAVIS-2017 and YouTube-Objects datasets respectively.

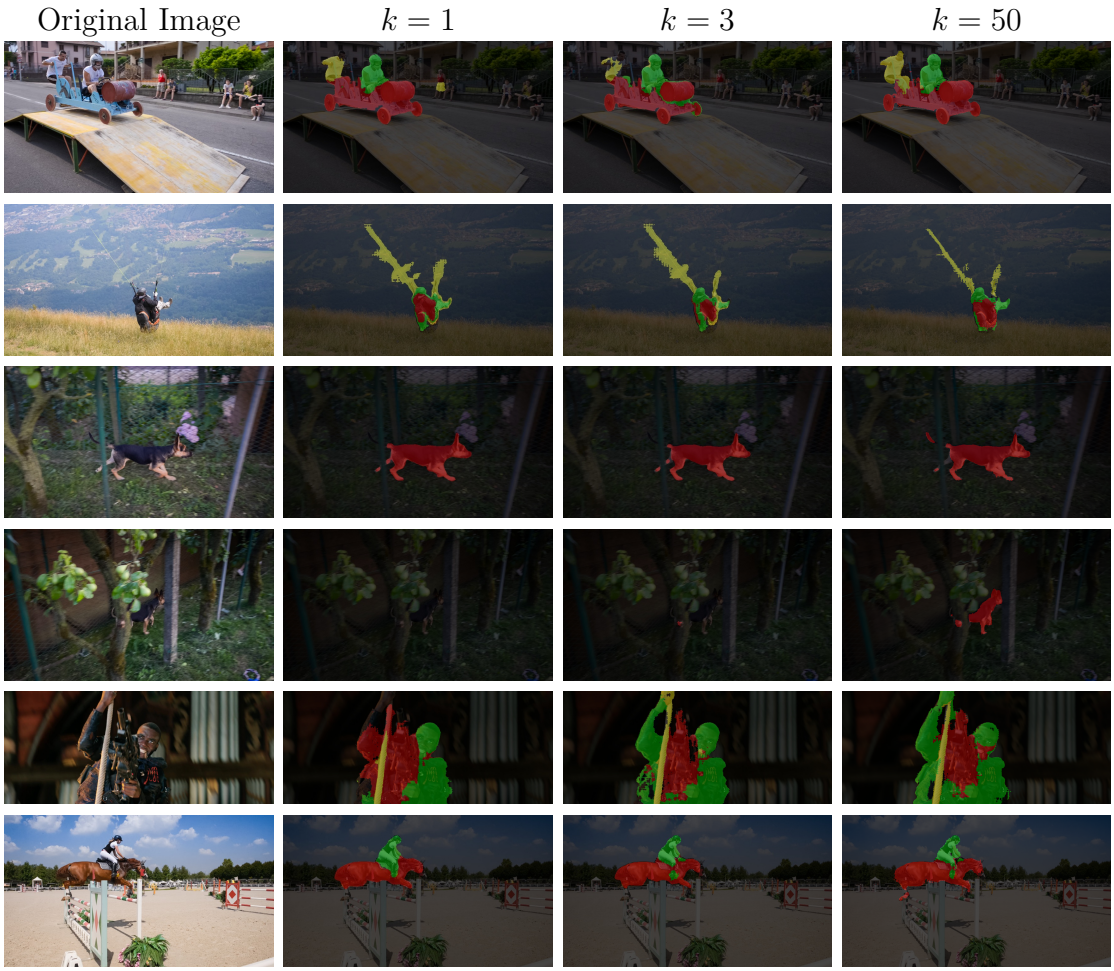


Figure D.6: The effect of visual word object representation on qualitative segmentation outputs. This figure shows that increasing the size of the visual word dictionary (K) improves the representation of the object, and thus improves segmentation outputs (qualitative). This is because it can better capture the intra-object variance. For instance, the lost face of the human in yellow (first row), the lost tail of the dog (third row), and the missing legs of the horse (last row) have been recovered in the last column (50 visual words), because of this property. Similarly, our method can address partial occlusions by representing different object parts using visual words, and tracking them robustly over the video (fourth row). All the visual words are learned in an unsupervised manner to represent object parts, as described in the main paper. The results are obtained without any fine-tuning.

