# Efficient Neural Network Verification and Training


**Alessandro De Palma**
Somerville College
University of Oxford

A thesis submitted for the degree of
*Doctor of Philosophy*
Hilary 2023

# Acknowledgements

In the first place, I want to thank my main supervisor, Pawan, for all the professional and personal advice in the last four years. Or perhaps I should say five, given his support from when I was only a potential student. Thank you, Pawan, for having balanced my sincere pessimism with unwavering optimism and enthusiasm on a weekly basis: I can assure this is non-trivial. I would also like to thank Phil who, on top of having been a co-author on my verification work, kindly acted as a joint supervisor in the final phases of my PhD. My sincere thanks go to my colleagues in the OVAL group: Rudy, Alban, Leo, Prateek, Jodie, Florian, Alasdair, Harkirat, Francisco. I really enjoyed our discussions in group meetings and reading groups, from which I learned a lot. Special thanks go to Leo, with whom I also shared unlikely nights out, and to Rudy, who mentored all my verification work. I was very happy to meet Pawan, Leo and Rudy again in a different role during my internship at DeepMind.

Complaining is undeniably a national sport, and I would also argue it is one of my greatest talents. With this statement, I would like to thank all the friends who put up with me in the course of my doctoral studies. The cybernetic anger management group: Panagiotis, with whom I shared a cosy Oxford flat and conversations on molotov-throwing Greek monks, and Vitaly, with whom I had the great luck to combine friendship and co-authorship. Thank you to the other people who supported me in Oxford: Ada, Bernardo, Kyriakos, Luigi, Nikitas; and remotely: Luca, Federica, Emanuele, Giuseppe. I also seize the occasion to thank everyone who supported me during my university years in Turin, Zurich, and Boston.

As obvious as it sounds, completing a PhD during the COVID-19 pandemic was a rather challenging experience. In spite of the isolation resulting from remote work, I am glad I exploited the resulting increase in mobility to move closer to the people I love, making long stays across Italy and France. In this regard, my most sincere thanks go to Ornella and my parents, who have been by my side throughout my PhD.

Finally, I would like to thank EPSRC and IBM Research for funding my studies.

Grazie.

# Abstract

In spite of their highly-publicized achievements in disparate applications, neural networks are yet to be widely deployed in safety-critical applications. In fact, fundamental concerns exist on the robustness, fairness, privacy and explainability of deep learning systems. In this thesis, we strive to increase trust in deep learning systems by presenting contributions pertaining to neural network verification and training. First, by designing dual solvers for popular network relaxations, we provide fast and scalable bounds on neural network outputs. In particular, we present two solvers for the convex hull of element-wise activation functions, and two algorithms for a formulation based on the convex hull of the composition of ReLU activations with the preceding linear layer. We show that these methods are significantly faster than off-the-shelf solvers, and improve on the speed-accuracy trade-offs of previous dual algorithms. In order to efficiently employ them for formal neural network verification, we design a massively parallel Branch-and-Bound framework around the bounding algorithms. Our contributions, which we publicly released as part of the OVAL verification framework, improved on the scalability of existing network verifiers, and proved to be influential for the development of more recent algorithms. Second, we present an intuitive and inexpensive algorithm to train neural networks for verifiability via Branch-and-Bound. Our method is shown to yield state-of-the-art performance on verifying robustness to small adversarial perturbations while reducing the training costs compared to previous algorithms. Finally, we conduct a comprehensive experimental evaluation of specialized training schemes to train networks for multiple tasks at once, showing that they perform on par with a simple baseline. We provide a partial explanation of our surprising results, aiming to stir further research towards the understanding of deep multi-task learning.

# Contents

**List of Figures** viii

**List of Tables** xii

**List of Abbreviations** xiv

**Notation** xv

**1 Introduction and Related Work** 1
- 1.1 Preamble: Trustworthy Deep Learning . . . 2
- 1.2 Neural Network Verification . . . 3
  - 1.2.1 Neural Network Bounds . . . 3
  - 1.2.2 Verification via Branch-and-Bound . . . 4
- 1.3 Neural Network Training . . . 6
  - 1.3.1 Training for Verifiability . . . 6
  - 1.3.2 Deep Multi-Task Learning . . . 7
- 1.4 Thesis Outline and Contributions . . . 8
  - 1.4.1 Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition . . . 9
  - 1.4.2 Scaling the Convex Barrier with Sparse Dual Algorithms . . . 9
  - 1.4.3 IBP Regularization for Verified Adversarial Robustness via Branch-and-Bound . . . 10
  - 1.4.4 In Defense of the Unitary Scalarization for Deep Multi-Task Learning . . . 11

**2 Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition** 12
- 2.1 Introduction . . . 15
- 2.2 Neural Network Verification . . . 18
  - 2.2.1 Problem Specification . . . 18
  - 2.2.2 Neural Network Bounding . . . 19
  - 2.2.3 Branch and Bound . . . 22
- 2.3 Better Bounding: Lagrangian Decomposition . . . 24

2.3.1 Problem Derivation . . . 24
2.3.2 Supergradient Solver . . . 26
2.3.3 Proximal Solver . . . 28
2.3.4 Comparison to Previous Dual Problems . . . 31
2.4 Better Branching . . . 33
2.4.1 Preliminaries: Approximations of Strong Branching . . . 33
2.4.2 Filtered Smart Branching . . . 35
2.5 Improved Branch and Bound . . . 37
2.5.1 Additional Branch and Bound Improvements . . . 37
2.5.2 Implementation Details . . . 39
2.6 Related Work . . . 41
2.7 Incomplete Verification Experiments . . . 42
2.7.1 Experimental Setup . . . 42
2.7.2 Bounding Algorithms . . . 43
2.7.3 Results . . . 45
2.8 Complete Verification . . . 47
2.8.1 Experimental Setup . . . 47
2.8.2 Bounding Algorithms . . . 49
2.8.3 Branching . . . 50
2.8.4 Intermediate Bounds . . . 53
2.8.5 Comparison of Complete Verifiers . . . 54
2.9 Discussion . . . 58

**3 Scaling the Convex Barrier with Sparse Dual Algorithms 59**
3.1 Introduction . . . 61
3.2 Preliminaries: Neural Network Relaxations . . . 63
3.2.1 Planet Relaxation . . . 64
3.2.2 A Tighter Relaxation . . . 65
3.3 A Dual Formulation for the Tighter Relaxation . . . 67
3.4 Active Set . . . 69
3.4.1 Solver . . . 69
3.4.2 Extending the Active Set . . . 71
3.5 Saddle Point . . . 74
3.5.1 Sparsity via Sufficient Statistics . . . 74
3.5.2 Solver . . . 77
3.6 Implementation Details, Technical Challenges . . . 80
3.6.1 Parallelism, masked forward/backward passes . . . 81
3.6.2 Stratified bounding for branch and bound . . . 81
3.7 Related Work . . . 83
3.8 Experiments . . . 84

3.8.1 Experimental Setting . . . 85
3.8.2 Incomplete Verification . . . 86
3.8.3 Branch and Bound . . . 91
3.9 Discussion . . . 98

**4 IBP Regularization for Verified Adversarial Robustness via Branch-and-Bound 99**
4.1 Introduction . . . 101
4.2 Background . . . 102
4.2.1 Neural Network Verification . . . 103
4.2.2 Training via the Robust Loss . . . 104
4.2.3 Hybrid Training Methods . . . 105
4.3 Training via IBP Regularization . . . 106
4.4 Verification Framework . . . 108
4.4.1 Branch–and-Bound Setup . . . 108
4.4.2 UPB Branching . . . 108
4.5 Related Work . . . 109
4.6 Experiments . . . 111
4.6.1 Verified Training . . . 111
4.6.2 Branching . . . 112
4.7 Conclusions . . . 113

**5 In Defense of the Unitary Scalarization for Deep Multi-Task Learning 115**
5.1 Introduction . . . 117
5.2 Related Work . . . 118
5.3 Multi-Task Learning Optimizers . . . 120
5.4 Experimental Evaluation . . . 124
5.4.1 Supervised Learning . . . 125
5.4.2 Reinforcement Learning . . . 129
5.5 Regularization in Specialized MTL Optimizers . . . 131
5.5.1 Ablation Study . . . 131
5.5.2 Technical Results . . . 132
5.5.3 Under-Optimization: Empirical Study . . . 136
5.6 Conclusions . . . 137

**6 Conclusions 138**
6.1 Summary . . . 139
6.2 Discussion and Future Work . . . 140
6.2.1 Neural Network Verification . . . 140
6.2.2 Training for Verifiability . . . 141
6.2.3 Deep Multi-Task Learning . . . 142

**Appendices**

**A Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition 144**
A.1 Proof of theorem 1 . . . 145
A.2 Proof of proposition 2 . . . 149
A.3 Sigmoid Activation function . . . 152
A.3.1 Convex hull computation . . . 153
A.3.2 Solving the $\mathcal{P}_k$ subproblems over sigmoid activation . . . 156
A.4 Momentum for the Proximal solver . . . 157
A.5 Supplementary Incomplete Verification Experiments . . . 159

**B Scaling the Convex Barrier with Sparse Dual Algorithms 161**
B.1 Limitations of Previous Dual Approaches . . . 162
B.2 Dual Initialisation . . . 163
B.2.1 Equivalence to Planet . . . 163
B.2.2 Big-M Dual . . . 164
B.2.3 Big-M solver . . . 165
B.3 Implementation Details for Saddle Point . . . 166
B.3.1 Constraint Price Caps . . . 166
B.3.2 Primal Initialization . . . 167
B.4 Dual Derivations . . . 168
B.5 Intermediate Bounds . . . 169
B.6 Pre-activation Bounds in $\mathcal{A}_k$ . . . 170
B.6.1 Motivating Example . . . 171
B.6.2 Derivation of $\mathcal{A}_k$ . . . 174
B.7 Masked Forward and Backward Passes . . . 177
B.7.1 Convolution as matrix-matrix multiplication . . . 178
B.7.2 Masked convolution as matrix-matrix multiplication . . . 178
B.8 Experimental Appendix . . . 179
B.8.1 Adversarially-Trained CIFAR-10 Incomplete Verification . . . 179
B.8.2 Sensitivity of Active Set to selection criterion and frequency . . . 180
B.8.3 MNIST Incomplete Verification . . . 183

**C IBP Regularization for Verified Adversarial Robustness via Branch-and-Bound 185**
C.1 Comparison between IBP-R and COLT . . . 186
C.2 Complete Verification Problem . . . 187
C.3 Intermediate Bounds . . . 187
C.4 Interval Bound Propagation . . . 188
C.5 UPB Branching: Beta-CROWN Dual and Planet relaxation . . . 189

C.5.1 Planet Relaxation . . . . . 189
C.5.2 Beta-CROWN Dual . . . . . 190
C.5.3 UPB Branching . . . . . 191
C.6 Experimental Details . . . . . 191
C.6.1 Experimental Setting, Hyper-parameters . . . . . 192
C.6.2 Jax Porting of COLT . . . . . 193
C.7 Supplementary Branching Experiments . . . . . 194

**D In Defense of the Unitary Scalarization for Deep Multi-Task Learning 195**
D.1 Societal Impact . . . . . 196
D.2 Supplement to the Overview of Multi-Task Optimizers . . . . . 196
D.2.1 MGDA . . . . . 196
D.2.2 IMTL . . . . . 198
D.2.3 PCGrad . . . . . 200
D.2.4 GradDrop . . . . . 202
D.3 Experimental Setting, Reproducibility . . . . . 204
D.3.1 Supervised Learning . . . . . 204
D.3.2 Reinforcement Learning . . . . . 205
D.3.3 Software Acknowledgments and Licenses . . . . . 207
D.4 Supplementary Supervised Learning Experiments . . . . . 208
D.4.1 Addendum . . . . . 208
D.4.2 Unregularized Experiments . . . . . 208
D.4.3 Sign-Agnostic GradDrop . . . . . 211
D.5 Supplementary Reinforcement Learning Experiments . . . . . 211
D.5.1 Addendum . . . . . 212
D.5.2 Ablation studies . . . . . 213
D.5.3 Sensitivity to Reward Normalization . . . . . 213

**Bibliography 216**

# List of Figures

2.1 Feasible domain of the convex hull for an ambiguous ReLU. Red circles indicate the vertices of the feasible region. . . . . 23
2.2 Distribution of runtime and gap to optimality on a network adversarially trained with the method by Madry et al. (2018), on CIFAR-10 images. In both cases, lower is better. The width at a given value represents the proportion of problems for which this is the result. Gurobi Planet always returns the optimal solution to problem (2.7), at a large computational cost. 44
2.3 Pointwise comparison for a subset of the methods on the data presented in Figure 2.2. Each datapoint corresponds to a CIFAR image, darker color shades mean higher point density in a logarithmic scale. The dotted line corresponds to the equality and in both graphs, lower is better along both axes. . . . . 46
2.4 Complete verification performance of different bounding algorithms within BaBSR, on the OVAL dataset. . . . . 51
2.5 Using Proximal as bounding algorithm, complete verification performance of different branching strategies, on the OVAL dataset. . . . . 52
2.6 Using Proximal for the last layer bounding, complete verification performance of different intermediate bounding schemes, on the OVAL dataset. . . . . 54
2.7 Performance of different complete verification algorithms, on the OVAL dataset. . . . . 56
2.8 Performance of different complete verifiers on adversarial robustness verification of larger, COLT-trained convolutional, networks (Balunovic and Vechev, 2020). . . . . 57

3.1 Upper bounds to the adversarial vulnerability for the SGD-trained network from Bunel et al. (2020a). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better, the width at a given value represents the proportion of problems for which this is the result. On average, both Active Set and Saddle Point achieve bounds tighter than Gurobi 1 cut with a smaller runtime. . . . . 88

3.2 Pointwise comparison for a subset of the methods on the data presented in figure 3.1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality. . . . . 89
3.3 Pointwise comparison between our proposed solvers on the data presented in figure 3.1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality. . . . . 90
3.4 Upper bounds to the adversarial vulnerability of a fully connected network with two hidden layers of width 7000. Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. When run for enough iterations, Saddle Point achieves bounds tighter than both Gurobi 1 cut and Active Set, whose tightness is constrained by memory, in less time. . 91
3.5 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. . . . . 94
3.6 Cactus plots on properties from Lu and Kumar (2020), displaying the percentage of solved properties as a function of runtime. Comparison of best performing method from figure 3.5 with tighter bounding schemes. . 96

4.1 Convex hull for an ambiguous ReLU (Ehlers, 2017). . . . . 104
4.2 Complete verification performance of different branching strategies on two IBP-R-trained CIFAR-10 networks from §4.6.1. . . . . 113

5.1 No algorithm outperforms unitary scalarization on the Multi-MNIST dataset. 125
5.2 While Specialized Multi-Task Optimizers (SMTOs) display larger runtimes, none of them outperforms the unitary scalarization on the CelebA dataset. 126
5.3 On Cityscapes, none of the SMTOs outperforms unitary scalarization, which proves to be the most cost-effective algorithm. Subfigures (a)-(d) report means for three runs, and their 95% CIs. . . . . 128
5.4 On Metaworld, none of the SMTOs significantly outperforms Unit. Scal., which is the least expensive method. Subfigures (a)-(b) report mean and 95% CI for the best (over the updates) average success rate. Subfigures (c)-(d) show box plots for the training time of 10,000 updates. . . . . 130
5.5 Mean and 95% CI (3 runs) avg. task validation accuracy over epochs on CelebA. SMTOs postpone the onset of overfitting, mirroring the effect of $\ell_2$ regularization on unitary scalarization. . . . . 132
5.6 Mean and 95% CI (3 runs) for $\left\|\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i\right\|_2$ on CelebA. Multiple-Gradient Descent Algorithm (MGDA) and Impartial Multi-Task Learning (IMTL) converge away from stationary points of unitary scalarization, indicating under-optimization. . . . . 132

A.1 Convex hull of the Sigmoid activation function for different input bound configurations. . . . . 153
A.2 Comparison of the distribution of runtime and gap to optimality on an SGD-trained network. In both cases, lower is better. The width at a given value represents the proportion of problems for which this is the result. Gurobi Planet always returns the optimal solution to problem (2.7), at a large computational cost. . . . . 159
A.3 Pointwise comparison for a subset of the methods on the data presented in Figure A.2. Each datapoint corresponds to a CIFAR image, darker colour shades mean higher point density in a logarithmic scale. The dotted line corresponds to the equality and in both graphs, lower is better along both axes. . . . . 160

B.1 $\mathcal{M}_k$ plotted on the $(\mathbf{z}_k, \mathbf{x}_k)$ plane, when $\hat{\mathbf{l}}_k \leq 0$ and $\hat{\mathbf{u}}_k \geq 0$. . . . . 164
B.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). 172
B.3 Upper bounds to the adversarial vulnerability for the network adversarially trained with the method by Madry et al. (2018), from Bunel et al. (2020a). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better, the width at a given value represents the proportion of problems for which this is the result. . . . . 180
B.4 Pointwise comparison for a subset of the methods on the data presented in figure 3.1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality. . . . . 181
B.5 Pointwise comparison between our proposed solvers on the data presented in figure B.3. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality. . . . . 181
B.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). Sensitivity of Active Set to selection criterion (see §3.4.2). . . . . 182
B.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. Results for the SGD-trained network from Bunel et al. (2020a). Sensitivity of Active Set to variable addition frequency $\omega$, with the selection criterion presented in §3.4.2. . . . . 182

B.8 Upper bounds to the adversarial vulnerability for the MNIST network trained with the verified training algorithm by Wong and Kolter (2018), from Lu and Kumar (2020). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. . . . . 183
B.9 Pointwise comparison for a subset of the methods on the data presented in Figure B.8. 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. . . . . 184

C.1 Specifications of the employed network architectures for the experiments of §4.6. 191
C.2 Depiction of the Interval Bound Propagation (IBP) hyper-rectangle. . . . 191
C.3 IBP-R hyper-parameters for the experiment of table 4.1. . . . . 193
C.4 Complete verification performance of different branching strategies, on two COLT-trained CIFAR-10 networks from §4.6.1. . . . . 194

D.1 Additional figures for the comparison of various SMTOs with the unitary scalarization on the MultiMNIST dataset (Sener and Koltun, 2018). . . . 209
D.2 Additional figures for the comparison of various SMTOs with the unitary scalarization on the CelebA (Liu et al., 2015) dataset. . . . . 209
D.3 Additional figures for the unregularized comparison of various SMTOs with the unitary scalarization on CelebA. SMTOs provide varying degrees of regularization. . . . . 210
D.4 Effect of regularization (dropout layers and weight decay) on the average task validation accuracy for all considered optimizers on the CelebA dataset: regularization improves the average performance of all algorithms. . . . . 210
D.5 Effect of regularization (dropout layers and weight decay) on unitary scalarization on the CelebA dataset: violin plots (20 runs) for the best avg. task validation accuracy over epochs. The width at a given value represents the proportion of runs yielding that result. Regularization improves the average performance while decreasing its variability. . . . . 211
D.6 Comparison of GradDrop (Chen et al., 2020) with sign-agnostic masking of the shared-representation gradients on the CelebA dataset (Liu et al., 2015). No statistically relevant difference between the two methods can be observed for the majority of the epochs. . . . . 212
D.7 Mean and 95% CI for the best avg. success rate on Metaworld. None of the SMTOs significantly outperforms unitary scalarization. . . . . 212
D.8 Mean and 95% CI for the avg. success rate on Metaworld. None of the SMTOs significantly outperforms unitary scalarization. . . . . 213