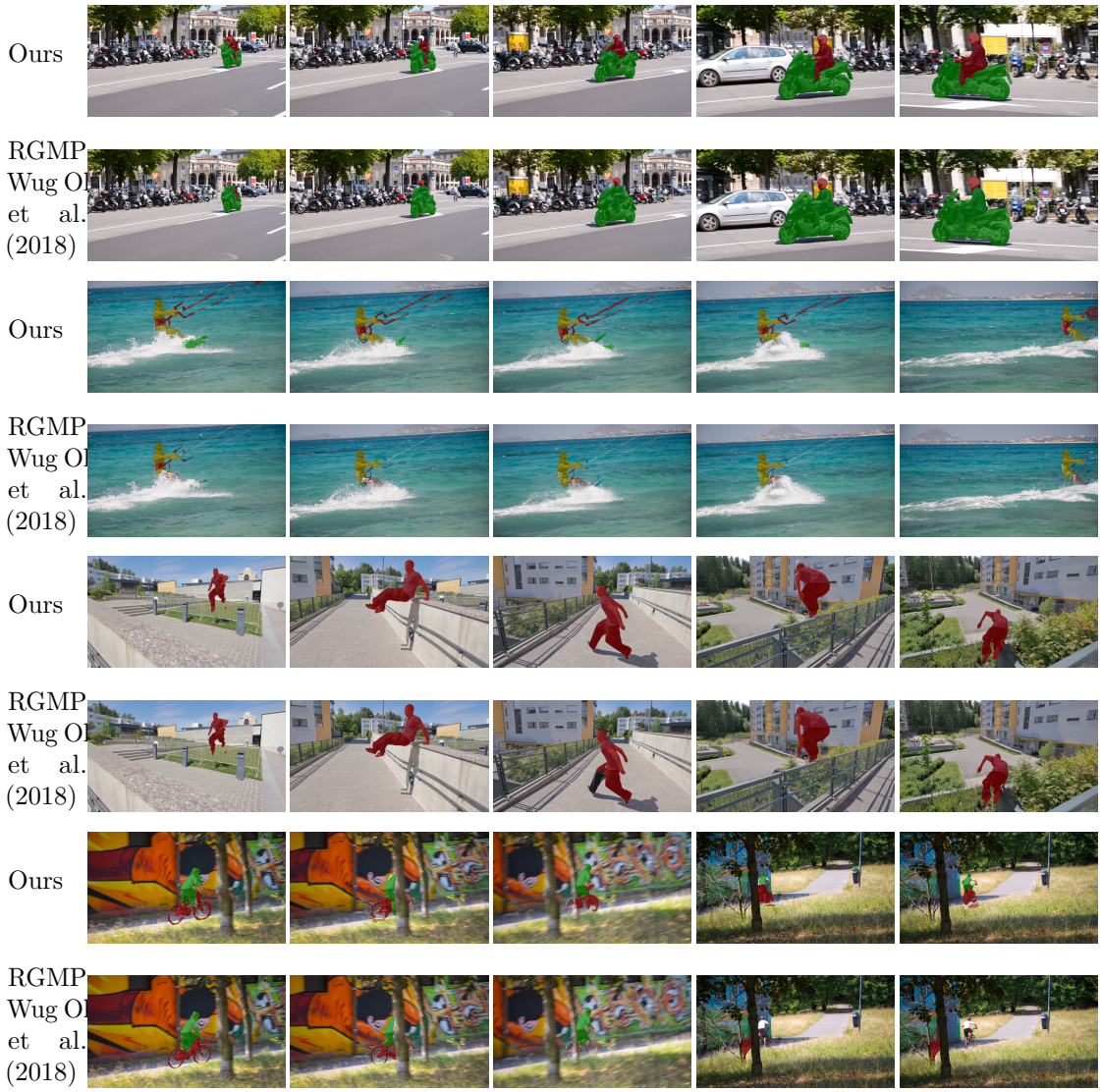


Figure D.7: Success cases of our method, and comparison to RGMP Wug Oh et al. (2018). In each of these videos, our method is able to accurately track the objects labelled in the first frame throughout the video. *First video:* Our algorithm accurately segments the person throughout the video, whilst RGMP cannot deal with the scale and viewpoint changes of the person and mistakes him for the motorbike. *Second video:* Our method is able to segment the kite-surfing harness and wires whilst RGMP loses track of these fine structures. Additionally, note how we are able to segment the heavily-occluded surf-board throughout the video, unlike RGMP. *Third video:* Both methods perform well on this example. *Fourth video:* RGMP loses track of the cyclist from the fourht frame onwards, whereas our method is robust to this occlusion. Mask propagation methods, such as RGMP, struggle with such occlusions. Our representation of objects as visual words is more robust in these situations. Full video results of these clips are included in the supplementary video.