D.9 For both MT10 and MT50, actor $l_2$ regularization pushes the average higher for unitary scalarization. . . . . 213
D.10 Additional figures for the comparison of SMTOs with the unitary scalarization on the Cityscapes (Cordts et al., 2016) dataset. . . . . 214
D.11 The learning outcomes of a Multitask SAC agent vary considerably depending on the reward normalisation hyperparameter. Each of the curves represents and average of 10 runs with shaded 95% confidence interval. . . 215
D.12 Metaworld's MT10 ablation experiments. . . . . 215

# List of Tables

2.1 Specifications of the OVAL dataset, a subset of the CIFAR-10 complete verification dataset from Lu and Kumar (2020). Each property is associated to a different $\epsilon_{\text{ver}}$ value. WK denotes the verified training algorithm by Wong and Kolter (2018). . . . . 47
2.2 Specifications of the COLT-based (Balunovic and Vechev, 2020) CIFAR-10 complete verification dataset. . . . . 48

3.1 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. . . . . 92
3.2 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. . . . . 94
3.3 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 result is highlighted in bold. Comparison of best performing method from figure 3.5 with tighter bounding schemes within BaBSR. 96
3.4 Number of verified properties and average runtime on the adversarially trained (Madry et al., 2018) `ConvSmall` network from the ERAN (Singh et al., 2020) dataset, on the first 1000 images of the CIFAR-10 test set. Results for FastC2V and OptC2V are taken from Tjandraatmadja et al. (2020), results for kPoly are taken from Singh et al. (2019a). . . . . 97

4.1 Performance of different verified training algorithms under $\ell_\infty$ norm perturbations on the CIFAR10 dataset. The table reports mean and sample standard deviation over 3 repetitions for our experiments, over 5 repetitions for (Shi et al., 2021) on $\epsilon_{\text{ver}} = 8/255$. The remaining results from the literature were executed with a single seed. The method with the best average performance for each perturbation radius is highlighted in bold. . 111

C.1 Main differences between COLT (§4.2.3) and IBP-R (§4.3). . . . . 186

D.1 Hyperparameters of the reinforcement learning (RL) experiments. Hyperparameters different from Sodhani et al. (2021) are in bold. . . . . 206

# List of Abbreviations

**BaB** . . . . Branch and Bound.

**CI** . . . . . Confidence Interval.

**COLT** . . . Convex Layer-wise Adversarial Training.

**IBP** . . . . Interval Bound Propagation.

**IMTL** . . . Impartial Multi-Task Learning.

**LP** . . . . . Linear Program.

**NN** . . . . . Neural Network.

**MDP** . . . . Markov Decision Process.

**MGDA** . . . Multiple-Gradient Descent Algorithm.

**MIP** . . . . Mixed Integer Program.

**MLP** . . . . Multi-Layer Perceptron.

**MTL** . . . . Multi-Task Learning.

**PGD** . . . . Projected Gradient Descent.

**RL** . . . . . Reinforcement Learning.

**SAC** . . . . Soft Actor-Critic.

**SDP** . . . . Semi-Definite Program.

**SGD** . . . . Stochastic Gradient Descent.

**SMT** . . . . Satisfiability Modulo Theory.

**SMTO** . . . Specialized Multi-Task Optimizer.

**SP-FW** . . . Saddle Point Frank Wolfe.

# Notation

| | |
|---|---|
| **Bold lower case** . . . | Vectors (e.g., $\mathbf{x}$). |
| **Upper case** . . . . . | Matrices (e.g., $W$). |
| $[\cdot, \cdot]$ . . . . . | Intervals (e.g., $[\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k]$). |
| $[\cdot]$ . . . . . | Vector or matrix entries (e.g., $\mathbf{x}[i]$ or $W[i, j]$). |
| $\odot$ . . . . . | Hadamard product. |
| $[\![\cdot, \cdot]\!]$ . . . . . | Integer ranges. |
| $\mathbb{1}_{\mathbf{a}}$ . . . . . | Indicator vector on condition $\mathbf{a}$. |
| $\mathbf{Conv}(f, \mathbf{a}, \mathbf{b})$ . . . . | Convex hull of function $f$ defined in $[\mathbf{a}, \mathbf{b}]$. |
| $\mathbf{Conv}(S)$ . . . . . | Convex hull of a set $S$. |
| $\mathbf{Aff}(S)$ . . . . . | Affine hull of a set $S$. |
| $\mathbf{col}_i(W)$ . . . . . | $i$-th column of matrix $W$. |
| $\mathbf{row}_i(W)$ . . . . . | $i$-th row of matrix $W$. |
| $W \diamond \mathbf{x}$ . . . . . | Shorthand for $\sum_i \mathrm{col}_i(W) \odot \mathbf{x}$. |
| $W \square \mathbf{x}$ . . . . . | Shorthand for $\sum_i \mathrm{col}_i(W)^T \mathbf{x}$. |
| $[\mathbf{x}]_+$ . . . . . | Positive part of a vector (i.e., $[\mathbf{x}]_+ := \max(\mathbf{x}, \mathbf{0})$). |
| $[\mathbf{x}]_-$ . . . . . | Negative part of a vector (i.e., $[\mathbf{x}]_- := \min(\mathbf{x}, \mathbf{0})$). |
| $\cos(\mathbf{x}, \mathbf{z})$ . . . . . | Cosine similarity between vectors $\mathbf{x}$ and $\mathbf{z}$. |

# 1 Introduction and Related Work

## Contents


1.1 Preamble: Trustworthy Deep Learning . . . . . **2**
1.2 Neural Network Verification . . . . . **3**
1.2.1 Neural Network Bounds . . . . . 3
1.2.2 Verification via Branch-and-Bound . . . . . 4
1.3 Neural Network Training . . . . . **6**
1.3.1 Training for Verifiability . . . . . 6
1.3.2 Deep Multi-Task Learning . . . . . 7
1.4 Thesis Outline and Contributions . . . . . **8**
1.4.1 Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition . . . . . 9
1.4.2 Scaling the Convex Barrier with Sparse Dual Algorithms 9
1.4.3 IBP Regularization for Verified Adversarial Robustness via Branch-and-Bound . . . . . 10
1.4.4 In Defense of the Unitary Scalarization for Deep Multi-Task Learning . . . . . 11

## 1.1 Preamble: Trustworthy Deep Learning

In the last few years, neural networks have achieved remarkable performance in a variety of high-profile applications, from protein folding (Senior et al., 2020; Jumper et al., 2021) to fast matrix multiplication (Fawzi et al., 2022). Owing to the publicity surrounding these achievements, neural-networks-based systems now regularly appear in mainstream information outlets (Geddes, 2022; Larousserie, 2022; Iannaccone, 2022), leading to an always-increasing media exposure. As a direct consequence, there has been a surge in interest around trustworthiness of machine learning algorithms (Varshney, 2022). In particular, efforts have been directed towards ensuring the fairness (Du et al., 2020), explainability (Angelov and Soares, 2020), robustness (Carlini and Wagner, 2017), and privacy (Abadi et al., 2016) of neural networks. Progress in deep learning has advanced at a particularly fast pace in computer vision (Krizhevsky et al., 2012; Voulodimos et al., 2018), where it has long achieved super-human performance on standardized tasks (O'Mahony et al., 2019). However, the discovery of adversarial examples (Szegedy et al., 2014; Goodfellow et al., 2015), human-imperceptible perturbations that can dramatically alter network predictions, has cast serious doubts on the foundations of this progress. As a result, increasing attention has been devoted to providing formal guarantees concerning neural network behaviour (Liu et al., 2021b). Furthermore, deep learning practices have been noted to be often based on folklore observations and consolidated pipelines, rather than on rigorous understanding of the complex algorithms being employed (Sculley et al., 2018; Hutson, 2018). Luckily, a number of works has sought to provide thorough evaluations of pre-existing algorithms (Greff et al., 2017; Lucic et al., 2018), often revealing the competitive performance of simpler baselines (Brockschmidt, 2020; Narang et al., 2021).

In this thesis, we take a step towards trustworthy deep learning by developing or providing support for efficient algorithms for neural network verification and training. Before outlining the individual contributions presented in this thesis (§1.4), we now provide an introduction to neural network verification (§1.2) and neural network training in the contexts of interest (§1.3).

## 1.2 Neural Network Verification

Neural networks have the potential to be employed in a variety of safety-critical contexts, including healthcare (Jiang et al., 2017), autonomous driving (Wu et al., 2017) and collision avoidance for commercial aircraft (Julian et al., 2016). It is therefore crucial that these systems are robust and verify a number of desirable properties. The field of neural network verification aims to provide formal guarantees on the behaviour of deep learning systems by focusing on input-output specifications: proving that, if the network input lies in a certain domain $\mathcal{C}$, all the network output will satisfy a given property $P$. Typically, $\mathcal{C}$ is a compact set, and $P$ is a Boolean formula over linear inequalities, common examples of properties being robustness to adversarial examples, open-loop control specifications (Ehlers, 2017; Katz et al., 2017), or safety properties for use within computer systems (He et al., 2022). Verifying a specification is equivalent to provably finding the sign of the minimum of a non-convex optimization problem (Bunel et al., 2018).

### 1.2.1 Neural Network Bounds

While solving a non-convex problem is notoriously hard, bounds on the minimum can provide an answer for a subset of the specifications at hand: the tighter the bounds, the fewer the properties that will lack a definite answer. Upper bounds to the original problem can be obtained by running a local optimization algorithm: in the context of adversarial robustness, this corresponds to running an adversarial attack (Goodfellow et al., 2015; Carlini and Wagner, 2017; Madry et al., 2018). In other words, computing upper bounds amounts to looking for counter-examples to the property, yielding an *unsound* verification algorithm: some of the false properties can be proved false. Lower bounds can be computed via Lagrangian duality (Wong and Kolter, 2018; Dvijotham et al., 2018b) or by replacing the problem with a convex outer-approximation (Ehlers, 2017; Zhang et al., 2018; Raghunathan et al., 2018; Singh et al., 2018, 2019b,a; Anderson et al., 2020). These yield an *incomplete* verification algorithm, meaning that some of the true properties can be proved true.

A wide variety of incomplete verifiers exist, with radically different speed-accuracy trade-offs. Given that any piecewise-linear network can be represented by means of ReLU activations (Bunel et al., 2020b), much attention has been devoted to computing bounds on these activations. Interval bound propagation (Gowal et al., 2018a; Mirman et al., 2018) outer-approximate the ReLU via a rectangle, yielding very loose yet efficient bounds. A popular family of methods replaces activation functions by a pair of linear lower and upper bounds (bound propagation algorithms). The slopes of the linear functions can be parallel (Wong and Kolter, 2018; Singh et al., 2018), or vary to minimize the area of the resulting relaxation (Zhang et al., 2018; Singh et al., 2019b), yielding effective speed-accuracy trade-offs. While most of these algorithms are usually restricted to standard feed-forward networks, recent work has sought to extend these techniques to more general computational graphs (Xu et al., 2020). A tighter approximation replaces the ReLU by its triangle-shaped convex hull (Ehlers, 2017), and has been commonly been employed as a benchmark for relaxation tightness, to the point of being referred to as the convex barrier for neural network verification (Salman et al., 2019b). This relaxations is amenable to custom dual solvers (Dvijotham et al., 2018b), such as those we present in §2. In order to overcome the convex barrier, some bounding algorithms represent the convex hull of the composition of the activation with the preceding linear layer (Anderson et al., 2020; Tjandraatmadja et al., 2020), cross-ReLU interactions (Singh et al., 2019a), or rely on Semi-Definite Programming (SDP) (Raghunathan et al., 2018; Dvijotham et al., 2020). We present custom solvers for the relaxation by Anderson et al. (2020) in §3.

### 1.2.2 Verification via Branch-and-Bound

The methods that can verify any given specification (*complete verification*) tend to adhere to the Branch-and-Bound (BaB) paradigm, either implicitly or explicitly (Bunel et al., 2020b). In other words, they recursively divide the original verification problem into subproblems (*branching*), on which bounds are computed (*bounding*: §1.2.1) until a definite answer can be provided. In the following, we will

mostly focus on piecewise-linear networks, for which a solution can be found in a finite number of steps: in the worst-case, which is NP-complete (Katz et al., 2017), branching will proceed until all subproblems correspond to a linear piece of the function that the network represents. At that point, the solution of the original non-convex problem can be found by solving each of the resulting Linear Programs (LP).

Seminal works in the area cast verification as a Mixed Integer Linear Program (MILP) (Maganti, 2017; Tjeng et al., 2019) or as a Satisfiability Modulo Theory (SMT) problem (Ehlers, 2017; Katz et al., 2017), which are commonly tackled via black-box solvers. In order to improve on the scalability of black-box solvers, another line of work pairs inexpensive over-approximations (network relaxations) with custom branching strategies (Wang et al., 2018a,b). However, these methods are outperformed by the use of commercial LP solvers (Gurobi Optimization, 2020) on the triangle relaxation within custom Branch-and-Bound frameworks (Bunel et al., 2020b). For problems with a low-dimensional input size, branching can be performed by performing binary splits on the input domain (Bunel et al., 2018; Wang et al., 2018a). For higher-dimensional inputs, as it is often the case when verifying adversarial robustness, branching is more effectively executed by dividing activation functions into their linear building blocks (activation splitting) (Bunel et al., 2020b; Lu and Kumar, 2020). We present effective activation splitting algorithms in §2 and §4.

In the last two to three years, many toolboxes for neural network verification have been publicly released (Bak et al., 2020; Henriksen and Lomuscio, 2020; Singh et al., 2020; Katz et al., 2019). Furthermore, as exemplified by the iterations of the International Competition on the Verification of Neural Networks (VNN-COMP) (VNN-COMP, 2020; Bak et al., 2021), complete verifiers have achieved substantial speed-ups, easily scaling to networks with tens of thousands of neurons. These improvements, spearheaded by some of the contributions presented in this thesis (see §2, §3), are linked to the use of fast, yet tight, dual bounds within custom Branch-and-Bound frameworks running on specialized hardware such as

GPUs (Bunel et al., 2020a; De Palma et al., 2021a,b; Xu et al., 2021; De Palma et al., 2021c; Wang et al., 2021a).

## 1.3 Neural Network Training

Neural network verification focuses on providing guarantees on pre-trained neural networks. However, trustworthiness can and should be also achieved from the training process itself. In this thesis, we focus on two specific training problems that are related to trustworthiness. The first, which is tightly linked to our contributions on neural network verification, is to train a network for robustness and verifiability (§1.3.1). The second, which is more distinct from the remainder of the contributions presented in this thesis, is the problem of training a network to perform multiple tasks at once (§1.3.2). Deep multi-task learning has recently seen a surge in complexity and variety of training schemes. We are interested in understanding and evaluating their performances compared to simpler baselines (see our contribution in §5), hoping to increase trust in deep multi-task learning systems.

### 1.3.1 Training for Verifiability

Neural networks are trained to maximize performance by minimizing a corresponding surrogate loss. In order to enforce a property during the training process, the surrogate loss must contain terms related the desired specification. Ideally, this could be done by employing the output of a neural network verifier in the training process. However, owing to the complexity of complete neural network verification (see §1.2), training typically relies on inexpensive neural network bounds (see §1.2.1).

A family of training schemes, named adversarial training, looks for counter-examples (upper bounds) to the desired property, and computes the training loss on the output of the counter-example search (Madry et al., 2018). As the search is typically run for few iterations, adversarial training algorithms scale relatively well with network sizes. Furthermore, they produce so-called “empirical" robustness: they drastically reduce the possibility of empirically finding counter-examples to the desired property. Nevertheless, the empirical robustness resulting from these

schemes is associated with poor verifiability, meaning that it is hard to provide guarantees on the absence of counter-examples. In order to address this shortcoming, verified training schemes use the output of inexpensive incomplete verifiers (lower bounds) as training signals (Wong and Kolter, 2018; Gowal et al., 2018a; Zhang et al., 2020). As a result, the networks verifiably satisfy the desired properties, but at the cost of expensive training and inferior standard performance. Finally, a line of work has sought to combine the benefits of both approaches by suitably adapting adversarial training for verifiability (Xiao et al., 2019; Balunovic and Vechev, 2020). In chapter 4 we present a training algorithm belonging to the last category. While the discussed methods were proposed in the context of training for adversarial robustness, they are all applicable to general input-output specifications.

### 1.3.2 Deep Multi-Task Learning

Deep Multi-Task Learning (MTL) proposes to leverage commonalities across tasks to increase performance while reducing memory and compute. These tasks can be homogeneous (have the same loss function for each task), as for instance in the case of multi-label classification (Liu et al., 2015), or heterogeneous, as common in datasets for scene understanding (Cordts et al., 2016). The predominant approach is hard parameter sharing, which involves using the same network parameters across tasks, possibly with task-specific predictive heads. Applications of deep MTL range from natural language processing (Collobert and Weston, 2008; Chen et al., 2021), to computer vision (Misra et al., 2016) and reinforcement learning (Kalashnikov et al., 2021).

Popular research avenues in deep MTL include specialized architectures (Misra et al., 2016; Guo et al., 2020), and training schemes (Kendall et al., 2018; Chen and Gu, 2018). In this thesis, we focus on what we call Specialized Multi-Task Optimizers (SMTO): algorithms that operate complex and expensive gradient manipulations aimed at facilitating the multi-task training process (Sener and Koltun, 2018; Yu et al., 2020; Chen et al., 2020; Liu et al., 2021c; Wang et al., 2021b; Liu et al., 2021a). In chapter §5, we present a critical reevaluation of progress in the area.

## 1.4 Thesis Outline and Contributions

This thesis adheres to the integrated format, whereby each chapter from 2 to 5 corresponds to a separate paper for which the author of the present thesis is a (possibly joint) first author. We now provide an overview of the overall thesis' content. We then end this section by providing a per-chapter summary of this thesis' contributions.

The bulk of the content of this thesis pertains to the topic of neural network verification. Chapters 2 and 3 provide bounding algorithms that improve on the tightness/scalability trade-offs of the methods available at the time of each paper's submission (Bunel et al., 2020a; De Palma et al., 2021a,b), hence allowing for the verification of more properties in a given time. All our algorithms leverage the parallelism of specialized hardware like GPUs (relying on popular deep learning frameworks such as PyTorch (Paszke et al., 2019)), and are suitable for a wide range of computational budgets. These bounding algorithms are integrated within a massively parallel Branch-and-Bound framework, whose branching strategy is improved in chapter 4, allowing for the efficient use of the bounds for complete neural network verification (De Palma et al., 2021c). These contributions were all publicly released as part of the OVAL complete verification framework, available at `https://github.com/oval-group/oval-bab`. Our verification work laid the basis for our participation to the first and second iterations of VNN-COMP (VNN-COMP, 2020; Bak et al., 2021), co-hosted at International Conference on Computer Aided Verification (CAV). The OVAL framework finished in a joint third place at the second VNN-COMP, where the top-ranking entry incorporated some of our contributions: our branching strategy presented in chapter 2. Furthermore, chapter 4 presents an efficient and simple algorithm to train neural networks for verifiability, yielding state-of-the-art results on robustness to small adversarial perturbations.

Finally, chapter 5 presents a critical evaluation of deep multi-task optimizers: we demonstrate that a simple baseline outperforms complex specialized methods, and provide a partial explanation of our surprising results.

### 1.4.1 Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition

Chapter 2 extends Bunel et al. (2020a), which was published in the proceedings of the $36^{th}$ conference on Uncertainty in Artificial Intelligence (UAI 2020).

**Contributions.** By applying Lagrangian Decomposition (Guignard and Kim, 1987) to a neural network relaxation associated to the element-wise convex hull of the activation function (see §1.2.1), the UAI paper presents two efficient incomplete verifiers. It is shown that the presented methods improve on the speed-accuracy trade-offs of previous solvers for the same relaxation (Dvijotham et al., 2018b). The extended version presented in this thesis, which was publicly released on arXiv (De Palma et al., 2021c), designs an efficient branch and bound framework around the bounding algorithms, leading to substantial complete verification speed-ups. In particular, it differs from the conference version by (i) presenting a new heuristic branching scheme for activation splitting (§2.4); (ii) improving on various components of the branch and bound framework employed in the preliminary version (§2.5.1), resulting in large complete verification improvements; (iii) refining the analysis linking our dual problem to previous dual approaches (§2.2.2, §2.3.4), which now includes initialization via any propagation-based algorithm and a geometric explanation of the effectiveness of our dual compared to the one by Dvijotham et al. (2018b); and (iv) expanding the experimental analysis to include both new benchmarks, and new baselines such as CROWN (Zhang et al., 2018), ERAN (Singh et al., 2020), nnenum (Bak et al., 2020) and VeriNet (Henriksen and Lomuscio, 2020).

### 1.4.2 Scaling the Convex Barrier with Sparse Dual Algorithms

Chapter 3 is based on De Palma et al. (2021b), which is currently under review. De Palma et al. (2021b) extends De Palma et al. (2021a), which was published in the proceedings of the $9^{th}$ International Conference on Learning Representations (ICLR 2021).

**Contributions.** In order to overcome the convex barrier for neural network bounds (see §1.2.1), Anderson et al. (2020) present a novel formulation capturing the convex hull of the composition of a linear operator with piecewise-linear activation functions. While the relaxation comes at the cost of exponentially many constraints, Anderson et al. (2020) show that the most violated constraint at a given point can be computed in linear time (separation oracle) and present a primal solver. Chapter 3 presents two memory-efficient dual solvers for the formulation by Anderson et al. (2020) which, by appropriately resorting to the separation oracle, achieve tighter bounds than the primal solver in a fraction of the time. As a result, we show that we can accelerate the formal verification of hard specifications within a Branch-and-Bound framework. The first proposed solver operates by maintaining an active set of dual variables, incurring a memory cost proportional to the target relaxation tightness (De Palma et al., 2021a). By casting the bounding task as a saddle-point problem, we present a second solver (De Palma et al., 2021b) that can attain a memory footprint that is linear and independent of the target tightness, yielding large bounding improvements on memory-intensive settings.

### 1.4.3 IBP Regularization for Verified Adversarial Robustness via Branch-and-Bound

The subject of chapter 4 is De Palma et al. (2022), which was presented at the ICML 2022 Workshop on Formal Verification of Machine Learning, where it won the best paper award.

**Contributions.** We present IBP-R, a simple and effective algorithm to train neural networks for verifiability of adversarial robustness specifications (see § 1.3.1). In order to increase the verifiability of adversarially-trained networks, IBP-R proposes to run the adversarial attacks (counter-example search) over domains significantly larger than those to be verified. Intuitively, doing so increases the margin by which a network is robust. Furthermore, it adds a regularization term that minimizes the area of the commonly-employed triangle ReLU relaxation (see §1.2.1) in order to reduce network over-approximation at verification time. As a result, the trained

networks are amenable to fast verification via the OVAL verification framework, which is modified to include a less computationally intensive branching strategy. In chapter 4, we show that IBP-R obtains state-of-the-art results on small adversarial perturbations on CIFAR-10, while reducing training times.

### 1.4.4 In Defense of the Unitary Scalarization for Deep Multi-Task Learning

Chapter 5 is taken from Kurin et al. (2022), which was published in the proceedings of the $36^{th}$ Conference on Neural Information Processing Systems (NeurIPS 2022).

**Contributions.** Deep MTL is commonly assumed to pose additional training challenges compared to the single-task scenario. Popular explanations include conflicting gradient directions across tasks Yu et al. (2020); Chen et al. (2020), task imbalances (Kendall et al., 2018; Chen and Gu, 2018), and inter-task competition Sener and Koltun (2018). As a consequence, recent research has argued against *unitary scalarization*, where training simply minimizes the sum of the task losses, and proposed several specialized optimizers, each addressing some of the hypotheses behind the complexity of the multi-task scenario (SMTOs, see §1.3.2). These optimizers are complex to implement, and involve additional computation and memory costs, often linked to the requirement of per-task gradients. In chapter 5, we present a unified evaluation of the performance of SMTOs on popular MTL benchmarks. Surprisingly, our analysis shows that, taking experimental variability into account and possibly employing appropriate regularization and stabilization techniques from the single-task literature, unitary scalarization performs on par with specialized optimizers while often significantly reducing training times. As a partial explanation, we suggest that SMTOs reduce overfitting on the sum of per-task losses, and provide theoretical and empirical evidence to support our claim. We hope our results will drive further research towards a better understanding of deep MTL.

# 2

# Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition

## Contents


2.1 Introduction . . . . . **15**
2.2 Neural Network Verification . . . . . **18**
2.2.1 Problem Specification . . . . . 18
2.2.2 Neural Network Bounding . . . . . 19
2.2.3 Branch and Bound . . . . . 22
2.3 Better Bounding: Lagrangian Decomposition . . . . . **24**
2.3.1 Problem Derivation . . . . . 24
2.3.2 Supergradient Solver . . . . . 26
2.3.3 Proximal Solver . . . . . 28
2.3.4 Comparison to Previous Dual Problems . . . . . 31
2.4 Better Branching . . . . . **33**
2.4.1 Preliminaries: Approximations of Strong Branching . . . 33
2.4.2 Filtered Smart Branching . . . . . 35
2.5 Improved Branch and Bound . . . . . **37**
2.5.1 Additional Branch and Bound Improvements . . . . . 37
2.5.2 Implementation Details . . . . . 39
2.6 Related Work . . . . . **41**
2.7 Incomplete Verification Experiments . . . . . **42**
2.7.1 Experimental Setup . . . . . 42
2.7.2 Bounding Algorithms . . . . . 43
2.7.3 Results . . . . . 45
2.8 Complete Verification . . . . . **47**
2.8.1 Experimental Setup . . . . . 47

2.8.2 Bounding Algorithms . . . . . 49
2.8.3 Branching . . . . . 50
2.8.4 Intermediate Bounds . . . . . 53
2.8.5 Comparison of Complete Verifiers . . . . . 54
2.9 Discussion . . . . . **58**

# Abstract

We improve the scalability of Branch and Bound (BaB) algorithms for formally proving input-output properties of neural networks. First, we propose novel bounding algorithms based on Lagrangian Decomposition. Previous works have used off-the-shelf solvers to solve relaxations at each node of the BaB tree, or constructed weaker relaxations that can be solved efficiently, but lead to unnecessarily weak bounds. Our formulation restricts the optimization to a subspace of the dual domain that is guaranteed to contain the optimum, resulting in accelerated convergence. Furthermore, it allows for a massively parallel implementation, which is amenable to GPU acceleration via modern deep learning frameworks. Second, we present a novel activation-based branching strategy. By coupling an inexpensive heuristic with fast dual bounding, our branching scheme greatly reduces the size of the BaB tree compared to previous heuristic methods. Moreover, it performs competitively with a recent strategy based on learning algorithms, without its large offline training cost. Finally, we design a BaB framework, named Branch and Dual Network Bound (BaDNB), based on our novel bounding and branching algorithms. We show that BaDNB outperforms previous complete verification systems by a large margin, cutting average verification times by factors up to 50 on adversarial robustness properties.

## 2.1 Introduction

As deep learning powered systems become more and more common, the lack of robustness of neural networks and their reputation for being “black boxes" has become increasingly worrisome. In order to deploy them in critical scenarios where safety and robustness would be a prerequisite, techniques that can prove formal guarantees for neural network behavior are needed. A particularly desirable property is resistance to adversarial examples (Goodfellow et al., 2015; Szegedy et al., 2014): perturbations maliciously crafted with the intent of fooling even extremely well performing models. After several defenses were proposed and subsequently broken (Athalye et al., 2018; Uesato et al., 2018), some progress has been made in being able to formally verify whether there exist any adversarial examples in the neighborhood of a data point (Tjeng et al., 2019; Wong and Kolter, 2018).

Verification algorithms fall into three categories: unsound (some false properties are proven false), incomplete (some true properties are proven true), and complete (all properties are correctly verified as either true or false). Unsound verification, which relies on approximate non-convex optimization, is not related to the topic of this work. Instead, we focus on incomplete verification and its role in complete verification. An incomplete verifier can be obtained via the computation of lower and upper bounds on the output of neural networks. Many complete verifiers can be seen as branch and bound algorithms (Bunel et al., 2018), which operate by dividing the property into subproblems (branching) for which incomplete verifiers are more likely to provide a definite answer (bounding). Bunel et al. (2020b) have recently proposed a branch and bound framework that scales to medium-sized convolutional networks, outperforming state-of-the-art complete verifiers (Katz et al., 2017; Wang et al., 2018b; Tjeng et al., 2019). The aim of this work is to significantly improve their design choices, in order to scale up the applicability of complete verifiers to larger networks. In the remainder of this section, we provide a high-level overview of our proposed improvements.

**Bounding** Most previous algorithms for computing bounds are either computationally expensive (Ehlers, 2017) or sacrifice tightness in order to scale (Gowal et al., 2018b; Mirman et al., 2018; Wong and Kolter, 2018; Singh et al., 2018; Zhang et al., 2018). Within complete verification Bunel et al. (2018, 2020b) chose tightness over scalability, employing off-the-shelf solvers (Gurobi Optimization, 2020) to solve a network relaxation obtained by replacing activation functions by their convex hull (Ehlers, 2017). In the context of incomplete verification, better speed-accuracy trade-offs were achieved by designing specialized solvers for such relaxation (Dvijotham et al., 2018b). In this work, we design a novel dual formulation for the bounding problem and two corresponding solvers, which we employ as a branch and bound subroutine. Our approach offers the following advantages:

- While previous bounding algorithms operating on the same network relaxation (Dvijotham et al., 2018b) are based on Lagrangian relaxations, we derive a new family of optimization problems for neural network bounds through Lagrangian Decomposition, which in general yields duals at least as strong as those obtained through Lagrangian relaxation (Guignard and Kim, 1987). For our bounding problem, the optimal solutions of both the Lagrangian Decomposition and Lagrangian relaxation will match. However we prove that, in the context of ReLU networks, for any dual bound from the approach by Dvijotham et al. (2018b) obtained in the process of dual optimization, the corresponding bounds obtained by our dual are at least as tight. Geometrically, our dual corresponds to a reduction of the dual space of the Lagrangian relaxation that always contains the optimum. We demonstrate empirically that our derivation computes tighter bounds in the same time when using supergradient methods, improving the quality of incomplete verification. We further refine performance by devising a proximal solver for the problem, which decomposes the task into a series of strongly convex subproblems. For each subproblem, we use an iterative method that lends itself to analytical optimal step sizes, thereby resulting in faster convergence.

- Both the supergradient and the proximal method hinge on linear operations similar to those used during network forward/backward passes. As a consequence, we can leverage the convolutional structure when necessary, while standard solvers are often restricted to treating it as a general linear operation. Moreover, both methods are easily parallelizable: when computing bounds on the activations at layer $k$, we need to solve two problems for each hidden unit of the network (for the upper and lower bounds). These can all be solved in parallel. Within branch and bound, we need to compute bounds for several different problem domains at once: we solve these problems in parallel as well. Our GPU implementation thus allows us to solve several hundreds of linear programs at once on a single GPU, a level of parallelism that would be hard to match on CPU-based systems.

**Branching** While bounding is often the computational bottleneck within each branch and bound iteration, a high quality branching strategy is crucial to reduce the branch and bound search tree (Achterberg and Wunderling, 2013). Strategies used for neural network verification typically split the domain on a coordinate of the network input (Wang et al., 2018a; Bunel et al., 2018; Royo et al., 2019), or on a given network activation (Ehlers, 2017; Katz et al., 2017; Wang et al., 2018b). It was recently shown (Bunel et al., 2020b) that, for convolutional networks with around one thousand neurons, it is preferable to split on the network activations (activation splitting). As this search space is significantly larger, the best-performing heuristic strategy favors computational efficiency over accuracy (Bunel et al., 2020b). In order to improve performance without significantly increasing branching costs, strategies based on learning algorithms were proposed recently (Lu and Kumar, 2020). We present a novel branching strategy that, by coupling an inexpensive heuristic with fast dual bounds, greatly improves upon previous approaches strategies (Bunel et al., 2020b) and performs competitively with learning algorithms without incurring large training costs.

**BaDNB** We design a massively parallel, GPU-accelerated, branch and bound framework around our bounding and branching algorithms. We conduct detailed ablation studies over the various components of the framework, named BaDNB, and show that it yields substantial complete verification speed-ups over the state-of-the-art algorithms.

The chapter is organized as follows: in section 2.2, we state the neural network verification problem and describe the technical background necessary for the understanding of our approach. Section 2.3 presents our novel formulation for neural network bounding, yielding efficient incomplete verifiers. Section 2.4 presents our branching scheme, to be used within branch and bound for complete verification. Technical and implementation details of BaDNB are outlined in section 2.5. In section 2.6, we discuss related work in the context of our contributions. Finally, sections 2.7 and 2.8 present an experimental evaluation of both our bounding algorithms and the branch and bound framework.

## 2.2 Neural Network Verification

Throughout this chapter, we will use bold lower case letters (for example, $\mathbf{x}$) to represent vectors and upper case letters (for example, $W$) to represent matrices. Brackets are used to indicate intervals ($[\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k]$) and vector or matrix entries ($\mathbf{x}[i]$ or $W[i, j]$). Moreover, we use $\odot$ for the Hadamard product, $[\![\cdot, \cdot]\!]$ for integer ranges, $\mathbb{1}_{\mathbf{a}}$ for the indicator vector on condition $\mathbf{a}$. Finally, we write $\text{Conv}(f, \mathbf{a}, \mathbf{b})$ and $\text{Conv}(S)$ respectively for the convex hull of function $f$ defined in $[\mathbf{a}, \mathbf{b}]$, and for the convex hull of set $S$.

We begin by formally introducing the problem of neural network verification (§2.2.1), followed by an outline of two popular solution strategies (§2.2.2, §2.2.3).

### 2.2.1 Problem Specification

Given a $d$-layer feedforward neural network $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_d}$, an input domain $\mathcal{C}$, and a property $P$, verification problem $(f, \mathcal{C}, P)$ is defined as follows:

$$\mathbf{x}_0 \in \mathcal{C} \ \wedge \ \mathbf{x}_d = f(\mathbf{x}_0) \implies P(\mathbf{x}_d).$$

Under the assumption that $P$ is a Boolean formula over linear inequalities (for instance, robustness to adversarial examples), we can represent both $f$ and $P$ as an $n$-layer neural network $f_P : \mathbb{R}^{n_0} \to \mathbb{R}$, which is said to be in canonical form if, for any $\mathbf{x}_0 \in \mathbb{R}^{n_0}$, $P(f(\mathbf{x}_0)) \implies f_P(\mathbf{x}_0) \geq 0$ (Bunel et al., 2018, 2020b). Verifying $(f, \mathcal{C}, P)$ then reduces to finding the sign of the minimum of the following optimization problem:

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

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

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

where constraints (2.1b) implement the linear layers of $f_P$ (fully connected or convolutional), while constraints (2.1c) implement its non-linear activation functions. We call $\hat{\mathbf{x}}_k \in \mathbb{R}^{n_k}$ pre-activations at layer $k$ and $n_k$ denotes the layer's width. In line with Dvijotham et al. (2018b), we assume that linear functions can be easily optimized over $\mathcal{C}$. In the following, we will first describe how to solve problem (2.1) approximately (§2.2.2), then exactly (§2.2.3).

### 2.2.2 Neural Network Bounding

The non-linearity of constraint (2.1c) makes problem (2.1) non-convex and NP-HARD (Katz et al., 2017). Therefore, many authors (Wong and Kolter, 2018; Dvijotham et al., 2018b; Zhang et al., 2018; Raghunathan et al., 2018; Singh et al., 2019b) have instead focused on the computation of a lower bound on the minimum, which significantly simplifies the optimization problem thereby yielding an efficient incomplete verification method. Here, we are concerned with approaches that allow for a dual interpretation (see §2.6 for an overview).