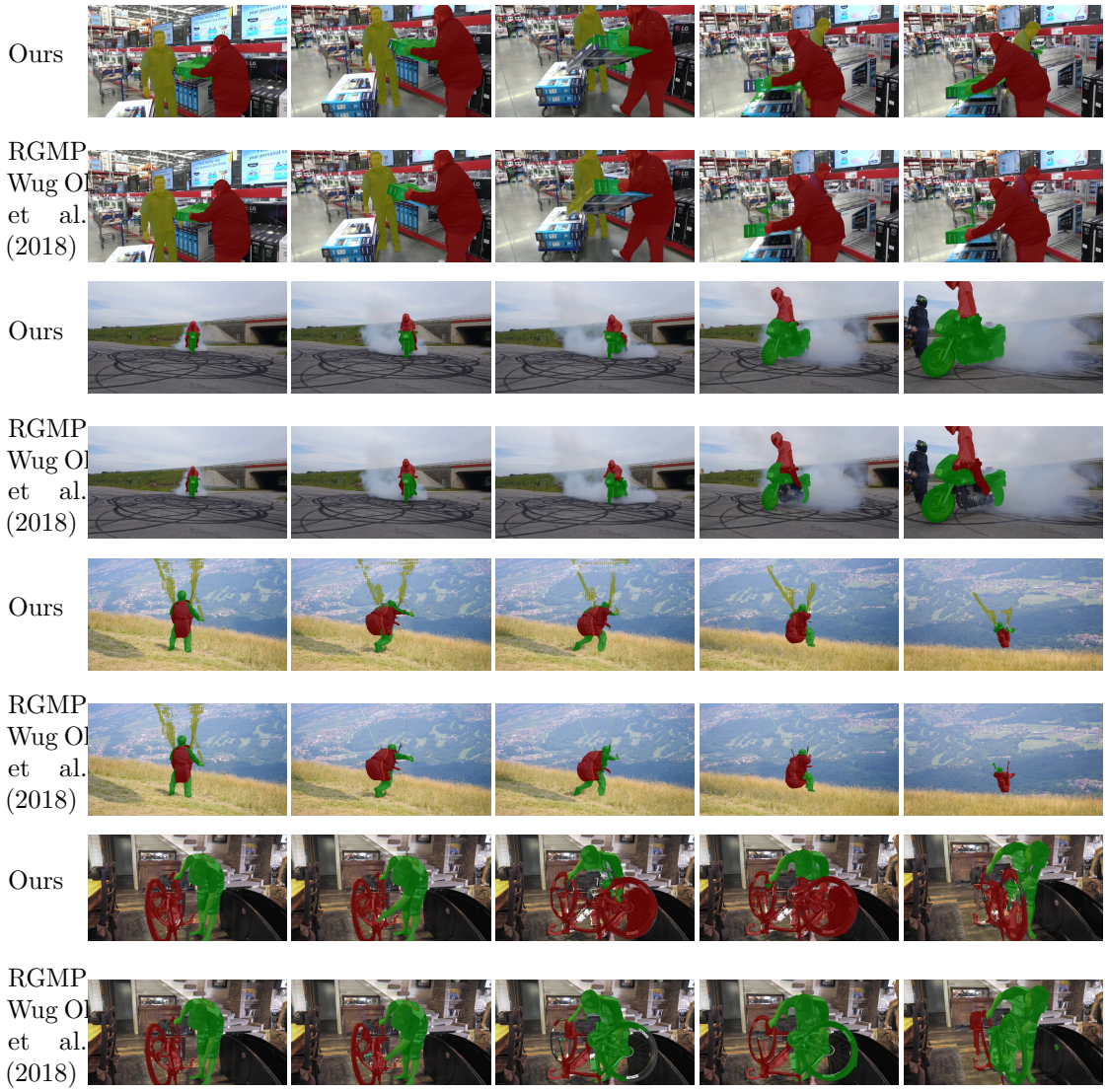


Figure D.8: Failure cases of our method and RGMP Wug Oh et al. (2018). *First video:* Note how our method does not properly segment the green box through the video. RGMP, on the other hand, loses track of the whole box, and also cannot deal with the two people occluding each other in the last two frames. *Second video:* In the fourth and fifth frames, our method confuses the cyclists legs and motorbike. RGMP segments the person properly, but not the entire motorbike. *Third video:* Our algorithm struggles to segment the fine structures of the parachute. RGMP, on the other hand, completely loses track of the parachute after the first frame. *Fourth video:* Our segmentation of the bicycle (particularly its spokes) is not very accurate. RGMP, on the other hand, makes a larger error between the bicycle and person when they occlude each other from the third frame onwards. Full video results of these clips are included in the supplementary video.

Sequence	Blackswan	Bmx-Trees	Breakdance	Camel	Car-Roundabout	Car-Shadow	Cows	Dance-Twirl	Dog	Drift-Chicane
F	0.98	0.77	0.70	0.85	0.92	0.99	0.98	0.81	0.95	0.72
J	0.94	0.55	0.69	0.81	0.96	0.96	0.94	0.81	0.93	0.65

Sequence	Drift-Straight	Goat	Horsejump-High	Kite-Surf	Libby	Motocross-Jump	Paragliding-Launch	Parkour	Scooter-Black	Soapbox
F	0.83	0.91	0.93	0.64	0.89	0.65	0.45	0.94	0.82	0.89
J	0.90	0.87	0.82	0.62	0.75	0.82	0.61	0.89	0.85	0.90

Table D.4: Per-sequence video object segmentation results for DAVIS-2016 Perazzi et al. (2016) dataset.

Sequence	Bike-Packing_1	Bike-Packing_2	Blackswan_1	Bmx-Trees_1	Bmx-Trees_2	Breakdance_1	Camel_1	Car-Roundabout_1	Car-Shadow_1	Cows_1	Dance-Twirl_1	Dog_1	Dogs-Jump_1	Dogs-Jump_2	Dogs-Jump_3
F	0.81	0.63	0.98	0.72	0.73	0.81	0.88	0.96	0.99	0.98	0.83	0.95	0.12	0.70	0.92
J	0.59	0.72	0.95	0.33	0.57	0.76	0.82	0.97	0.95	0.94	0.82	0.92	0.09	0.55	0.85

Sequence	Drift-Chicane_1	Drift-Straight_1	Goat_1	Gold-Fish_1	Gold-Fish_2	Gold-Fish_3	Gold-Fish_4	Gold-Fish_5	Horsejump-High_1	Horsejump-High_2	India_1	India_2	India_3	Judo_1	Judo_2
F	0.70	0.87	0.90	0.58	0.59	0.49	0.60	0.60	0.93	0.89	0.45	0.41	0.56	0.84	0.77
J	0.68	0.91	0.86	0.62	0.56	0.60	0.65	0.75	0.78	0.67	0.46	0.45	0.60	0.81	0.76

Sequence	Kite-Surf_1	Kite-Surf_2	Kite-Surf_3	Lab-Coat_1	Lab-Coat_2	Lab-Coat_3	Lab-Coat_4	Lab-Coat_5	Libby_1	Loading_1	Loading_2	Loading_3	Mbike-Trick_1	Mbike-Trick_2	Motocross-Jump_1
F	0.67	0.26	0.94	0.44	0.51	0.49	0.41	0.62	0.91	0.89	0.55	0.89	0.75	0.76	0.57
J	0.28	0.16	0.72	0.07	0.13	0.55	0.43	0.66	0.76	0.93	0.47	0.84	0.62	0.75	0.48

Sequence	Motocross-Jump_2	Paragliding-Launch_1	Paragliding-Launch_2	Paragliding-Launch_3	Parkour_1	Pigs_1	Pigs_2	Pigs_3	Scooter-Black_1	Scooter-Black_2	Shooting_1	Shooting_2	Shooting_3	Soapbox_1	Soapbox_2	Soapbox_3
F	0.52	0.84	0.83	0.58	0.95	0.55	0.62	0.84	0.77	0.77	0.31	0.55	0.68	0.82	0.81	0.84
J	0.69	0.77	0.58	0.15	0.90	0.52	0.48	0.93	0.64	0.80	0.33	0.59	0.44	0.84	0.75	0.76

Table D.5: Per-sequence video object segmentation results for DAVIS-2017 Pont-Tuset et al. (2017) dataset.

Sequence	Aeroplane	Bird	Boat	Car	Cat	Cow	Dog	Horse	Motorbike	Train
$\mathcal{J}(\%)$	87.1	84.7	81.0	82.9	78.8	76.3	80.3	72.9	77.8	89.5

Table D.6: Per-sequence video object segmentation results for Youtube-Objects dataset Prest et al. (2012); Jain and Grauman (2014).

E

Alpha MAML: Adaptive Model-Agnostic Meta-Learning

Contents

E.1 Appendix

We also derive the Alpha MAML update equations for the bigger case of multiple tasks in one batch as follows, where i denotes the iteration number and t is used to denote the task index:

$$\begin{aligned}
\theta'_t &= \theta_{i-1} - \alpha_i \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{train(t)}}(f_{\theta_{i-1}}) \\
\alpha_{i+1} &= \alpha_i + \alpha_{\text{hyperlr}} \sum_{\mathcal{T}_t \sim p(\mathcal{T})} \nabla_{\theta'_t} \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t}) . \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{train(t)}}(f_{\theta_{i-1}}) \\
\beta_i &= \beta_{i-1} + \beta_{\text{hyperlr}} \sum_{\mathcal{T}_t \sim p(\mathcal{T})} \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t}) . \nabla_{\theta_{i-2}} \mathcal{L}_{\mathcal{T}_{test(i-1)}}(f_{\theta'_t}) \\
\theta_i &= \theta_{i-1} - \beta_i \sum_{\mathcal{T}_t \sim p(\mathcal{T})} \nabla_{\theta_{i-1}} \mathcal{L}_{\mathcal{T}_{test(t)}}(f_{\theta'_t})
\end{aligned} \tag{E.1}$$

In the update for β, the second gradient is the previous step's gradient.