#### 2.2.2.1 Propagation-based methods

Assume we have access to upper and lower bounds (respectively $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{l}}_k$) on the value that $\hat{\mathbf{x}}_k$ can take, for $k \in [\![1, n-1]\!]$. We call these *intermediate bounds*: we detail how to compute them in §2.2.2.3. Moreover, let $\underline{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k) = \underline{\mathbf{a}}_k \odot \hat{\mathbf{x}}_k + \underline{\mathbf{b}}_k$

and $\bar{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k) = \bar{\mathbf{a}}_k \odot \hat{\mathbf{x}}_k + \bar{\mathbf{b}}_k$ be two linear functions that bound $\sigma_k(\hat{\mathbf{x}}_k)$ from below and above, respectively. Then, problem (2.1) can be replaced by the following convex outer approximation:

$$\begin{aligned} \min_{\mathbf{x},\hat{\mathbf{x}}} \quad & \hat{x}_n \qquad \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} \quad k \in [\![0, n-1]\!], \\ & \underline{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k) \leq \mathbf{x}_k \leq \bar{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k) \quad k \in [\![1, n-1]\!], \\ & \hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \qquad\qquad k \in [\![1, n-1]\!]. \end{aligned} \tag{2.2}$$

A popular and inexpensive class of bounding algorithms solves a relaxation of problem (2.2) by back-propagating $\underline{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k)$ and $\bar{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k)$ through the network (Wong and Kolter, 2018; Weng et al., 2018; Singh et al., 2018; Zhang et al., 2018; Singh et al., 2019b). In the dual space, these methods correspond to evaluating the Lagrangian relaxation of problem (2.2) at a specific dual point. Let us denote $[\mathbf{a}]_- = \min(0, \mathbf{a})$, $[\mathbf{a}]_+ = \max(0, \mathbf{a})$. The Lagrangian relaxation of problem (2.2) can be written in the following unconstrained form (Salman et al., 2019b, equations (8), (9), (38)):

$$\begin{aligned} & \max_{\boldsymbol{\mu},\boldsymbol{\lambda}} \; d_P(\boldsymbol{\mu}, \boldsymbol{\lambda}), \quad \text{where:} \\ d_P(\boldsymbol{\mu}, \boldsymbol{\lambda}) = \min_{\mathbf{x},\hat{\mathbf{x}}} & \left[ \begin{array}{l} W_n\mathbf{x}_{n-1} + b_n + \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T \left(\hat{\mathbf{x}}_k - W_k\mathbf{x}_{k-1} - \mathbf{b}_k\right) \\ + \sum_{k=1}^{n-1} [\boldsymbol{\lambda}_k]_-^T \left(\mathbf{x}_k - \underline{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k)\right) + \sum_{k=1}^{n-1} [\boldsymbol{\lambda}_k]_+^T \left(\mathbf{x}_k - \bar{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k)\right) \end{array} \right. \\ \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\ & \hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \qquad k \in [\![1, n-1]\!]. \end{aligned} \tag{2.3}$$

Salman et al. (2019b) show that propagation-based bounding algorithms are equivalent to evaluating problem (2.3) at a suboptimal point $(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}})$, given by:

$$\begin{aligned} & \bar{\boldsymbol{\lambda}}_{n-1} = -W_n^T, \\ & \bar{\boldsymbol{\mu}}_k = \bar{\mathbf{a}}_k \odot [\bar{\boldsymbol{\lambda}}_k]_+ + \underline{\mathbf{a}}_k \odot [\bar{\boldsymbol{\lambda}}_k]_- \quad k \in [\![1, n-1]\!], \\ & \bar{\boldsymbol{\lambda}}_{k-1} = W_k^T \bar{\boldsymbol{\mu}}_k \qquad\qquad\qquad k \in [\![2, n-1]\!]. \end{aligned} \tag{2.4}$$

The dual assignment (2.4) is obtained via a single *backward pass* through the network, an operation analogous to the gradient backpropagation employed for neural network training. Moreover, exploiting the structure of equation (2.4), the objective value of problem (2.3) at such dual point can be conveniently computed as:

$$d_P(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}) = \min_{\mathbf{x}_0 \in \mathcal{C}} \left(-\bar{\boldsymbol{\mu}}_1^T W_1 \mathbf{x}_0\right) + b_n - \sum_{k=1}^{n-1} \left([\bar{\boldsymbol{\lambda}}_k]_-^T \underline{\mathbf{b}}_k + [\bar{\boldsymbol{\lambda}}_k]_+^T \bar{\mathbf{b}}_k + \bar{\boldsymbol{\mu}}_k^T \mathbf{b}_k\right). \tag{2.5}$$

#### 2.2.2.2 Lagrangian Relaxation of the non-convex formulation

Propagation-based methods provide a lower bound to problem (2.2), which is a rather loose approximation to problem (2.1). An alternative approach, presented by Dvijotham et al. (2018b), relies on taking the dual of non-convex problem (2.1) directly and solving it via supergradient methods. By relaxing (2.1b) and (2.1c) via Lagrangian multipliers, and exploiting intermediate bounds, Dvijotham et al. (2018b) obtain the following dual:

$$\begin{aligned} &\max_{\boldsymbol{\mu},\boldsymbol{\lambda}} \; d_O(\boldsymbol{\mu},\boldsymbol{\lambda}), \quad \text{where:} \\ d_O(\boldsymbol{\mu},\boldsymbol{\lambda}) = \min_{\mathbf{x},\hat{\mathbf{x}}} &\left[ \begin{array}{l} \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T \left(\hat{\mathbf{x}}_k - W_k \mathbf{x}_{k-1} - \mathbf{b}_k\right) + \sum_{k=1}^{n-1} \boldsymbol{\lambda}_k^T \left(\mathbf{x}_k - \sigma_k(\hat{\mathbf{x}}_k)\right) \\ + W_n \mathbf{x}_{n-1} + b_n \end{array} \right. \\ \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in [\sigma_k(\hat{\mathbf{l}}_k), \sigma_k(\hat{\mathbf{u}}_k)] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \qquad k \in [\![1, n-1]\!]\,. \end{aligned} \tag{2.6}$$

For $\sigma_k(\hat{\mathbf{x}}_k) = \max(\hat{\mathbf{x}}_k, 0)$, Dvijotham et al. (2018b), prove that problem (2.6) is equivalent to a dual of the following convex problem[1]:

$$\begin{aligned} \min_{\mathbf{x},\hat{\mathbf{x}}} \quad \hat{x}_n \qquad \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!]\,, \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in \text{Conv}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) && k \in [\![1, n-1]\!]\,, \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in [\sigma_k(\hat{\mathbf{l}}_k), \sigma_k(\hat{\mathbf{u}}_k)] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] && k \in [\![1, n-1]\!]\,, \end{aligned} \tag{2.7}$$

where $\text{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ is the convex hull of constraint (2.1c). Salman et al. (2019b) generalize the result to any activation function that acts element-wise on $\hat{\mathbf{x}}_k$ and prove that, under mild assumptions, strong duality holds for problem (2.7). Therefore, as $\text{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) \subseteq \{(\mathbf{x}_k, \hat{\mathbf{x}}_k) \,|\boldsymbol{\sigma}_k(\hat{\mathbf{x}}_k) \leq \mathbf{x}_k \leq \bar{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k)\}$, the bounding algorithm by Dvijotham et al. (2018b) will converge to tighter bounds than propagation-based algorithms.

#### 2.2.2.3 Intermediate bounds

Convex relaxations (for instance, problems (2.2), (2.7)) and dual problems (for instance, problem (2.6)) are often defined as a function of intermediate bounds

[1] Dvijotham et al. (2018b) write $\text{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) = \{(\mathbf{x}_k, \hat{\mathbf{x}}_k) \,|\boldsymbol{\sigma}_{k,\text{opt}}(\hat{\mathbf{x}}_k) \leq \mathbf{x}_k \leq \bar{\boldsymbol{\sigma}}_{k,\text{opt}}(\hat{\mathbf{x}}_k)\}$, and consider the dual that results from relaxing both inequalities, along with equality (2.1b).

$\hat{\mathbf{l}}_j$ and $\hat{\mathbf{u}}_j$ on the values of $\hat{\mathbf{x}}_j \in \mathbb{R}^{n_j}$. These values are computed by running a bounding algorithm over subnetworks. Specifically, we are looking for bounds to versions of problem (2.1) for which, instead of defining the objective function on the activation of layer $n$, we define it over $\hat{\mathbf{x}}_j[i]$. As we need to repeat this process for $j \in [\![1, n-1]\!]$ and $i \in [\![1, n_j]\!]$, intermediate bound computations can easily become the computational bottleneck for neural network bounding. Therefore, typically, intermediate bounds are computed with inexpensive propagation-based algorithms (§2.2.2.1), whereas the lower bounding of the network output $\hat{x}_n$ relies on more costly convex relaxations (outlined in §2.2.2.2) (Bunel et al., 2020b).

### 2.2.3 Branch and Bound

Neural network bounding is concerned with solving an approximation of problem (2.1), and may verify a subset of the properties: those for which the computed lower bound is positive. However, in order to guarantee that any given property will be verified, we need to solve problem (2.1) exactly. The lack of convexity rules out local optimization algorithms such as gradient descent, which will not provably converge to the global optimum. Therefore, many complete verification methods are akin to global optimization algorithms such as branch and bound (see §2.6) (Bunel et al., 2018).

#### 2.2.3.1 Operating principle

In the context of our verification problem (2.1), branch and bound starts by computing bounds on the minimum: a lower bound is obtained via a *bounding algorithm* (see §2.2.2), while an upper bound can be determined heuristically, as any feasible point yields a valid upper bound. If the property cannot yet be verified (that is, the lower bound is negative and the upper bound is positive), the property's feasible domain is divided into a number of smaller problems via some *branching strategy*. The algorithm then proceeds by computing bounds for each subproblem, exploiting the fact that a subproblem's lower bound is guaranteed to be at least as tight as the one for its parent problem (that is, before the branching). Subproblems

which cannot contain the global lower bound are progressively discarded: in the canonical form (see §2.2.1), this happens if a local lower bound is positive. An incumbent solution to problem (2.1) is defined as the smallest encountered upper bound. The order in which subproblems are explored is determined by a *search strategy*. Finally, the verification procedure terminates when either no subproblem has a negative lower bound, or when the incumbent becomes positive.

#### 2.2.3.2 Branch and bound for piecewise-linear networks

We now turn our attention to the class of piecewise-linear networks. For simplicity, we assume all the activation functions are ReLUs, as other common piecewise-linear activations such as MaxPooling units can be converted into a series of ReLU-based layers (Bunel et al., 2020b). We describe BaBSR from Bunel et al. (2020b), a specific instantiation of branch and bound that proved particularly effective in the context of larger piecewise-linear networks.

Let us classify ReLU activations depending on the signs of pre-activation bounds $\hat{\mathbf{l}}_k$ and $\hat{\mathbf{u}}_k$. A given ReLU $\sigma_k(\hat{\mathbf{x}}_k[i]) = \max(\hat{\mathbf{x}}_k[i], 0)$ is passing if $\hat{\mathbf{l}}_k[i] \geq 0$, blocking if $\hat{\mathbf{u}}_k[i] \leq 0$, and ambiguous otherwise. Note that non-ambiguous ReLUs can be replaced by linear functions. At every iteration, BaSBR picks the subproblem with the lowest lower bound, and branches by separating an ambiguous ReLU into its two linear phases (*ReLU branching*). The ReLU on which to branch is selected according to a heuristic that estimates the effect of the split on the subproblem lower bound, based on an inexpensive approximation of the bounding algorithm by Wong and Kolter (2018) (for details, see §2.4). Lower bounding is performed by solving the Linear Program (LP) corresponding to the ReLU version of problem

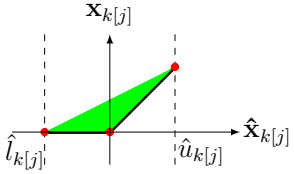



**Figure 2.1:** Feasible domain of the convex hull for an ambiguous ReLU. Red circles indicate the vertices of the feasible region.

(2.7), where the convex hull is defined as follows (Ehlers, 2017):

$$\mathrm{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) := \begin{cases} \mathbf{x}_k \geq 0, \quad \mathbf{x}_k \geq \hat{\mathbf{x}}_k \\ \mathbf{x}_k \leq \frac{\hat{\mathbf{u}}_k \odot (\hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k)}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} & \text{if } \hat{\mathbf{l}}_k \leq 0 \text{ and } \hat{\mathbf{u}}_k \geq 0, \quad (2.8a) \\ \mathbf{x}_k = 0 & \text{if } \hat{\mathbf{u}}_k \leq 0, \quad (2.8b) \\ \mathbf{x}_k = \hat{\mathbf{x}}_k & \text{if } \hat{\mathbf{l}}_k \geq 0. \quad (2.8c) \end{cases}$$

Upper bounds are computed by evaluating the neural network at the solution of the LP. Finally, the intermediate bounds for each LP are obtained by taking the layer-wise best bonds between interval bound propagation (Gowal et al., 2018b; Mirman et al., 2018) and the propagation-based method by Wong and Kolter (2018).

In the remainder of this chapter, we present improvements to both the bounding algorithm (§2.3) and the branching strategy (§2.4), then outline the details of the resulting branch and bound framework (§2.5).

## 2.3 Better Bounding: Lagrangian Decomposition

We will now describe a novel dual approach to obtain a lower bound to problem (2.1) and relate it to the duals described in section 2.2.2. We present two bounding algorithms: a supergradient method (§2.3.2), and a solver based on proximal maximization (§2.3.3).

### 2.3.1 Problem Derivation

Our approach is based on Lagrangian Decomposition, also known as variable splitting (Guignard and Kim, 1987). Due to the compositional structure of neural networks, most constraints involve only a limited number of variables. As a result, we can split the problem into meaningful, easy to solve subproblems. We then impose constraints that the solutions of the subproblems should agree.

We start from problem (2.7), a convex outer approximation of the original non-convex problem (2.1) where activation functions are replaced by their convex hull. In the following, we will use ReLU activation functions as an example: their convex hull is defined in equation (2.8). We stress that the derivation can be

extended to other non-linearities. For example, appendix A.3 describes the case of sigmoid activation function. In order to obtain an efficient decomposition, we divide the constraints into subsets that allow for easy optimization subtasks. Each subset will correspond to a pair of an activation layer, and the linear layer coming after it. The only exception is the first linear layer which is combined with the restriction of the input domain to $\mathcal{C}$. Using this grouping of the constraints, we can concisely write problem (2.7) as:

$$\begin{aligned} \min_{\mathbf{x},\hat{\mathbf{x}}} \quad & \hat{x}_n \qquad \text{s.t.} \qquad \mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_1), \\ & \qquad\qquad\quad \mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_k, \hat{\mathbf{x}}_{k+1}) \qquad k \in [\![1, n-1]\!], \end{aligned} \tag{2.9}$$

where the constraint subsets are defined as:

$$\mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_1) := \begin{cases} \mathbf{x}_0 \in \mathcal{C} \\ \hat{\mathbf{x}}_1 = W_1 \mathbf{x}_0 + \mathbf{b}_1, \end{cases}$$

$$\mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_k, \hat{\mathbf{x}}_{k+1}) := \begin{cases} (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in \mathrm{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k), \\ (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in [\sigma_k(\hat{\mathbf{l}}_k), \sigma_k(\hat{\mathbf{u}}_k)] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k], \\ \hat{\mathbf{x}}_{k+1} = W_{k+1} \mathbf{x}_k + \mathbf{b}_k. \end{cases}$$

To obtain a Lagrangian Decomposition, we duplicate the variables so that each subset of constraints has its own copy of the variables in which it is involved. Formally, we rewrite problem (2.9) as follows:

$$\min_{\mathbf{x},\hat{\mathbf{x}}} \quad \hat{x}_n \qquad \text{s.t.} \quad \mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}), \tag{2.10a}$$

$$\mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}) \qquad k \in [\![1, n-1]\!], \tag{2.10b}$$

$$\hat{\mathbf{x}}_{A,k} = \hat{\mathbf{x}}_{B,k} \qquad k \in [\![1, n-1]\!]. \tag{2.10c}$$

The additional equality constraints (2.10c) impose agreements between the various copies of variables. We introduce the dual variables $\boldsymbol{\rho}$ and derive the Lagrangian dual:

$$\begin{aligned} \max_{\boldsymbol{\rho}} \quad & q(\boldsymbol{\rho}), \quad \text{where:} \qquad q(\boldsymbol{\rho}) = \min_{\mathbf{x},\hat{\mathbf{x}}} \; \hat{x}_{A,n} + \sum_{k=1}^{n-1} \boldsymbol{\rho}_k^T (\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}) \\ & \text{s.t.} \quad \mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}), \\ & \qquad\quad \mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}) \qquad k \in [\![1, n-1]\!]. \end{aligned} \tag{2.11}$$

Any value of $\boldsymbol{\rho}$ provides a valid lower bound by virtue of weak duality. While we will maximize over the choice of dual variables in order to obtain as tight a bound

as possible, we will be able to interrupt the optimization process at any point and obtain a valid bound by evaluating $q$. It remains to show how to solve problem (2.11) efficiently in practice: this is the subject of §2.3.2 and §2.3.3. In §2.3.4, we analyze the relationship between our dual (2.11), propagation-based methods and problem (2.6).

### 2.3.2 Supergradient Solver

In line with the work by Dvijotham et al. (2018b), who use supergradient methods on their dual (2.6), we present a supergradient-based solver (Algorithm 1) for problem (2.11).

At a given point $\boldsymbol{\rho}$, obtaining the supergradient requires us to know the values of $\hat{\mathbf{x}}_A$ and $\hat{\mathbf{x}}_B$ for which the inner minimization is achieved. Based on the identified values of $\hat{\mathbf{x}}_A^*$ and $\hat{\mathbf{x}}_B^*$, we can then compute the supergradient $\nabla_{\boldsymbol{\rho}} q = \hat{\mathbf{x}}_B^* - \hat{\mathbf{x}}_A^*$ and move in its direction:

$$\boldsymbol{\rho}^{t+1} = \boldsymbol{\rho}^t + \alpha^t \nabla_{\boldsymbol{\rho}} q(\boldsymbol{\rho}^t), \tag{2.12}$$

where $\alpha^t$ corresponds to a step size schedule that needs to be provided. It is also possible to use any variant of gradient descent, such as Adam (Kingma and Ba, 2015).

It remains to show how to perform the inner minimization over the primal variables. By design, each of the variables is only involved in one subset of constraints. As a result, the computation completely decomposes over the subproblems, each corresponding to one of the subset of constraints. We therefore simply need to optimize linear functions over one subset of constraints at a time.

**Algorithm 1** Supergradient method

**function** SUPERGRADIENT_COMPUTE_BOUNDS(Problem (2.7))
  Initialize dual variables $\boldsymbol{\rho}^0$ via proposition 2
  **for** $t \in [\![0, T-1]\!]$ **do**
    $\hat{\mathbf{x}}^{*,t} \in \operatorname{argmin}_{\mathbf{x},\hat{\mathbf{x}}} q(\boldsymbol{\rho})$ as detailed in §2.3.2.1 and §2.3.2.2 ▷ inner minimization
    $\nabla_{\boldsymbol{\rho}} q(\boldsymbol{\rho}^t) \leftarrow \hat{\mathbf{x}}_B^{*,t} - \hat{\mathbf{x}}_A^{*,t}$ ▷ compute supergradient
    $\boldsymbol{\rho}^{t+1} \leftarrow \boldsymbol{\rho}^t + \alpha^t \nabla_{\boldsymbol{\rho}} q(\boldsymbol{\rho}^t)$ ▷ supergradient step
  **return** $q(\boldsymbol{\rho}^T)$

#### 2.3.2.1 Inner minimization: $\mathcal{P}_0$ subproblems

To minimize over $\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}$, the variables constrained by $\mathcal{P}_0$, we need to solve:

$$\begin{aligned}(\mathbf{x}_0^*, \hat{\mathbf{x}}_{A,1}^*) = \underset{\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}}{\operatorname{argmin}} \; & -\boldsymbol{\rho}_1^T \hat{\mathbf{x}}_{A,1} \\ \text{s.t} \quad & \mathbf{x}_0 \in \mathcal{C}, \qquad \hat{\mathbf{x}}_{A,1} = W_1 \mathbf{x}_0 + \mathbf{b}_0.\end{aligned} \tag{2.13}$$

Rewriting the problem as a linear function of $\mathbf{x}_0$ only, problem (2.13) is simply equivalent to minimizing $-\boldsymbol{\rho}_1^T W_1 \mathbf{x}_0$ over $\mathcal{C}$. We assumed that the optimization of a linear function over $\mathcal{C}$ was efficient. We now give examples of $\mathcal{C}$ where problem (2.13) can be solved efficiently.