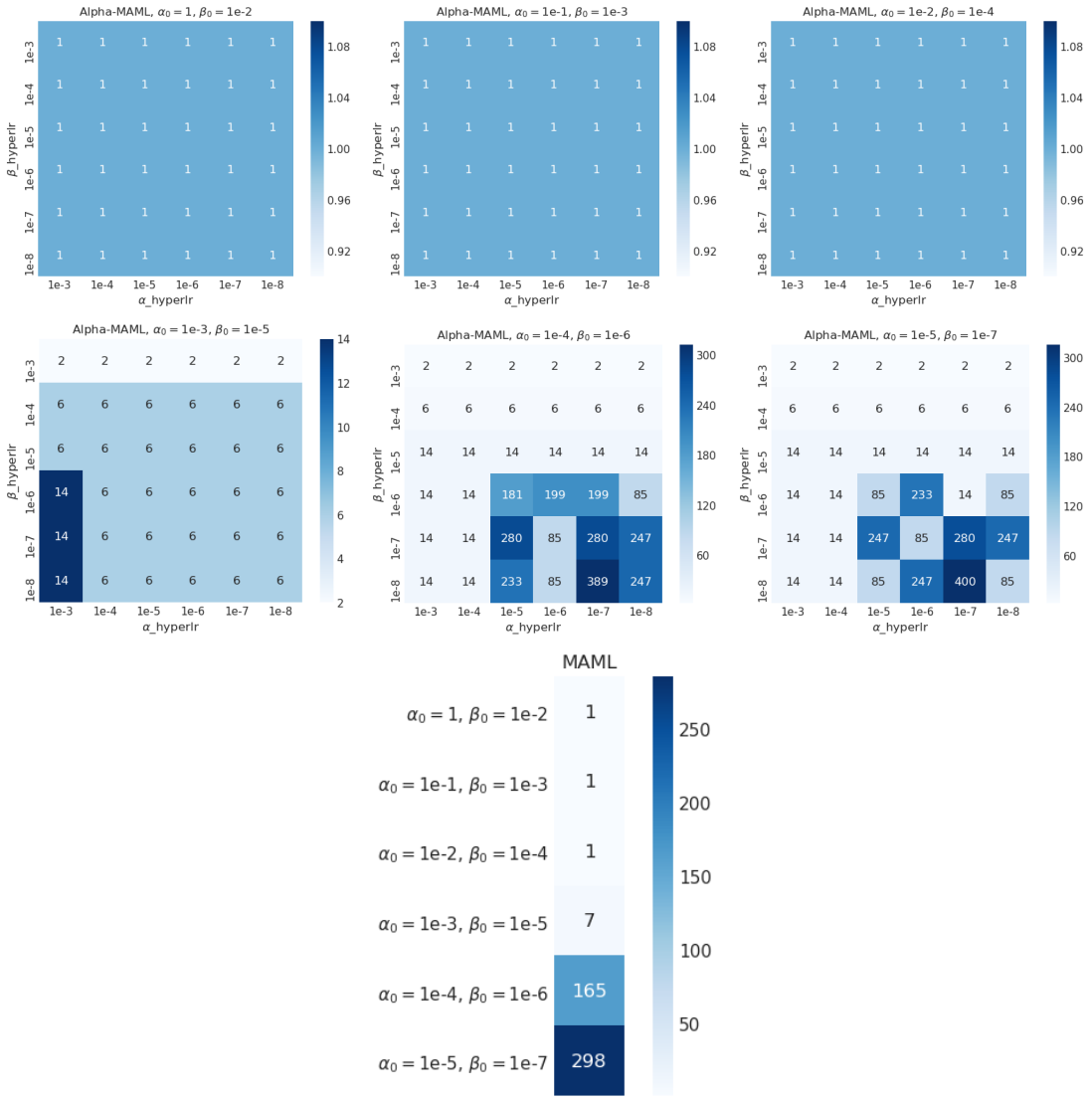


Figure E.1: Grid Search for selecting the hyper-parameters in MAML and Alpha-MAML: shows the number of iterations to converge to a training loss of 2.99 for Omniglot dataset. It can be seen that as we go far from the tuned hyper-parameter range, converge of Alpha-MAML is better than MAML. Specially for the case of $\alpha_0 = 1e-4, \beta_0 = 1e-6$ Alpha-MAML converged in less than 15 iterations for a wide range of α_{hyperlr} and β_{hyperlr}, whereas MAML takes 165 iterations. Same is the case with $\alpha_0 = 1e-5, \beta_0 = 1e-7$, for most of the choices of α_{hyperlr} and β_{hyperlr}, Alpha-MAML converged in lesser number of iterations than MAML.

Implementation details The network has four modules with 3×3 convolutions and 64 filters. This is followed by batch normalization, a ReLU activation, and strided convolutions. The final output is fed into a softmax layer. The images are downsampled to 28×28, and the dimensionality of the last hidden layer is 64.

An augmentation scheme similar to original MAML implementation is applied, where images are augmented with 90 degrees rotated images.

Bibliography

Mohsan S. Alvi, Andrew Zisserman, and Christoffer Nellåker. Turning a blind eye: Explicit removal of biases and variation from deep neural network embeddings. *CoRR*, 2018.

Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. *PLDI*, 2019a.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. *Conference on Computer Vision and Pattern Recognition*, 2018.

Ross Anderson, Joey Huchette, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. In *Integer Programming and Combinatorial Optimization*. Springer International Publishing, 2019b.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 2020.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. In *International Conference on Learning Representations*, 2019.

David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *ACM-SIAM Discrete Algorithms*, pages 1027–1035, 2007.

Linchao Bao, Baoyuan Wu, and Wei Liu. Cnn in mrf: Video object segmentation via inference in a cnn-based higher-order spatio-temporal mrf. In *CVPR*, 2018.

Atılım Güneş Baydin, Robert Cornish, David Martínez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *ICLR*, 2018.

Harkirat Singh Behl, Michael Sapienza, Gurkirt Singh, Suman Saha, Fabio Cuzzolin, and Philip Torr. Incremental tube construction for human action detection. In *BMVC*, 2018.

Harkirat Singh Behl, Atılım Güneş Baydin, and Philip H.S. Torr. Alpha maml: Adaptive model-agnostic meta-learning. In *ICML, AutoML Workshop*, 2019.

Harkirat Singh Behl, Atılım Güneş Baydin, Ran Gal, Philip Torr, and Vibhav Vineet. Autosimulate: (quickly) learning synthetic data generation. In *ECCV*, 2020a.

Harkirat Singh Behl, Mohammad Najafi, and Philip H. S. Torr. Meta learning deep visual words for fast video object segmentation. In *IROS*, 2020b.

Harkirat Singh Behl, M. Pawan Kumar, Philip H. S. Torr, and Krishnamurthy (Dj) Dvijotham. Overcoming the convex barrier for simplex inputs. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2021.

Y. Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8): 1889–1900, 2000. doi: 10.1162/089976600300015187.

Kristin P Bennett, Gautam Kunapuli, Jing Hu, and Jong-Shi Pang. Bilevel optimization and machine learning. In *WCCI*, 2008.