**Bounded Input Domain** If $\mathcal{C}$ is defined by a set of lower bounds $\mathbf{l}_0$ and upper bounds $\mathbf{u}_0$ (as in the case of $\ell_\infty$ adversarial examples), optimization will simply amount to choosing either the lower or upper bound depending on the sign of the linear function. The optimal solution is:

$$\mathbf{x}_0 = \mathbb{1}_{\boldsymbol{\rho}_1^T W_1 < 0} \odot \hat{\mathbf{l}}_0 + \mathbb{1}_{\boldsymbol{\rho}_1^T W_1 \geq 0} \odot \hat{\mathbf{u}}_0, \qquad \hat{\mathbf{x}}_{A,1} = W_1 \mathbf{x}_0 + \mathbf{b}_1. \tag{2.14}$$

**$\ell_2$ Balls** If $\mathcal{C}$ is defined by an $\ell_2$ ball of radius $\epsilon$ around a point $\bar{\mathbf{x}}$ ($\|\mathbf{x}_0 - \bar{\mathbf{x}}\|_2 \leq \epsilon$), optimization amounts to choosing the point on the boundary of the ball such that the vector from the center to it is opposed to the cost function. Formally, the optimal solution is given by:

$$\hat{\mathbf{x}}_{A,1} = W_1 \mathbf{x}_0 + \mathbf{b}_1, \qquad \mathbf{x}_0 = \bar{\mathbf{x}} + (\epsilon / \|\boldsymbol{\rho}_1\|_2) \, \boldsymbol{\rho}_1. \tag{2.15}$$

#### 2.3.2.2 Inner minimization: $\mathcal{P}_k$ subproblems

For the variables constrained by subproblem $\mathcal{P}_k$ $(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1})$, we need to solve:

$$\begin{aligned}(\hat{\mathbf{x}}_{B,k}^*, \hat{\mathbf{x}}_{A,k+1}^*) = \underset{\hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}}{\operatorname{argmin}} \quad & \boldsymbol{\rho}_k^T \hat{\mathbf{x}}_{B,k} - \boldsymbol{\rho}_{k+1}^T \hat{\mathbf{x}}_{A,k+1} \\ \text{s.t} \quad & (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in \operatorname{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k), \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in [\sigma_k(\hat{\mathbf{l}}_k), \sigma_k(\hat{\mathbf{u}}_k)] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k], \\ & \hat{\mathbf{x}}_{A,k+1} = W_{k+1} \mathbf{x}_k + \mathbf{b}_{k+1}.\end{aligned} \tag{2.16}$$

We outline the minimization steps for the case of ReLU activation functions. However, the following can be generalized to other activations. For example, appendix A.3 describes the solution for the sigmoid function. For $\sigma_k(\hat{\mathbf{x}}_k) = \max(\mathbf{0}, \hat{\mathbf{x}}_k)$, $\text{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ is given by equation (2.8), and we can find a closed form solution. Using the last equality of the problem, and omitting constant terms, we can start by rewriting the objective function as $\boldsymbol{\rho}_k^T \hat{\mathbf{x}}_{B,k} - \boldsymbol{\rho}_{k+1}^T W_{k+1} \mathbf{x}_k$. The optimization can be performed independently for each of the coordinates, over which both the objective function and the constraints decompose completely. The ensuing minimization will then depend on whether a given ReLU is ambiguous or equivalent to a linear function.

**Ambiguous ReLUs** If the ReLU is ambiguous, the shape of the convex hull is represented in Figure 2.1. For each dimension $i$, problem (2.16) is a linear program, which means that the optimal point will be a vertex. The possible vertices for $(\hat{\mathbf{x}}_{B,k}[i], \mathbf{x}_k[i])$ are $(\hat{\mathbf{l}}_k[i], 0)$, $(0, 0)$, and $(\hat{\mathbf{u}}_k[i], \hat{\mathbf{u}}_k[i])$. In order to find the minimum, we can therefore evaluate the objective function at these three points and keep the one with the smallest value. Denoting the vertex set by $V_{k,i}$:

$$\underset{(\hat{\mathbf{x}}_{B,k}[i], \mathbf{x}_k[i]) \in V_{k,i}}{\text{argmin}} \left( \boldsymbol{\rho}_k[i] \hat{\mathbf{x}}_{B,k}[i] - (\boldsymbol{\rho}_{k+1}^T W_{k+1})[i] \mathbf{x}_k[i] \right). \tag{2.17}$$

**Non-ambiguous ReLUs** If for a ReLU we have $\mathbf{l}_k[i] \geq 0$ or $\mathbf{u}_k[i] \leq 0$, $\text{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ is a simple linear equality constraint. For those coordinates, the problem is analogous to the one solved by equation (2.14), with the linear function being minimized over the $\hat{\mathbf{x}}_{B,k}[i]$ box bounds being $\boldsymbol{\rho}_k^T[i] \hat{\mathbf{x}}_{B,k}[i]$ in the case of blocking ReLUs or $\left(\boldsymbol{\rho}_k^T - \boldsymbol{\rho}_{k+1}^T W_{k+1}\right)[i] \hat{\mathbf{x}}_{B,k}[i]$ in the case of passing ReLUs.

### 2.3.3 Proximal Solver

We now present a second solver (Algorithm 2) for problem (2.11), relying on proximal maximization rather than supergradient methods (as in §2.3.2).

#### 2.3.3.1 Augmented Lagrangian

Applying proximal maximization to the dual function $q$ results in the Augmented Lagrangian Method, which is also known as the method of multipliers. Let us indicate the value of a variable at the $t$-th iteration via superscript $t$. For our problem[2], the method of multipliers will correspond to alternating between the following updates to the dual variables $\boldsymbol{\rho}$:

$$\boldsymbol{\rho}_k^{t+1} = \boldsymbol{\rho}_k^t + \frac{\hat{\mathbf{x}}_{B,k}^t - \hat{\mathbf{x}}_{A,k}^t}{\eta_k}, \tag{2.18}$$

and updates to the primal variables $\hat{\mathbf{x}}$, which are carried out as follows:

$$\begin{aligned}
\left(\mathbf{x}^t, \hat{\mathbf{x}}^t\right) &= \underset{\mathbf{x},\hat{\mathbf{x}}}{\operatorname{argmin}} \mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}^t\right), \qquad \text{where:} \\
\mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}^t\right) &:= \hat{\mathbf{x}}_{A,n} + \sum_{k=1}^{n-1} \boldsymbol{\rho}_k^T \left(\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}\right) + \sum_{k=1}^{n-1} \frac{1}{2\eta_k} \|\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}\|^2 \\
&\text{s.t.} \quad \mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}), \\
&\qquad \mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}) \qquad k \in [\![1, n-1]\!] .
\end{aligned} \tag{2.19}$$

The term $\mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}\right)$ is the Augmented Lagrangian of problem (2.10). The additional quadratic term in (2.19), compared to the objective of $q(\boldsymbol{\rho})$, arises from the proximal terms on $\boldsymbol{\rho}$. It has the advantage of making the problem strongly convex, and hence easier to optimize. Later on, we will show that this allows us to derive optimal step-sizes in closed form. The weight $\eta_k$ is a hyperparameter of the algorithm. A high value will make the problem more strongly convex and therefore quicker to solve, but it will also limit the ability of the algorithm to perform large updates.

While obtaining the new values of $\boldsymbol{\rho}$ is trivial using equation (2.18), problem (2.19) does not have a closed-form solution. We show how to solve it efficiently nonetheless.

#### 2.3.3.2 Frank-Wolfe Algorithm

Problem (2.19) can be optimized using the conditional gradient method, also known as the Frank-Wolfe algorithm (Frank and Wolfe, 1956). The advantage it provides is that there is no need to perform a projection step to remain in the feasible

[2]We refer the reader to Bertsekas and Tsitsiklis (1989) for the derivation of the update steps.

**Algorithm 2** Proximal method

**function** PROXIMAL_COMPUTE_BOUNDS(Problem (2.7))
  Initialize dual variables $\boldsymbol{\rho}^0$ via proposition 2
  $\hat{\mathbf{x}}^0 \in \operatorname{argmin}_{\mathbf{x},\hat{\mathbf{x}}} q(\boldsymbol{\rho}^0)$ as detailed in §2.3.2.1 and §2.3.2.2 ▷ initialize primals
  **for** $t \in [\![0, T-1]\!]$ **do**
    **for** $j \in [\![0, J-1]\!]$ **do** ▷ inner minimization loop
      **for** $k \in [\![0, n-1]\!]$ **do** ▷ block-coordinate loop
        $(\hat{\mathbf{x}}^j_{B,k}, \hat{\mathbf{x}}^j_{A,k+1}) \in \operatorname{argmin}_{\hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}} q(\boldsymbol{\rho}^t_k + \frac{\hat{\mathbf{x}}^t_{B,k} - \hat{\mathbf{x}}^t_{A,k}}{\eta_k})$ ▷ conditional gradient
        Compute layer optimal step size $\gamma^*_k$ using (2.21)
        $(\hat{\mathbf{x}}^t_{B,k}, \hat{\mathbf{x}}^t_{A,k+1}) \leftarrow \gamma^*_k(\hat{\mathbf{z}}^j_{B,k}, \hat{\mathbf{z}}^j_{A,k+1}) + (1-\gamma^*_k)(\hat{\mathbf{x}}^t_{B,k}, \hat{\mathbf{x}}^t_{A,k+1})$
    Compute $\boldsymbol{\rho}^{t+1}$ using equation (2.18) or (A.19) ▷ dual update
  **return** $q(\boldsymbol{\rho}^T)$

domain. Indeed, the different iterates remain in the feasible domain by construction as convex combination of points in the feasible domain. At each time step, we replace the objective by a linear approximation and optimize this linear function over the feasible domain to get an update direction, named conditional gradient. We then take a step in this direction. As the Augmented Lagrangian is smooth over the primal variables, there is no need to take the Frank-Wolfe step for all the network layers at once. We can in fact do it in a block-coordinate fashion, where a block is a network layer, with the goal of speeding up convergence.

**Conditional Gradient Computation** Let us denote iterations for the inner problem by the superscript $j$. Obtaining the conditional gradient requires minimizing a linearization of $\mathcal{L}(\hat{\mathbf{x}}, \boldsymbol{\rho})$ on the primal variables, restricted to the feasible domain:

$$\begin{aligned}\left(\mathbf{z}^j, \hat{\mathbf{z}}^j\right) &= \operatorname*{argmin}_{\mathbf{x},\hat{\mathbf{x}}} \nabla_{\hat{\mathbf{x}}} \mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}^t\right)^T \hat{\mathbf{x}} \\ \text{s.t.} \quad & \mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}), \\ & \mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}) \qquad k \in [\![1, n-1]\!].\end{aligned}$$

This computation corresponds exactly to the one we do to perform the inner minimization of problem (2.11) over $\mathbf{x}$ and $\hat{\mathbf{x}}$ in order to compute the supergradient (cf. §2.3.2.1, §2.3.2.2). To make this equivalence clearer, we point out that the linear coefficients of the primal variables will maintain the same form (with the difference that the dual variables are represented as their closed-form update for the following iteration), as $\nabla_{\hat{\mathbf{x}}_{B,k}} \mathcal{L}(\hat{\mathbf{x}}, \boldsymbol{\rho}^t) = \boldsymbol{\rho}^{t+1}_k$ and $\nabla_{\hat{\mathbf{x}}_{A,k+1}} \mathcal{L}(\hat{\mathbf{x}}, \boldsymbol{\rho}^t) = -\boldsymbol{\rho}^{t+1}_k$. The equivalence

of conditional gradient and supergradient is not particular to our problem. A more general description can be found in the work of Bach (2015).

**Block-Coordinate Steps** As the Augmented Lagrangian (2.19) is smooth in the primal variables, we can perform the Frank-Wolfe steps in a block-coordinate fashion (Lacoste-Julien et al., 2013). The conditional gradient computation decomposes over the subproblems (2.16), it is therefore natural to consider each $(\hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1})$ as a separate variable block. As the values of the primals at following layers are inter-dependent through the gradient of the Augmented Lagrangian, these block-coordinate updates will result in faster convergence. For each $k \in [\![0, n-1]\!]$, denoting again conditional gradients as $\hat{\mathbf{z}}$, the Frank-Wolfe steps for the $k$-th block take the following form:

$$(\hat{\mathbf{x}}^j_{B,k}, \hat{\mathbf{x}}^j_{A,k+1}) = \gamma_k(\hat{\mathbf{z}}^j_{B,k}, \hat{\mathbf{z}}^j_{A,k+1}) + (1-\gamma_k)(\hat{\mathbf{x}}^{j-1}_{B,k}, \hat{\mathbf{x}}^{j-1}_{A,k+1}). \tag{2.20}$$

Let us denote by $\mathbf{x}^j_{\gamma_k}$ a vector of all 0s, except for the $\hat{\mathbf{x}}_A$ and $\hat{\mathbf{x}}_B$ entries of the $k$-th block, which are set to equation (2.20). Due to the structure of problem (2.19), we can compute an optimal step size $\gamma^*_k$ by solving a one dimensional quadratic problem:

$$\gamma^*_k \in \underset{\gamma_k \in [0,1]}{\operatorname{argmin}} \mathcal{L}(\mathbf{x}^j_{\gamma_k}, \boldsymbol{\rho}^t).$$

Let us denote by $\text{Clip}_{[0,1]}$ an operator clipping a value into $[0,1]$, and let us cover corner cases through dummy assignments: $\eta_n = \infty$, and $\hat{\mathbf{x}}_{B,0} = \hat{\mathbf{z}}_{B,0} = 0$. Then, $\gamma^*_k$ is given by:

$$\gamma^*_k = \underset{[0,1]}{\text{Clip}} \left( \frac{\nabla_{\hat{\mathbf{x}}_{B,k}} \mathcal{L}\left(\hat{\mathbf{x}}^j, \boldsymbol{\rho}^t\right)^T (\hat{\mathbf{x}}^{j-1}_{B,k} - \hat{\mathbf{z}}^j_{B,k}) + \nabla_{\hat{\mathbf{x}}_{A,k+1}} \mathcal{L}\left(\hat{\mathbf{x}}^j, \boldsymbol{\rho}^t\right)^T (\hat{\mathbf{x}}^{j-1}_{A,k+1} - \hat{\mathbf{z}}^j_{A,k+1})}{\frac{1}{\eta_k} \left\| \hat{\mathbf{z}}^j_{B,k} - \hat{\mathbf{x}}^{j-1}_{B,k} \right\|^2 + \frac{1}{\eta_{k+1}} \left\| \hat{\mathbf{z}}^j_{A,k+1} - \hat{\mathbf{x}}^{j-1}_{A,k+1} \right\|^2} \right) \tag{2.21}$$

Finally, inspired by previous work on accelerating proximal methods (Lin et al., 2017; Salzo and Villa, 2012), we apply momentum on the dual updates to accelerate convergence; for details we refer the reader to appendix A.4.

### 2.3.4 Comparison to Previous Dual Problems

We conclude this section by comparing our dual problem (2.11) to the duals presented in §2.2.2, focusing on ReLU activation functions.

#### 2.3.4.1 Lagrangian Relaxation of the non-convex formulation

We first consider problem (2.6) by Dvijotham et al. (2018b). From the high level perspective, our decomposition considers larger subsets of constraints and hence results in a smaller number of dual variables to optimize over.

**Proposition 1.** *Assume* $\sigma_k(\hat{\mathbf{x}}_k) = \max(\mathbf{0}, \hat{\mathbf{x}}_k) \ \forall \ k \in [\![1, n-1]\!]$. *Then,* $\max_{\boldsymbol{\mu},\boldsymbol{\lambda}} d_O(\boldsymbol{\mu}, \boldsymbol{\lambda}) = \max_{\boldsymbol{\rho}} q(\boldsymbol{\rho})$, *that is, the dual solutions of problem* (2.11) *and problem* (2.6) *coincide. Additionally, both dual solutions coincide with the solution of problem* (2.7).

*Proof.* Recall that the ReLU version of problem (2.7) is an LP (see (2.8)). Due to linear programming duality (Lemaréchal, 2001), $\max_{\boldsymbol{\rho}} q(\boldsymbol{\rho}) = p^*$, where $p^*$ denotes the solution of problem (2.7). Moreover, Theorem 2 by Dvijotham et al. (2018b) shows that problem (2.6) corresponds to the Lagrangian dual of problem (2.7). Therefore, invoking linear programming duality again, $\max_{\boldsymbol{\mu},\boldsymbol{\lambda}} d_O(\boldsymbol{\mu}, \boldsymbol{\lambda}) = \max_{\boldsymbol{\rho}} q(\boldsymbol{\rho}) = p^*$. □

While proposition 1 states that problems (2.11) and (2.6) will yield the same bounds at optimality, this does not imply that the two derivations yield solvers with the same efficiency. In fact, we will next prove that, for ReLU-based networks, our formulation dominates problem (2.6), producing bounds at least as tight based on the same dual variables. In fact, problem (2.11) operates on a subset of the dual space of problem (2.6) that always contains the dual optimum.

**Theorem 1.** *Let us assume* $\sigma_k(\hat{\mathbf{x}}_k) = \max(\mathbf{0}, \hat{\mathbf{x}}_k) \ \forall \ k \in [\![1, n-1]\!]$. *For dual point* $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ *of problem* (2.6) *by Dvijotham et al. (2018b) yielding bound* $d(\boldsymbol{\mu}, \boldsymbol{\lambda})$, *it holds that* $q(\boldsymbol{\mu}) \geq d(\boldsymbol{\mu}, \boldsymbol{\lambda})$. *Furthermore, if* $\boldsymbol{\lambda}'_{n-1} = -W_n^T \mathbf{1}$ *and* $\boldsymbol{\lambda}'_{k-1} = W_k^T \boldsymbol{\mu}_k$ *for* $k \in [\![2, n-1]\!]$, *then* $q(\boldsymbol{\mu}) = d(\boldsymbol{\mu}, \boldsymbol{\lambda}')$.

*Proof.* See appendix A.1. □

Theorem 1 motivates the use of dual (2.11) over the general form of problem (2.6). Moreover, it shows that, for ReLU activations, this application of Lagrangian

Decomposition coincides with a modified version of problem (2.6) with additional equality constraints. Such a modification can be also found in the work by Salman et al. (2019b, appendix G.1). However, its advantages for iterative bounding algorithms and connections to Lagrangian Decomposition were not investigated in their work.

#### 2.3.4.2 Propagation-based methods

We now turn our attention to propagation-based methods (see §2.2.2.1).