J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, 2013.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, 2011.

Leonard Berrada, Andrew Zisserman, and M Pawan Kumar. Deep Frank-Wolfe for neural network optimization. *International Conference on Learning Representations*, 2019a.

Leonard Berrada, Andrew Zisserman, and M Pawan Kumar. Training neural networks for and by interpolation. *arXiv preprint*, 2019b.

Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84, 2018.

Gregory Bonaert, Dimitar I. Dimitrov, Maximilian Baader, and Martin T. Vechev. Fast and precise certification of transformers. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *ICML*, 2019.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of ReLU-based neural networks via dependency analysis. *AAAI Conference on Artificial Intelligence*, 2020.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Lagrangian decomposition for neural network verification. *Conference on Uncertainty in Artificial Intelligence*, 2020a.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, P Kohli, P Torr, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020b.

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems 31*, 2018.

S. Caelles, K. Maninis, J. Pont, L. Leal-Taixé, D. Cremers, and L. Van Gool. One-shot video object segmentation. In *CVPR*, 2017.

Sergi Caelles, Alberto Montes, Kevis-Kokitsi Maninis, Yuhua Chen, Luc Van Gool, Federico Perazzi, and Jordi Pont-Tuset. The 2018 davis challenge on video object segmentation. *arXiv*, 2018.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy*, 2017.

Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan L. Yuille, and Yuyin Zhou. Transunet: Transformers make strong encoders for medical image segmentation. *CoRR*, 2021.

L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE PAMI*, 2018a.

Yuhua Chen, Jordi Pont, Alberto Montes, and Luc Van Gool. Blazingly fast video object segmentation with pixel-wise metric learning. In *CVPR*, 2018b.

Jingchun Cheng, Yi Tsai, Shengjin Wang, and Ming Yang. Segflow: Joint learning for video object segmentation and optical flow. In *ICCV*, 2017.

Jingchun Cheng, Yi-Hsuan Tsai, Wei-Chih Hung, Shengjin Wang, and Ming-Hsuan Yang. Fast and accurate online video object segmentation via tracking parts. In *CVPR*, 2018.

Hai Ci, Chunyu Wang, and Yizhou Wang. Video object segmentation by learning location-sensitive embeddings. In *ECCV*, 2018.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 2007.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016.

Emmanuel Gbenga Dada, Joseph Stephen Bassi, Haruna Chiroma, Shafi'i Muhammad Abdulhamid, Adebayo Olusola Adetunmbi, and Opeyemi Emmanuel Ajibuwa. Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, 2019.

Alessandro de Palma, Harkirat Behl, Rudy R Bunel, Philip Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. In *International Conference on Learning Representations*, 2021.

Luca Del Pero, Susanna Ricco, Rahul Sukthankar, and Vittorio Ferrari. Discovering the physical parts of an articulated object class from multiple videos. In *CVPR*, 2016.

J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. AdverTorch v0.1: An adversarial robustness toolbox based on pytorch. *arXiv preprint arXiv:1902.07623*, 2019.

Thanh-Toan Do, Anh Nguyen, and Ian Reid. Affordancenet: An end-to-end deep learning approach for object affordance detection. In *ICRA*, 2018.

Carl Doersch and Andrew Zisserman. Sim2real transfer learning for 3d human pose estimation: motion to the rescue. In *NeurIPS*, 2019.

Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *Conference on Uncertainty in Artificial Intelligence*, 2018.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. *Uncertainty in Artificial Intelligence*, 2019.

Nikita Dvornik, Julien Mairal, and Cordelia Schmid. Modeling visual context is key to augmenting object detection datasets. In *ECCV*, 2018.

Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *ICCV*, 2017.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *Automated Technology for Verification and Analysis*, 2017.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 2019.

Chelsea Finn and Sergey Levine. ICML tutorial: Meta-learning, 2019. `https://sites.google.com/view/icml19metalearning`.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017a.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017b.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017c.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, 2018.

A Gaidon, Q Wang, Y Cabon, and E Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016.

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *ICML*, 2018.

Iliyan Georgiev, Thiago Ize, Mike Farnsworth, Ramon Montoya-Vozmediano, Alan King, Brecht Van Lommel, Angel Jimenez, Oscar Anson, Shinji Ogaki, Eric Johnston, et al. Arnold: A brute-force production path tracer. *TOG*, 2018.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018a.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *SECML NeurIPS*, 2018b.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding real world indoor scenes with synthetic data. In *CVPR*, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *CoRR*, 2019.

João F Henriques, Sebastien Ehrhardt, Samuel Albanie, and Andrea Vedaldi. Small steps and giant leaps: Minimal newton solvers for deep learning. In *ICCV*, 2019.

Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. In *arXiv*, 2017.

Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. On pre-trained image features and synthetic images for deep learning. In *ECCVW*, 2018.

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 2012.

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001a.

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *ICANN*, 2001b.

Tomas Hodan, Frank Michel, Eric Brachmann, Wadim Kehl, Anders Glent Buch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, et al. BOP: Benchmark for 6D object pose estimation. In *ECCV*, 2018.

Tomáš Hodaň, Vibhav Vineet, Ran Gal, Emanuel Shalev, Jon Hanzelka, Treb Connell, Pedro Urbina, Sudipta Sinha, and Brian Guenter. Photorealistic image synthesis for object instance detection. *ICIP*, 2019.

Yuan-Ting Hu, Jia-Bin Huang, and Alexander Schwing. Maskrnn: Instance level video object segmentation. In *NIPS*, 2017.

Yuan-Ting Hu, Jia-Bin Huang, and Alexander G. Schwing. Videomatch: Matching based video object segmentation. In *ECCV*, 2018.

Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. In *EMNLP*, 2019.

F. Hutter, H. Hoos, and K. Leyton-Brown. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 1209–1216. ACM, 2013.

Nicola Landro Ignazio Gallo, Gianmarco Ria and Riccardo La Grassa. Image and text fusion for upmc food-101 using bert and cnns. In *International Conference on Image and Vision Computing New Zealand (IVCNZ 2020)*, 2020.

Suyog Dutt Jain and Kristen Grauman. Supervoxel-consistent foreground propagation in video. In *ECCV*, 2014.

Varun Jampani, Raghudeep Gadde, and Peter V. Gehler. Video propagation networks. In *CVPR*, 2017.

Won-Dong Jang and Chang-Su Kim. Online video object segmentation via convolutional trident network. In *CVPR*, 2017.

R. G. Jeroslow. Representability in mixed integer programming, i: Characterization results. *Discrete Applied Mathematics*, 1987.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2019.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *ICCV*, 2019.

Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *International Conference on Computer-Aided Verification*, 2017.

Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: making rgb-based 3d detection and 6d pose estimation great again. In *ICCV*, 2017.

Jacqueline Kenney, Thomas Buckley, and Oliver Brock. Interactive segmentation for manipulation in unstructured environments. In *ICRA*, 2009.

Anna Khoreva, Rodrigo Benenson, Eddy Ilg, Thomas Brox, and Bernt Schiele. Lucid data dreaming for object tracking. In *CVPR Workshops*, 2017.

Douwe Kiela, Suvrat Bhooshan, Hamed Firooz, and Davide Testuggine. Supervised multimodal bitransformers for classifying images and text. *arXiv preprint arXiv:1909.02950*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *International Conference on Machine Learning, Deep Learning Workshop*, 2015.

Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *NeurIPS*, 2011.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2011.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

Tuan Anh Le, Atılım Güneş Baydin, Robert Zinkov, and Frank Wood. Using synthetic data to train neural networks is model-based reasoning. In *IJCNN*, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Claude Lemaréchal. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer, 2001.

Fuxin Li, Taeyoung Kim, Ahmad Humayun, David Tsai, and James Rehg. Video segmentation by tracking many figure-ground segments. In *ICCV*, 2013.

Siyang Li, Bryan Seybold, Alexey Vorobyov, Alireza Fathi, Qin Huang, and C.-C. Jay Kuo. Instance embedding transfer to unsupervised video object segmentation. In *CVPR*, 2018.

Xiaoxiao Li and Chen Change Loy. Video object segmentation with joint re-identification and attention-aware mask propagation. In *ECCV*, 2018.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few shot learning. *CoRR*, 2017.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *AISTATS*, 2020.

Gilles Louppe and Kyle Cranmer. Adversarial variational optimization of non-differentiable simulators. In *AISTATS*, 2019.

Jingyue Lu and M Pawan Kumar. Neural network branching for neural network verification. In *International Conference on Learning Representations*, 2020.

J Luiten, P Voigtlaender, and B Leibe. Premvos: Proposal-generation, refinement and merging for the davis challenge on video object segmentation 2018. In *CVPR Workshops*, 2018.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, 2015.

Laurens Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 2008.

Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *ICLR*, 2019.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, 2015.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations*, 2018.

Nicolas Maerki, Federico Perazzi, Oliver Wang, and Alexander Sorkine-Hornung. Bilateral space video segmentation. In *CVPR*, 2016.

K.-K. Maninis, S. Caelles, Y. Chen, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool. Video object segmentation without temporal information. *IEEE PAMI*, 2018.

James Martens. Deep learning via hessian-free optimization. In *ICML*, 2010.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, 2015.

Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Comput. Surv.*, 2021.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *ECCV*, 2020.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. *International Conference on Machine Learning*, 2018.

Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *Conference on Computer Vision and Pattern Recognition*, 2016.

Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*. Springer Berlin Heidelberg, 1978.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Devang K Naik and RJ Mammone. Meta-neural networks that learn by learning. In *IJCNN*, 1992.

M. Najafi, V. Kulharia, T. Ajanthan, and P. H. S. Torr. Similarity learning for dense label transfer. *CVPR Workshops*, 2018.

Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, 2019.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net: learning local features from images. In *NIPS*, 2018.

Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *CoRR*, 2021.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *NIPS Autodiff Workshop*, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 2011.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *CVPR*, 2016.

F. Perazzi, A. Khoreva, R. Benenson, B. Schiele, and A.Sorkine-Hornung. Learning video object segmentation from static images. In *CVPR*, 2017.

Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv*, 2017.

Alessandro Prest, Christian Leistner, Javier Civera, Cordelia Schmid, and Vittorio Ferrari. Learning object class detectors from weakly annotated video. In *CVPR*, 2012.