**Proposition 2.** *Let $\bar{d}_P$ be a lower bound to problem (2.1) obtained via a propagation-based bounding algorithm. Then, if $\sigma_k(\hat{\mathbf{x}}_k) = \max(\mathbf{0}, \hat{\mathbf{x}}_k)\ \forall\ k \in [\![1, n-1]\!]$, there exist some dual points $\bar{\boldsymbol{\rho}}$ and $\left(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}\right)$ such that $q(\bar{\boldsymbol{\rho}}) = d_O\left(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}\right) = \bar{d}_P$, and both $\bar{\boldsymbol{\rho}}$ and $\left(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}\right)$ can be computed at the cost of a backward pass through the network.*

*Proof.* See appendix A.2. □

Proposition 2 shows that both problem (2.6) and problem (2.11) can be inexpensively initialized via propagation-based algorithms such as CROWN (Zhang et al., 2018) or the one by Wong and Kolter (2018). We exploit this result in our computational evaluation (§2.7, §2.8).

## 2.4 Better Branching

In this section, we present a novel branching strategy aimed at branch and bound for neural network verification.

### 2.4.1 Preliminaries: Approximations of Strong Branching

Recall that branch and bound discards subproblems when the available lower bound on their minimum becomes positive (see §2.2.3). Let us denote the employed bounding algorithm by $\mathcal{A}$, and its lower bound for subproblem $p$ as $l_{\mathcal{A}}(p)$. Moreover, let $c_{d\{h\}}(p)$ be the $h$-th children of $p$ according to branching decision $d$. Ideally, we would like to take the branching decision that maximizes the chances that some $c_{d\{h\}}(p)$ is discarded, in order to minimize the size of the branch and bound

tree. In order to do so, we could compute $l_{\mathcal{A}}(c_{d\{h\}}(p))$ according to every possible branching decision $d$ and each relative child $h$. In the context of branch and bound for integer programming, this branching strategy is traditionally referred to as *full strong branching* (Morrison et al., 2016). As full strong branching is impractical on the large search spaces encountered in neural network verification, it is usually replaced by an approximation. For branching based on input domain splitting, each branching decision $d$ corresponds to an index of the input space, that is: $d := i \in \mathbb{R}^{n_0}$. In this context, Bunel et al. (2018) replace $\mathcal{A}$ (the bounding algorithm used for subproblem lower bounds[3]) by a looser yet inexpensive method. Specifically, they rely on the bounding algorithm by Wong and Kolter (2018), denoted WK, which is a propagation-based method for ReLUs, where $\bar{\sigma}_k(\hat{\mathbf{x}}_k) = \frac{[\hat{\mathbf{u}}_k]_+ \odot (\hat{\mathbf{x}}_k - [\hat{\mathbf{l}}_k]_-)}{[\hat{\mathbf{u}}_k]_+ - [\hat{\mathbf{l}}_k]_-}$, and $\underline{\sigma}_k(\hat{\mathbf{x}}_k) = \frac{[\hat{\mathbf{u}}_k]_+ \odot \hat{\mathbf{x}}_k}{[\hat{\mathbf{u}}_k]_+ - [\hat{\mathbf{l}}_k]_-}$ (see §2.2.2.1). The resulting branching strategy, termed Smart Branching (SB) by Bunel et al. (2018), branches on the input coordinate $i$ such that:

$$i \in \underset{i \in \mathbb{R}^{n_0}}{\operatorname{argmax}} \left( \min_{h \in [\![1,2]\!]} \left\{ l_{\mathrm{WK}}(c_{i\{h\}}(p)) \right\} \right).$$

However, input-based branching was found to be ineffective on large convolutional networks (Bunel et al., 2020b). With this in mind, in BaBSR, Bunel et al. (2020b) rely on a branching strategy that operates by splitting a ReLU into its two linear phases (see §2.2.3.2). The original SB heuristic is unsuitable for ReLU branching, as it requires a number of backward passes linear in the size of the space of branching decisions: in general, $\left(\sum_{k=1}^{n-1} n_k\right) \gg n_0$. Therefore, Smart ReLU (SR) branching, the heuristic adopted within BaBSR, approximates strong branching even further. At the cost of a single backward pass, it assigns scores $\mathbf{s}_{\mathrm{SR}}$ and $\mathbf{t}_{\mathrm{SR}}$ to all possible branching decisions (that is, to each ReLU):

$$\begin{aligned}
\bar{\boldsymbol{\lambda}}_{n-1} &= -W_n^T \mathbf{1}, \qquad \bar{\boldsymbol{\lambda}}_{k-1} = W_k^T \left( \frac{[\hat{\mathbf{u}}_k]_+}{[\hat{\mathbf{u}}_k]_+ - [\hat{\mathbf{l}}_k]_-} \odot \bar{\boldsymbol{\lambda}}_k \right) \quad k \in [\![2, n-1]\!], \\
\mathbf{s}_{\mathrm{SR},k} &= \left| \begin{array}{l} \max\left\{0, \bar{\boldsymbol{\lambda}}_k \odot \mathbf{b}_k\right\} \\ -\frac{\hat{\mathbf{u}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} \odot \bar{\boldsymbol{\lambda}}_k \odot \mathbf{b}_k + \frac{\hat{\mathbf{u}}_k \odot \hat{\mathbf{l}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} \odot [\bar{\boldsymbol{\lambda}}_k]_+ \end{array} \right| \odot \mathbb{1}_{\hat{\mathbf{l}}_k < 0, \hat{\mathbf{u}}_k > 0} \quad k \in [\![1, n-1]\!], \\
\mathbf{t}_{\mathrm{SR},k} &= \frac{-\hat{\mathbf{u}}_k \odot \hat{\mathbf{l}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} \odot [\bar{\boldsymbol{\lambda}}_k]_+ \odot \mathbb{1}_{\hat{\mathbf{l}}_k < 0, \hat{\mathbf{u}}_k > 0} \qquad k \in [\![1, n-1]\!].
\end{aligned} \tag{2.22}$$

[3] in the case of Bunel et al. (2018), this means solving the LP in problem (2.7).

Then, SR branches on the ReLU having the largest $\mathbf{s}_{\mathrm{SR},k}$ scores or, if such a ReLU's score is below a threshold, the largest backup scores $\mathbf{t}_{\mathrm{SR},k}$. This strategy can be seen as an approximation of SB. In fact, scores $\mathbf{s}_{\mathrm{SR},k}$ approximate the change in the bounds by Wong and Kolter (2018) that would arise from splitting on the ambiguous ReLUs at layer $k$. In more detail, they consider the effect of splitting within equation (2.5), without backpropagating the effect on $\bar{\boldsymbol{\lambda}}_j$ for $j \in [\![1, k-1]\!]$ via equation (2.4). The two arguments of the maximum in equation (2.22) correspond to the blocking and passing cases, whereas the remaining terms represent the ambiguous case. Backup scores $\mathbf{t}_{\mathrm{SR},k}$, instead, correspond to the product of the Lagrangian multiplier for $\mathbf{x}_k \leq \bar{\sigma}_k(\hat{\mathbf{x}}_k)$, and the maximum distance of $\bar{\sigma}_k(\hat{\mathbf{x}}_k)$ from $\sigma(\hat{\mathbf{x}}_k)$ (in fact, $\bar{\mathbf{b}}_k = \frac{-\hat{\mathbf{u}}_k \odot \hat{\mathbf{l}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k}$). They hence provide a second estimation of the effect that a given ReLU split, through its associated reduction of the feasible space, would have on bounding.

### 2.4.2 Filtered Smart Branching

We now present *Filtered Smart Branching* (FSB), our novel strategy for activation splitting. Bunel et al. (2020b) show that, in spite of its rougher approximation of strong branching, SR significantly outperforms SB on larger networks. Therefore, it is natural to investigate whether improving SR's approximation quality would further reduce the size of the branch and bound tree. First, inspired by SB, we replace the maximization within $\mathbf{s}_{\mathrm{SR},k}$ with a minimization. Keeping $\bar{\boldsymbol{\lambda}}$ as in equation (2.22), we obtain:

$$\mathbf{s}_{\mathrm{FSB},k} = \left| \begin{array}{l} \min\left\{0, \bar{\boldsymbol{\lambda}}_k \odot \mathbf{b}_k\right\} \\ -\frac{\hat{\mathbf{u}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} \odot \bar{\boldsymbol{\lambda}}_k \odot \mathbf{b}_k + \frac{\hat{\mathbf{u}}_k \odot \hat{\mathbf{l}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} \odot [\bar{\boldsymbol{\lambda}}_k]_+ \end{array} \right| \odot \mathbb{1}_{\hat{\mathbf{l}}_k < 0, \hat{\mathbf{u}}_k > 0} \quad k \in [\![1, n-1]\!]\,. \tag{2.23}$$

Compared to $\mathbf{s}_{\mathrm{SR},k}$, $\mathbf{s}_{\mathrm{FSB},k}$ is designed to balance the branch and bound tree, prioritizing branching decisions that yield bounding improvements in both children, rather than one of them. As for $\mathbf{s}_{\mathrm{SR},k}$, the $\mathbf{s}_{\mathrm{FSB},k}$ scores owe their computational efficiency to their shortsightedness and can be computed at the cost of a single gradient backpropagation. However, considering that our bounding algorithms (§2.3) require multiple forward/backward passes over the network, we can afford to marginally increase the branching cost if doing so benefits the quality of split.

We propose a layered approach: we employ scores $\mathbf{s}_{\text{FSB},k}$ and $\mathbf{t}_{\text{SR},k}$ to select the most promising candidate branching choices for each layer. Denoting a branching choice by a pair $(k,i) \in [\![1, n-1]\!] \times \mathbb{R}^{n_k}$, we create a set of $O(n)$ candidate choices $D_{\text{FSB}} := \cup_k D_{\text{FSB},k}$, where:

$$D_{\text{FSB},k} = \left\{ \left(k, \operatorname*{argmax}_{i \in \mathbb{R}^{n_k}} \mathbf{s}_{\text{FSB},k}[i]\right), \left(k, \operatorname*{argmax}_{i \in \mathbb{R}^{n_k}} \mathbf{t}_{\text{SR},k}[i]\right) \right\} \quad k \in [\![1, n-1]\!]. \quad (2.24)$$

As, in general, $O(n) < \left(\sum_{k=1}^{n-1} n_k\right)$, we can then afford to compute lower bounds for each of the candidates using a fast dual bounding algorithm $\mathcal{A}_{\text{FSB}}$. In our implementation, $\mathcal{A}_{\text{FSB}}$ returns the tightest bounds between CROWN[4] (Zhang et al., 2018) and the algorithm by Wong and Kolter (2018). FSB splits on the activation determined by:

$$d_{\text{FSB}} \in \operatorname*{argmax}_{d \in D_{\text{FSB}}} \left( \min_{h \in [\![1,2]\!]} \left\{ l_{\mathcal{A}_{\text{FSB}}}(c_{d\{h\}}(p)) \right\} \right), \quad (2.25)$$

where the $D_{\text{FSB}}$ candidate set is determined via equation (2.24). FSB is both conceptually simple and effective in practice. In fact, we will show in section 2.8 that FSB significantly improves on SR (Bunel et al., 2020b). Moreover, it is strongly competitive with a strategy that mimics strong branching via learning algorithms (Lu and Kumar, 2020), without its training costs. Finally, we point out that, while FSB was presented in the context of ReLU branching, the technique generalizes to other activation functions. Let us divide an activation's domain into intervals: $[\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] = \cup_j [\hat{\mathbf{l}}_k^j, \hat{\mathbf{u}}_k^j]$. It is convenient to branch on the intervals if $\text{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ satisfies:

$$\cup_j \left\{ \text{Conv}(\sigma_k, \hat{\mathbf{l}}_k^j, \hat{\mathbf{u}}_k^j) \right\} \subset \text{Conv}(\cup_j \left\{ \text{Conv}(\sigma_k, \hat{\mathbf{l}}_k^j, \hat{\mathbf{u}}_k^j) \right\}) = \text{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k). \quad (2.26)$$

Then, in order to apply FSB, it suffices to: (i) adapt $\mathbf{s}_{\text{FSB},k}$ and $\mathbf{t}_{\text{SR},k}$ to the chosen activation's linear bounding functions $\bar{\sigma}_k(\hat{\mathbf{x}}_k)$ and $\underline{\sigma}_k(\hat{\mathbf{x}}_k)$, (ii) choose an appropriate bounding algorithm $\mathcal{A}_{FSB}$. For instance, the work by Zhang et al. (2018) provides suitable $\bar{\sigma}_k(\hat{\mathbf{x}}_k)$, $\underline{\sigma}_k(\hat{\mathbf{x}}_k)$ and $\mathcal{A}_{FSB}$ for both the hyperbolic tangent and sigmoid, which satisfy equation (2.26).

[4] As mentioned in §2.2.2.1, CROWN is a propagation-based bounding algorithm. In particular, for ReLU activations, it employs $\bar{\sigma}_k(\hat{\mathbf{x}}_k) = \frac{[\hat{\mathbf{u}}_k]_+ \odot (\hat{\mathbf{x}}_k - [\hat{\mathbf{l}}_k]_-)}{[\hat{\mathbf{u}}_k]_+ - [\hat{\mathbf{l}}_k]_-}$ and $\underline{\sigma}_k(\hat{\mathbf{x}}_k) = \left( \mathbb{1}_{-\hat{\mathbf{l}}_k \leq \hat{\mathbf{u}}_k} + \mathbb{1}_{\hat{\mathbf{l}}_k \geq 0} \right) \hat{\mathbf{x}}_k$.

## 2.5 Improved Branch and Bound

Having presented the building blocks of our branch and bound framework (§2.3, §2.4), we now present further technical and implementation details.

### 2.5.1 Additional Branch and Bound Improvements

This section presents the remaining details for Branch and Dual Network Bound (BaDNB), our branch and bound framework for neural network verification designed around dual bounding algorithms (§2.3) and Filtered Smart Branching (§2.4). We start from the treatment of intermediate bounds (§2.5.1.1), then present a simple heuristic to dynamically adapt the bounding tightness within the branch and bound tree (§2.5.1.2), and describe how to obtain upper bounds on the minimum of each subproblem (§2.5.1.3).

#### 2.5.1.1 Intermediate Bounds

Branching on a ReLU at layer $k$ will potentially influence all $\hat{\mathbf{l}}_j$ and $\hat{\mathbf{u}}_j$ for $j \in [\![k+1, n-1]\!]$. Therefore, BaBSR by Bunel et al. (2020b) updates the relevant intermediate bounds after every branching decision, leading to $\sum_{k=2}^{n-1} 2n_k$ bounding computations per subproblem in the worst case (that is, when the branching is performed on the first layer). For medium-sized convolutional networks, $\sum_{k=2}^{n-1} 2n_k$ will be in the order of thousands. In order to compensate for the large computational expense, BaBSR relies on relatively loose bounding algorithms for intermediate bounds, taking the layer-wise best bounds between the method by Wong and Kolter (2018) and Interval Bound Propagation (Mirman et al., 2018).

Dual bounding algorithms such as ours (§2.3) or the one by Dvijotham et al. (2018b) are significantly less expensive than solving the convex hull LP (2.7) to optimality. Nevertheless, the use of dual iterative algorithms for intermediate bounds would be the bottleneck of each branch and bound iteration. In order to tighten intermediate bounds without incurring significant expenses, and considering its remarkable performance at the cost of a single backward pass (for instance, see §2.7), we propose to employ CROWN (Zhang et al., 2018). In particular, we use

the layer-wise best bounds between CROWN and the method by Wong and Kolter (2018). Furthermore, as BaDNB employs cheaper last layer bounding than BaBSR, even inexpensive intermediate bounding can make up a large portion of a branch and bound iteration's runtime. Therefore, in order to maximize the number of visited nodes within a given time, we only compute intermediate bounds at the root of the branch and bound tree. In §2.8, we will show that, while it sacrifices the tightness of the bounds, such a choice pays off experimentally.

#### 2.5.1.2 Dynamically Adjusting the Tightness of Bounds

BaDNB relies on dual bounding algorithms, whose advantage over black-box LP solvers is to quickly reach close-to-optimal bounds (see §2.7). In addition, they allow for massively parallel implementations, which we exploit by computing bounds for a batch of branch and bound subproblems at once (§2.5.2). However, the level of tightness required for a batch of possible heterogeneous subproblems is not clear in advance. Due to the structure of duals (2.6) and (2.11), the bounding improvement per iteration will decrease as the solver approaches optimality, both in theory and in practice. Therefore, a straightforward solution is to choose a fixed number iterations near the "knee point" of the curve plotting bounds over iterations for the root of the branch and bound tree. However, a similar "diminishing returns" law holds for the bounding improvement caused by branching as one moves deeper in the branch and bound tree. Therefore, at some depth in the tree, it will be more convenient to invest the computational resources in tighter bounding rather than branching. In order to take this into account, we devise a simple heuristic to dynamically adjust tightness for last layer bounding.

Let us again denote by $\mathcal{A}$ the bounding algorithm used for the subproblem lower bounds, adding a subscript to indicate the employed number of iterations. We denote by $t(\mathcal{A}_T)$ the cost of running $\mathcal{A}$ with $T$ iterations. We start from a relatively inexpensive setting $\mathcal{A}_{T_0}$, and choose $m$ different speed-accuracy trade-offs: $\mathcal{A}_{T_0}, \mathcal{A}_{T_1}, \ldots, \mathcal{A}_{T_{m-1}}$, with $t(\mathcal{A}_{T_j}) < t(\mathcal{A}_{T_{j+1}})$ for $j \in [\![0, m-2]\!]$. Let us denote by $c^{-1}(p)$ the parent of subproblem $p$ and keep an exponential moving average $i(p)$ of the

lower bound improvement from parent to child: $i(p) := \alpha \left( l_{\mathcal{A}_{T_j}}(p) - l_{\mathcal{A}_{T_j}}(c^{-1}(p)) \right) + (1-\alpha) i(c^{-1}(p))$, where $\alpha \in [0,1]$. Furthermore, let us estimate the tightness of the bounds given by each $\mathcal{A}_{T_j}$ on some test subproblem[5] $r$. We update the bounding algorithm from $\mathcal{A}_{T_j}$ to $\mathcal{A}_{T_{j+1}}$ when the following condition is satisfied:

$$i(p_{\min}) < \left( l_{\mathcal{A}_{T_{j+1}}}(r) - l_{\mathcal{A}_{T_j}}(r) \right) \frac{t(\mathcal{A}_{T_j})}{t(\mathcal{A}_{T_{j+1}})},$$

where $p_{\min}$ denotes the subproblem with the smallest lower bound within the current subproblem batch. In other words, we increase the number of iterations when an estimation of the associated bounds tightening, normalized by its runtime overhead, exceeds the current expected branching improvement.

#### 2.5.1.3 Upper Bounds

Similarly to BaBSR (Bunel et al., 2020b), we compute an upper bound on the minimum of the current subproblem by evaluating the network at an input point $\mathbf{x}_0$ from the lower bound computation. BaBSR's use of LP solvers allows them to evaluate the network at the primal optimal solution of problem (2.7). However, as explained in §2.5.1.2, in general BaDNB will not run the dual iterative algorithms presented in §2.3 to convergence. Therefore, we will evaluate the network at some feasible yet suboptimal $\mathbf{x}_0$. In practice, for supergradient-type methods like algorithm 1, we evaluate the network at the last computed inner minimizer from problem (2.13). For Algorithm 2, instead, we evaluate the network at the last $\mathbf{x}_0$ found while optimizing problem (2.19).

### 2.5.2 Implementation Details

The calculations involved in the various components of our branch and bound framework correspond to standard linear algebra operations commonly employed during the forward and backward passes of neural networks. For instance, operations of the form $\hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1}$ are exactly forward passes of the network, while operations like $\boldsymbol{\rho}_{k+1}^T W_{k+1}$ are analogous to the backpropagation of gradients. This

[5] In our implementation, in order to capture the effect of activation splits (see §2.4), we set $r$ to the first encountered subproblem at a depth of 4 in the branch and bound tree.

makes it possible for us to leverage the engineering efforts made to enable fast training and evaluation of neural networks, and easily take advantage of GPU accelerations. As an example, when dealing with convolutional layers, we can employ specialized implementations rather than building the equivalent $W_k$ matrix, which would contain a lot of redundancy.

#### 2.5.2.1 Bounding Algorithms

We implement propagation-based algorithms (§2.2.2.1), the algorithm by Dvijotham et al. (2018b), and our methods based on Lagrangian Decomposition (§2.3) within a unified framework, exploiting their common building blocks. One of the benefits of these dual bounding algorithms is that they are easily parallelizable. In fact, when computing the upper and lower bounds for all the neurons of a layer, there is no dependency between the different problems, so we are free to solve them all simultaneously in a batched mode. The approach closely mirrors the batching over samples commonly employed for training neural networks.

#### 2.5.2.2 Branch and Bound

For complete verification, the use of branch and bound opens up yet another stream of parallelism. In fact, it is possible to batch over subproblems as well, for both branching and bounding. In detail, a batch is formed by the $B$ subproblems having the lowest lower bound (in §2.8, $B$ ranges from 100 to 1600, depending on the given experiment). We first compute and execute branching decisions for the batch, then move on to the batch of children, whose size is $2B$ (the branching is binary for ReLU activations). Then, for the branch and bound specifications which require it, intermediate bounds are updated for the entire batch, parallelizing both over the $2B$ subproblems and the neurons of a layer (leading to up to 45000 bounding computations at once, in our experiments). Finally, we compute lower bounds for the $2B$ subproblems, and get upper bounds as detailed in §2.5.1.3.

## 2.6 Related Work

The work by Bunel et al. (2018, 2020b) presented an unified view of neural network verification, providing an interpretation of state-of-the-art complete verification methods as branch and bound algorithms (see §2.2.3). Such interpretation holds for a wide array of approaches, including SMT solvers (Ehlers, 2017; Katz et al., 2017), Mixed Integer Programming (MIP) formulations (Tjeng et al., 2019), ReLUVal and Neurify (Wang et al., 2018a,b). By presenting modifications to the search strategy, the bounding process and the branching algorithm, the methods by Bunel et al. (2020b) outperform previous complete verifiers by a significant margin, on a variety of standard datasets such as those from Ehlers (2017); Katz et al. (2017). Therefore, building upon its success, we started from the framework by Bunel et al. (2020b) and presented various improvements to improve its scaling capabilities.

While many of our contributions (§2.4, §2.5) are to be employed within branch and bound, bounding algorithms (§2.3) can be additionally seen as stand-alone incomplete verifiers (see §2.2). Although they cannot verify properties for all problem instances, incomplete verifiers scale significantly better, as they trade speed for completeness. So far, we have focused on approaches presenting a dual interpretation. However, incomplete verifiers can be more generally described as solvers for relaxations of problem (2.1). In fact, explicitly or implicitly, these are all equivalent to propagating a convex domain through the network to over-approximate the set of reachable values. Some approaches (Ehlers, 2017; Salman et al., 2019b) rely on off-the-shelf solvers to solve accurate relaxations such as Planet (equation (2.8)) (Ehlers, 2017), which is the best known linear-sized approximation of the problem. On the other hand, as Planet and other more complex relaxations do not have closed form solutions, some researchers have also proposed easier to solve, looser formulations. Many of these fall into the category of propagation-based methods (§2.2.2.1) (Wong and Kolter, 2018; Weng et al., 2018; Singh et al., 2018; Zhang et al., 2018; Singh et al., 2019b), which solve linear relaxations with only two constraints per activation function, yielding large yet inexpensive over-approximations. Others relaxed the problem even further in order to obtain faster

solutions, either by propagating intervals (Gowal et al., 2018b), or through abstract interpretation (Mirman et al., 2018). Our bounding algorithms and the one by Dvijotham et al. (2018b) are custom dual solvers for the convex hull of the element-wise activation function (Planet, in the ReLU case). Finally we point out that, while tighter convex relaxations exist, they involve a quadratic number of variables or exponentially many constraints. The semi-definite programming method of Raghunathan et al. (2018), or the relaxation by Anderson et al. (2020), obtained from relaxing strong Mixed Integer Programming formulations, fall in this category. We do not address them here.

## 2.7 Incomplete Verification Experiments

In this section, we test the speed-accuracy trade-offs of our bounding algorithms in an incomplete verification setting. In particular, we compare them with various bounding algorithms on an adversarial robustness task, for images of the CIFAR-10 test set (Krizhevsky, 2009).

### 2.7.1 Experimental Setup

For each test image, we compute an upper bound on the vulnerability of a network to each possible misclassification. In other words, we upper bound the difference between the 9 logits associated to incorrect classes, and the ground truth logit, under an allowed perturbation $\epsilon_{\text{ver}}$ in infinity norm of the inputs. If for any class the upper bound on the difference is negative, then we are certain that the network is robust against that adversarial perturbation. We employ a ReLU-based convolutional network used by Wong and Kolter (2018) and whose structure corresponds to the “Wide" architecture in table 2.1. We train the network against perturbations of a size up to $\epsilon_{\text{train}} = 2/255$ in $\ell_\infty$ norm, and test for adversarial vulnerability on $\epsilon_{\text{ver}} = 2.7/255$. Adversarial training is performed via the method by Madry et al. (2018), based on an attacker using 50 steps of projected gradient descent to obtain the samples. Additional experiments for a network trained using standard

stochastic gradient descent and cross entropy, with no robustness-related term in the objective, can be found in appendix A.5.

### 2.7.2 Bounding Algorithms

We consider the following bounding algorithms:

- **IBP** Interval Bound Propagation (Gowal et al., 2018b; Mirman et al., 2018), whose bounds correspond to setting all dual variables to 0 in dual problems (2.6) and (2.11).
- **WK** and **CROWN**, the propagation-based methods by respectively Wong and Kolter (2018) and Zhang et al. (2018). Exploiting proposition 2, the bounds by both algorithms correspond to a specific dual assignment for both problem (2.6) and problem (2.11).
- **DSG+** uses supergradient methods on dual (2.6), the method by Dvijotham et al. (2018b). We use the Adam (Kingma and Ba, 2015) updates rules and decrease the step size linearly between two values, similarly to the experiments of Dvijotham et al. (2018b). We experimented with other step size schedules, like constant step size or $\frac{1}{t}$ schedules, which all performed worse.
- **Dec-DSG+** is a direct application of Theorem 1: it obtains a dual point $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ by optimizing problem (2.6) via DSG+ and then evaluates $q(\boldsymbol{\mu})$ in problem (2.11) to obtain the final bounds.
- **Supergradient** is the first of the two solvers presented (§2.3.2), using a supergradient method on problem (2.11). As for DSG+, we use Adam updates and linearly decay the step size.
- **Proximal** is the solver presented in §2.3.3, performing proximal maximization on problem (2.11). We use a small fixed number of iterations for the inner problems (specifically, we set $J = 2$ in algorithm 2).

- **Gurobi** is our gold standard method. It employs the commercial black box solver Gurobi to solve problem (2.7) to optimality. We make use of LP incrementalism (warm-starting): as the experiment involves computing 9 different output upper bounds, we warm-start each LP from the LP of the previous neuron.

- **Gurobi-TL** is the time-limited version of the above, which stops at the first dual simplex iteration for which the total elapsed time exceeded that required by 400 iterations of the proximal method.

Exploiting proposition 2, all dual iterative algorithms (Proximal, Supergradient, and DSG+) are initialized from CROWN, which usually outperforms other propagation-based algorithms on ReLU networks (Zhang et al., 2018). In all cases, we pre-compute intermediate bounds (see §2.2.2.3) using the layer-wise best amongst

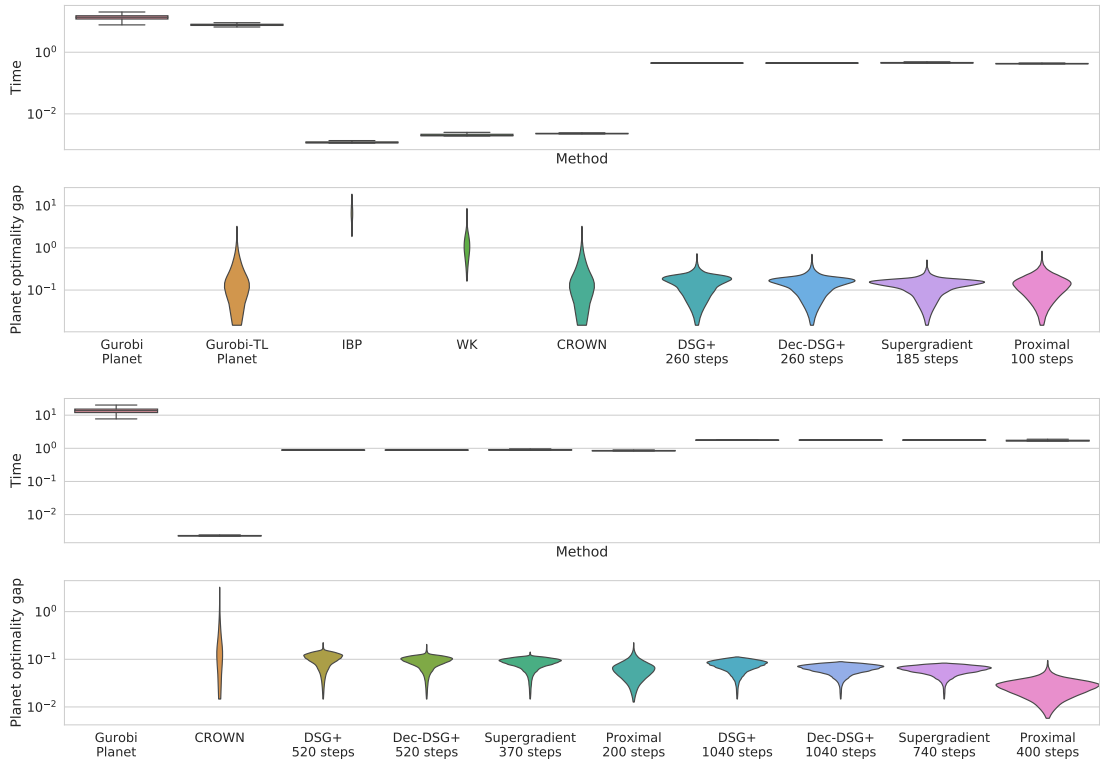



**Figure 2.2:** Distribution of runtime and gap to optimality on a network adversarially trained with the method by Madry et al. (2018), on CIFAR-10 images. In both cases, lower is better. The width at a given value represents the proportion of problems for which this is the result. Gurobi Planet always returns the optimal solution to problem (2.7), at a large computational cost.

CROWN and WK. This reflects the bounding schemes used within branch and bound for networks of comparable size (Bunel et al., 2020b; Lu and Kumar, 2020). Hyper-parameters were tuned on a small subset of the CIFAR-10 test set. For both supergradient methods (Supergradient, and DSG+), we decrease the step size linearly from $10^{-2}$ to $10^{-4}$. For Proximal, we employ momentum coefficient $\mu = 0.3$ (see appendix A.4) and, for all layers, linearly increase the weight of the proximal terms from $10^1$ to $5 \times 10^2$. Because of their small cost per iteration, dual iterative methods allow the user to choose amongst a variety of trade-offs between tightness and speed. In order to perform a fair comparison, we fixed the number of iterations for the various methods so that each of them would take the same average time,. This was done by tuning the iteration ratios on a subset of the images. We report results for three different computational budgets. Note that the Lagrangian Decomposition has a higher cost per iteration due to its more complex primal feasible set. The cost of the proximal method is even larger, as it requires an iterative procedure for the inner minimization (2.19). All methods were implemented in PyTorch (Paszke et al., 2019) and run on a single Nvidia Titan V GPU, except those based on Gurobi, which were run on 4 cores of i9-7900X CPUs. The amenability of dual methods do GPU acceleration is a big part of their advantages over off-the-shelf solvers. Experiments were run under Ubuntu 16.04.2 LTS.

### 2.7.3 Results

We measure the time to compute last layer bounds, and report the gap to the optimal solution of problem (2.7), which is based on the Planet relaxation (2.8) for our ReLU benchmark. Figure 2.2 presents the distribution of results for all bounding algorithms. The fastest method is by IP, which requires only a few linear algebra operations over the last network layer. However, the bounds it returns are consistently very loose. WK and CROWN have a similarly low computational cost: they both require a single backward pass through the network per optimization problem. Nevertheless, CROWN generates much tighter average bounds, and is therefore the best candidate for the initialization of dual iterative algorithms. At

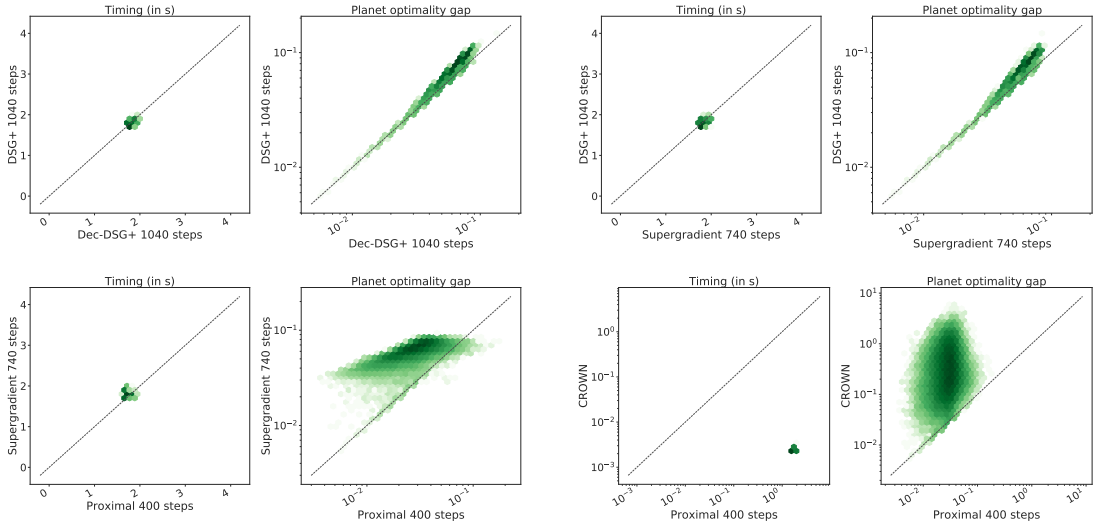



**Figure 2.3:** Pointwise comparison for a subset of the methods on the data presented in Figure 2.2. Each datapoint corresponds to a CIFAR image, darker color shades mean higher point density in a logarithmic scale. The dotted line corresponds to the equality and in both graphs, lower is better along both axes.

the opposite end of the spectrum, Gurobi is extremely slow but provides the best achievable bounds. Furthermore, time-limiting the LP solver significantly worsens the produced bounds without a noticeable cut in runtimes. This is due to the high cost per iteration of the dual simplex algorithm. For DSG+, Supergradient and Proximal, the improved quality of the bounds as compared to IP, WK and CROWN shows that there are benefits in actually solving the relaxation rather than relying on approximations. In particular, a few iterations of the iterative algorithms cuts away the looser part of CROWN's bounds distribution, and an increased computational budget leads to significantly better average bounds. While the relative cost of dual solvers over CROWN (three order of magnitudes more time) might seem disproportionate, we will see in §2.8.2 that the gain in tightness is crucial for faster complete verification. Furthermore, by looking at the point-wise comparisons in Figure 2.3, we can see that Supergradient yields consistently better bounds than DSG+. As both methods employ the Adam update rule (and the same hyper-parameters, which are optimal for both), we can conclude that operating on the Lagrangian Decomposition dual (2.1) produces better speed-accuracy trade-offs compared to the dual (2.6) by Dvijotham et al. (2018b). This is in line with the

| Network Specifications | | Verification Properties | Network Architecture |
|---|---|---|---|
| Name: | Base | 100 properties | Conv2d(3,8,4, stride=2, padding=1) |
| Activation: | ReLU | | Conv2d(8,16,4, stride=2, padding=1) |
| Training method: | WK | $\epsilon_{\text{ver}} \in \left[\frac{3.5}{255}, \frac{14.5}{255}\right]$ | linear layer of 100 hidden units |
| Total activations: | 3172 | | linear layer of 10 hidden units |
| Name: | Wide | 100 properties | Conv2d(3,16,4, stride=2, padding=1) |
| Activation: | ReLU | | Conv2d(16,32,4, stride=2, padding=1) |
| Training method: | WK | $\epsilon_{\text{ver}} \in \left[\frac{3.5}{255}, \frac{12.4}{255}\right]$ | linear layer of 100 hidden units |
| Total activations: | 6244 | | linear layer of 10 hidden units |
| Name: | Deep | 100 properties | Conv2d(3,8,4, stride=2, padding=1) |
| Activation: | ReLU | | Conv2d(8,8,3, stride=1, padding=1) |
| Training method: | WK | $\epsilon_{\text{ver}} \in \left[\frac{3.5}{255}, \frac{14.9}{255}\right]$ | Conv2d(8,8,3, stride=1, padding=1) |
| Total activations: | 6756 | | Conv2d(8,8,4, stride=2, padding=1) |
| | | | linear layer of 100 hidden units |
| | | | linear layer of 10 hidden units |

**Table 2.1:** Specifications of the OVAL dataset, a subset of the CIFAR-10 complete verification dataset from Lu and Kumar (2020). Each property is associated to a different $\epsilon_{\text{ver}}$ value. WK denotes the verified training algorithm by Wong and Kolter (2018).

expectations from Theorem 1. Moreover, while a direct application of the Theorem (Dec-DSG+) does improve on the DSG+ bounds, operating on the Decomposition dual (2.1) is empirically more effective. Finally, on average, the Proximal yields better bounds than those returned by Supergradient, further improving on the DSG+ baseline. In particular, we stress that the support of optimality gap distribution is larger for Proximal, with a heavier tail towards better bounds.