Mahdi Rad and Vincent Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *ICCV*, 2017.

Prabhakar Raghavan. How AI is powering a more helpful Google. `https://www.blog.google/products/search/search-on/`, 2020.

Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *Neural Information Processing Systems*, 2018.

Aravind Rajeswaran, Chelsea Finn, Sham M. Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *NeurIPS*, 2019.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *Fifth International Conference on Learning Representations (ICLR)*, 2017.

Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *CVPR*, 2017.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *PAMI*, 2017.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *ECCV*, 2016.

Stephan R. Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *ICCV*, 2017.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

Germán Ros, Laura Sellart, Joanna Materzynska, David Vázquez, and Antonio M. López. The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. Learning to simulate. In *ICLR*, 2019.

Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Semantic foggy scene understanding with synthetic data. *IJCV*, 2018.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *Neural Information Processing Systems*, 2019.

Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *CVPR*, 2019.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987a.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-...* PhD thesis, Technische Universitat Munchen, Germany, 1987b.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *NIPS*, 2015.

Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *PAMI*, 2017.

Hanif D. Sherali and Gyunghyun Choi. Recovery of primal solutions when using subgradient optimization methods to solve lagrangian duals of linear programs. *Operations Research Letters*, 1996.

Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, USA, 1994.

Jae Shin Yoon, Francois Rameau, Junsik Kim, Seokju Lee, Seunghak Shin, and In So Kweon. Pixel-level matching for video object segmentation using convolutional neural networks. In *ICCV*, 2017.

Mennatullah Siam, Chen Jiang, Steven Lu, Laura Petrich, Mahmoud Gamal, Mohamed Elhoseiny, and Martin Jagersand. Video object segmentation using teacher-student adaptation in a human robot interaction (hri) setting. In *ICRA*, 2019.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Neural Information Processing Systems*, 2018.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. *Neural Information Processing Systems*, 2019a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 2019b.

Gagandeep Singh, Jonathan Maurer, Christoph Müller, Matthew Mirman, Timon Gehr, Adrian Hoffmann, Petar Tsankov, Dana Drachsler Cohen, Markus Püschel, and Martin Vechev. ETH robustness analyzer for neural networks (ERAN), 2020.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NIPS*, 2017a.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NIPS*, 2017b.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

Hao Su, Charles Ruizhongtai Qi, Yangyan Li, and Leonidas J. Guibas. Render for CNN: viewpoint estimation in images using cnns trained with rendered 3d model views. In *ICCV*, 2015.

Yuxiang Sun, Ming Liu, and Max Q-H Meng. Motion removal for reliable rgb-d slam in dynamic environments. *RAS*, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2014.

Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. In *CVPR*, 2018.

Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *arXiv preprint arXiv:2006.14076*, 2020.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *International Conference on Learning Representations*, 2019.

M.J. Todd. *The Computation of Fixed Points and Applications*. Lecture Notes in Mathematics; 513. Springer-Verlag, 1976.

Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Jonathan Tremblay, Thang To, and Stan Birchfield. Falling things: A synthetic dataset for 3d object detection and pose estimation. In *CVPR*, 2018.

Yi-Hsuan Tsai, Ming-Hsuan Yang, and Michael J. Black. Video segmentation via object flow. In *CVPR*, 2016.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, 2016.

Gul Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from synthetic humans. In *CVPR*, 2017.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, 2016a.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, 2016b.

VNN-COMP. International verification of neural networks competition (VNN-COMP). *Verification of Neural Networks workshop at the International Conference on Computer-Aided Verification*, 2020.

Paul Voigtlaender and Bastian Leibe. Online adaptation of convolutional neural networks for video object segmentation. In *BMVC*, 2017.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Neural Information Processing Systems*, 2018.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *NeurIPS*, 2021.

Xin Wang, Devinder Kumar, Nicolas Thome, Matthieu Cord, and Frédéric Precioso. Recipe recognition with large multimodal food dataset. In *2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, 2015.

Stefan Webb, Tom Rainforth, Yee Whye Teh, and M Pawan Kumar. A statistical approach to assessing neural network robustness. *International Conference on Learning Representations*, 2019.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *International Conference on Machine Learning*, 2018.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *International Conference on Machine Learning*, 2018.

Seoung Wug Oh, Joon Lee, Kalyan Sunkavalli, and Seon Joo Kim. Fast video object segmentation by reference-guided mask propagation. In *CVPR*, 2018.

Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *RSS*, 2018.

Huaxin Xiao, Jiashi Feng, Guosheng Lin, Yu Liu, and Maojun Zhang. Monet: Deep motion exploitation for video object segmentation. In *CVPR*, 2018.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33, 2020.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers, 2021.

Dawei Yang and Jia Deng. Learning to generate synthetic 3d training data through hybrid gradient. In *CVPR*, 2020.

Linjie Yang, Yanran Wang, Xuehan Xiong, Jianchao Yang, and Aggelos K. Katsaggelos. Efficient video object segmentation via network modulation. In *CVPR*, 2018.

Dingwen Zhang, Le Yang, Deyu Meng, Dong Xu, and Junwei Han. Spftn: A self-paced fine-tuning network for segmenting objects in weakly labelled videos. In *CVPR*, 2017a.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Neural Information Processing Systems*, 2018.

Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas A. Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *CVPR*, 2017b.