## 2.8 Complete Verification

We now evaluate our branch and bound framework and its building blocks within complete verification. As detailed in §2.8.1, we focus on proving (or disproving) a network's adversarial robustness. We first study the effect of each presented branch and bound component (§2.8.2, §2.8.3, §2.8.4), then compare BaDNB with state-of-the-art complete verifiers in §2.8.5.

### 2.8.1 Experimental Setup

Tables 2.1 and 2.2 present the details of the two verification datasets on which we conduct our experimental evaluation. For both, the goal is to verify whether a

network is robust to $\ell_\infty$ norm perturbations of radius $\epsilon_{\text{ver}}$ on images of the CIFAR-10 (Krizhevsky, 2009) test set. For each complete verifier, we measure the time to termination, limited at one hour. In case of timeouts, the runtime is replaced by such time limit. All experiments were run under Ubuntu 16.04.2 LTS. The dataset by Lu and Kumar (2020), which we name "OVAL", was introduced to test their novel GNN-based branching algorithm (table 2.1). It consists of three different ReLU-based convolutional networks of varying size, which were robustly trained with $\epsilon_{\text{train}} = 2/255$ using the algorithm by Wong and Kolter (2018). For each network, it associates an incorrect class and a perturbation radius $\epsilon_{\text{ver}}$ to a subset of the CIFAR-10 test images. The radii $\epsilon_{\text{ver}}$ are found via a binary search, and are designed to ensure that each problem meets a certain problem difficulty when verified by BaBSR (Bunel et al., 2020b). As a consequence, the dataset lacks properties that are easily verifiable regardless of the employed algorithm, or that are hardly verified by any method. Therefore, we believe it is an effective testing ground for complete verifiers. These properties can be represented in the canonical form of problem (2.1) by setting $\hat{x}_n$ to be the difference between the ground truth logit and the target logit. In order to complement the dataset by Lu and Kumar (2020), we additionally experiment on two larger networks trained using COLT, the recent adversarial training scheme by Balunovic and Vechev (2020) (table 2.2). In this case, we employ a fixed $\epsilon_{\text{ver}}$, chosen to be the radius employed for training. We focus on the first 100 elements of the CIFAR-10 testset, excluding misclassified

| **Network Specifications** | | **Verification Properties** | **Network Architecture** |
|---|---|---|---|
| Name: | 2/255 | 82 properties | Conv2d(3, 32, 3, stride=1, padding=1) |
| Activation: | ReLU | | Conv2d(32,32,4, stride=2, padding=1) |
| Training method: | COLT | $\epsilon_{\text{ver}} = \frac{2}{255}$ | Conv2d(32,128,4, stride=2, padding=1) |
| Total activations: | 49411 | | linear layer of 250 hidden units |
| | | | linear layer of 10 hidden units |
| Name: | 8/255 | 56 properties | Conv2d(3, 32, 5, stride=2, padding=2) |
| Activation: | ReLU | | Conv2d(32,128,4, stride=2, padding=1) |
| Training method: | COLT | $\epsilon_{\text{ver}} = \frac{8}{255}$ | linear layer of 250 hidden units |
| Total activations: | 16643 | | linear layer of 10 hidden units |

**Table 2.2:** Specifications of the COLT-based (Balunovic and Vechev, 2020) CIFAR-10 complete verification dataset.

images. Differently from the dataset by Lu and Kumar (2020), the goal is to verify robustness with respect to any misclassification. The properties can be converted to the canonical form by suitably adding auxiliary layers (Bunel et al., 2020b).

### 2.8.2 Bounding Algorithms

The computational bottleneck of BaBSR was found to be bounding algorithm (Bunel et al., 2020b). Therefore, we start our experimental evaluation by examining the effect of replacing Gurobi Planet, the bounding algorithm from BaBSR, with some of the dual methods evaluated in §2.7. Specifically, we will employ DSG+, Supergradient and Proximal amongst dual iterative algorithms. For each of these three methods, we initialize the problem relative to each subproblem with the dual variables from the parent's bounding, and with CROWN for the root subproblem. Additionally, we compare against **WK + CROWN**, which returns the best bounds amongst WK and CROWN, as representative of propagation-based methods. As explained in section 2.5.2, due to the highly parallelizable nature of the dual algorithms, we are able to compute lower bounds for multiple subproblems at once for DSG+, Supergradient, Proximal, and WK + CROWN. In detail, the number of simultaneously solved subproblems is 300 for the Base network, and 200 for the Wide and Deep networks. For Gurobi, which does not support GPU acceleration, we improve on the original BaBSR implementation by Bunel et al. (2020b) by computing bounds relative to different subproblems in parallel over the CPUs. For both supergradient methods (our Supergradient, and DSG+), we decrease the step size linearly from $\alpha = 10^{-3}$ to $\alpha = 10^{-4}$: the initial step size is smaller than in §2.7 to account for the parent initialization. For Proximal, we do not employ momentum and keep the weight of the proximal terms fixed to $\eta = 10^2$ for all layers throughout the iterations. As in the previous section, the number of iterations for the bounding algorithms are tuned to employ roughly the same time: we use 100 iterations for Proximal, 160 for Supergradient, and 260 for DSG+. Dual bounding computations (for both intermediate and subproblem bounds) were run on a single Nvidia Titan Xp GPU. Gurobi was run on i7-6850K CPUs, using 6 cores.

Figure 2.4 shows that the use of GPU-accelerated dual iterative algorithms is highly beneficial within complete verification. In fact, in spite of the loss in tightness compared to Gurobi, the efficiency of the implementation and the convenient speed-accuracy trade-offs significantly speed up the verification process. On the other hand, the bounds returned by propagation-based methods are too loose to be effectively employed for the last layer bounding, and lead to a very large number of timed out properties. The larger performance difference with respect to incomplete verification (§2.7) is explained by the fact that dual information cannot be propagated from parent to child. Dual solvers, instead, compared to one-off approximations like WK + CROWN, can more effectively exploit the change in subproblem specifications linked to activation splitting. Furthermore, consistently with the incomplete verification results in the last section, Figure 2.4 shows that the Supergradient outperforms DSG+ on average, thus confirming the benefits of our Lagrangian Decomposition approach. Furthermore, Proximal provides an additional increase in average performance over DSG+ on the larger networks, which is visible especially over the properties that are easier to verify. The gap between competing bounding methods increases with the size of the employed network, both in terms of layer width and network depth. We have seen that, by exploiting dual bounding algorithms, the performance of BaBSR can be significantly improved. We will now move on to studying the role of other branch and bound components.

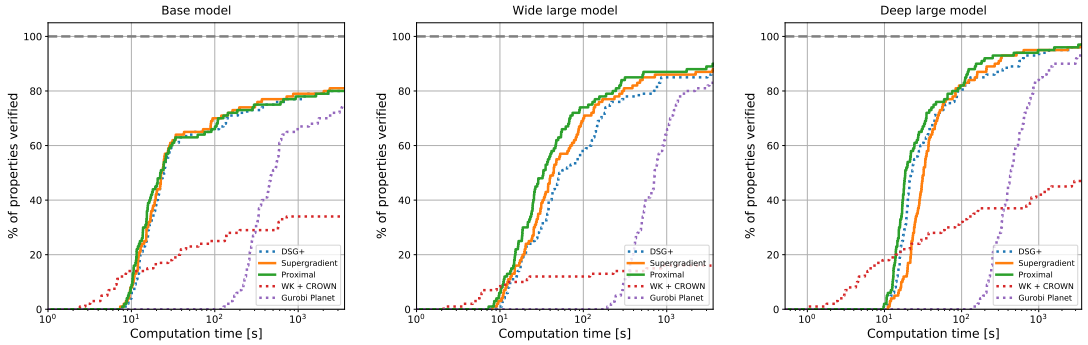
### 2.8.3 Branching

The use of our Proximal solver improves the average verification performance within BaBSR (see §2.8.2). We now keep the bounding algorithm fixed to Proximal and study the effect of various branching strategies on verification time. As in BaBSR, we update intermediate bounds after each activation split with the best bounds between IBP and WK. We consider the following branching schemes:

- **SR** denotes the original branching scheme from BaBSR (Bunel et al., 2020b), relying on the activation scores found in equation (2.22).

| | Base | | | Wide | | | Deep | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout |
| DSG+ | 812.84 | 135 686.10 | 20.00 | 638.63 | 73 591.82 | 13.00 | 257.01 | 21 928.52 | 4.00 |
| Supergradient | **776.35** | 147 347.26 | **19.00** | 561.50 | 74 274.48 | 12.00 | 228.67 | 14 851.40 | **3.00** |
| Proximal | 808.62 | 166 478.44 | 20.00 | **498.69** | 80 703.96 | **10.00** | **206.54** | 18 139.62 | **3.00** |
| WK + CROWN | 2417.83 | 725 627.96 | 66.00 | 3042.91 | 736 415.20 | 84.00 | 2055.09 | 382 353.30 | 53.00 |
| Gurobi Planet | 1352.15 | 7013.72 | 25.00 | 1236.54 | 2737.36 | 17.00 | 782.84 | 848.32 | 7.00 |

**(a)** Comparison of average solving time, average number of solved subproblems and the percentage of timed out properties. The best performing method is highlighted in bold.




**(b)** Cactus plots: percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

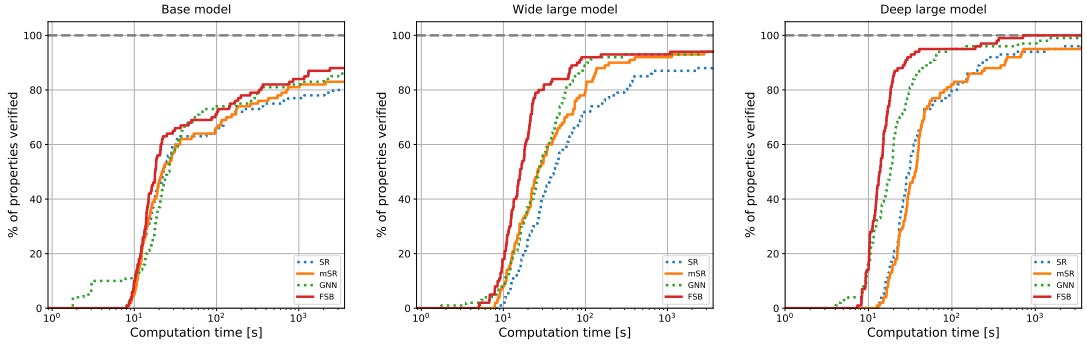
**Figure 2.4:** Complete verification performance of different bounding algorithms within BaBSR, on the OVAL dataset.

- **mSR** denotes our modification of the SR branching scheme (min-based SR), in which $\mathbf{s}_{\text{SR}}$ scores are replaced by $\mathbf{s}_{\text{FSB}}$ as seen in equation (2.23).
- **GNN** is the learning-based approach from Lu and Kumar (2020). It exploits a trained Graph Neural Network, which takes the topology of the network to be verified as input, to perform the branching decision, and falls back to SR whenever the decision from the GNN is deemed unsatisfactory. We re-train the network using Proximal as bounding algorithm, omitting the online fine-tuning due to its modest empirical impact (Lu and Kumar, 2020).
- **FSB** is our novel branching scheme (see §2.4), which exploits the mSR scores to select a subset of the branching choices, and then approximates strong branching using bounds from WK + CROWN.

Similarly to dual bounding, branching computations were parallelized over batches of subproblems, and run on a single Nvidia Titan Xp GPU for all considered methods. In the context of our implementation, the time required for branching is negligible

| | Base | | | Wide | | | Deep | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout |
| SR | 822.61 | 146 700.30 | 20.00 | 524.24 | 67 495.74 | 11.00 | 234.23 | 15 443.30 | 4.00 |
| mSR | 704.39 | 125 989.32 | 17.00 | 299.59 | 38 477.18 | 6.00 | 256.17 | 13 573.06 | 4.00 |
| GNN | 633.71 | 92 904.44 | 14.00 | 274.57 | 25 337.36 | 6.00 | 88.43 | 3959.28 | 1.00 |
| FSB | **546.22** | **85 433.94** | **12.00** | **247.13** | **20 324.06** | **6.00** | **32.29** | **1702.02** | **0.00** |

**(a)** Comparison of average runtime, average number of solved subproblems and the percentage of timed out properties. The best performing method is highlighted in bold.




**(b)** Cactus plots: percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

**Figure 2.5:** Using Proximal as bounding algorithm, complete verification performance of different branching strategies, on the OVAL dataset.

with respect to the cost of the bounding procedure, for all considered branching schemes. For this set of experiments, Proximal is additionally employed to compute upper bounds to a modified version of problem (2.1), where the minimization is replaced by a maximization, for each subproblem. This maximizes the dual information available to the GNN, and provides an additional infeasibility check[6] for all methods. In fact, as infeasible subproblems will result in an unbounded dual, the subproblem can be discarded whenever the values of the upper and lower bounds cross. Empirically, this results in a modest decrease in the size of the enumeration tree and a minor increase in runtime (compare the results for SR in figure 2.5, with those for Proximal in figure 2.4).

Figure 2.5 shows that our simple modification of SR successfully reduces the average number of subproblems to termination. This results in faster verification for both the Base and Wide network. For the Deep network, however, mSR tends to branch on earlier layers, thus involving a larger average number of intermediate bounding computations (see §2.5.1.1). This overhead is not matched by a significant

[6] A given activation split might empty the feasible region of problem (2.7).

reduction of the enumeration tree, slowing down overall verification. The results for FSB demonstrate that coupling the mSR scores with fast dual algorithms yields a larger and more consistent reduction of subproblems with respect to SR. As a consequence, FSB produces significant verification speed-ups to SR, which range from roughly 35% on the Base network to an order of magnitude on the Deep network. Furthermore, FSB is strongly competitive with GNN. On the considered networks, FSB outperforms the learned approach both in terms of average verification time and number of subproblems. Differently from GNN, strong verification performance is achieved without incurring large training-related offline costs.

### 2.8.4 Intermediate Bounds

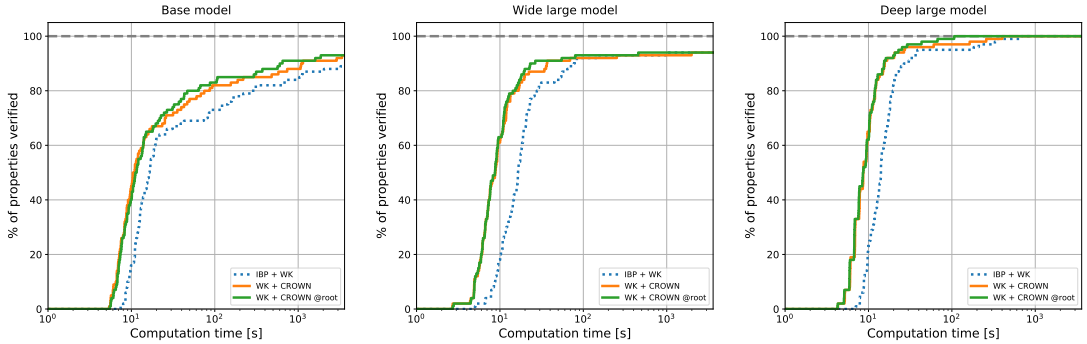
In this section, we consider the effect of various intermediate bounding strategies (see §2.2.2.3) on final verification performance. For this set of experiments, we use Proximal as bounding algorithm (for the subproblem bounds), and FSB for the branching strategy, computing intermediate bounds with the following algorithms:

- **IBP + WK** denotes the layer-wise best bounds returned by IBP and WK (see §2.7), updated after each activation split (see §2.5.1.1). This is the intermediate bounding strategy employed in BaBSR (Bunel et al., 2020b).
- **WK + CROWN** denotes the layer-wise best bounds returned by WK and CROWN, updated after each activation split.
- **WK + CROWN @root** restricts WK + CROWN to the root of the branch and bound tree, foregoing any possible tightening after activation splits.

As expected from the incomplete verification experiments in §2.7, replacing IBP with CROWN markedly tightens intermediate bounds. This is testified by the decrease in the number of average subproblems to termination visible in figure 2.6. As a consequence, WK + CROWN reduces verification time for all the three considered networks. Moreover, restricting WK + CROWN to the branch and bound root results in an increase in the average number of subproblems to termination. This is due to both the reduced cost per branch and bound iteration,

| | Base | | | Wide | | | Deep | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout |
| IBP + WK | 526.77 | 94 959.20 | 11.00 | 245.67 | 21 429.28 | **6.00** | 31.46 | 1704.10 | **0.00** |
| WK + CROWN | 384.60 | 56 823.50 | **7.00** | 248.23 | 15 750.48 | **6.00** | 17.95 | 490.16 | **0.00** |
| WK + CROWN @ROOT | **329.46** | 97 219.08 | **7.00** | **230.30** | 43 952.94 | **6.00** | **11.41** | 533.90 | **0.00** |

**(a)** Comparison of average runtime, average number of solved subproblems and the percentage of timed out properties. The best performing method is highlighted in bold.




**(b)** Cactus plots: percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

**Figure 2.6:** Using Proximal for the last layer bounding, complete verification performance of different intermediate bounding schemes, on the OVAL dataset.

and a loss in bounding tightness. As evidenced by the decrease in verification time on all the three considered networks, WK + CROWN @root produces better speed-accuracy trade-offs than the other intermediate bounding strategies, and is particularly convenient for larger networks.

### 2.8.5 Comparison of Complete Verifiers

In §2.8.2, §2.8.3, and §2.8.4, we have studied the effect of isolated branch and bound components on the OVAL dataset. We conclude our experimental evaluation by comparing our branch and bound framework with state-of-the-art complete verifiers on both the OVAL and COLT datasets. For both benchmarks, we consider the following algorithms:

- **MIP** solves problem (2.1) as a Mixed-Integer linear Program (MIP) via Gurobi, exploiting the representation of ReLU activations used by (Tjeng et al., 2019). Gurobi was run on i7-6850K CPUs, using 6 cores. In order to minimize pre-processing time, intermediate bounds are pre-computed with the layer-wise best bounds between IBP and WK, on an Nvidia Titan Xp GPU.

- **Gurobi BaBSR** is a multi-core adaptation of the BaBSR branch and bound framework (Bunel et al., 2020b). Gurobi is run on 6 cores from i7-6850K CPUs (see "Gurobi Planet" in §2.8.2), branching and intermediate bounding were run on an Nvidia Titan Xp GPU.

- **Proximal BaBSR** replaces the Gurobi-based Planet solver used in BaBSR with our Proximal bounding algorithm (see "Proximal" in §2.8.2), run on an Nvidia Titan Xp GPU.

- **BaDNB** is our novel branch and bound framework (see §2.5). Compared to Proximal BaBSR, it employs FSB as branching strategy, dynamically determines the number of Proximal iterations, and computes intermediate bounds via WK + CROWN @root (see §2.8.4). BaDNB was run on an Nvidia Titan Xp GPU.

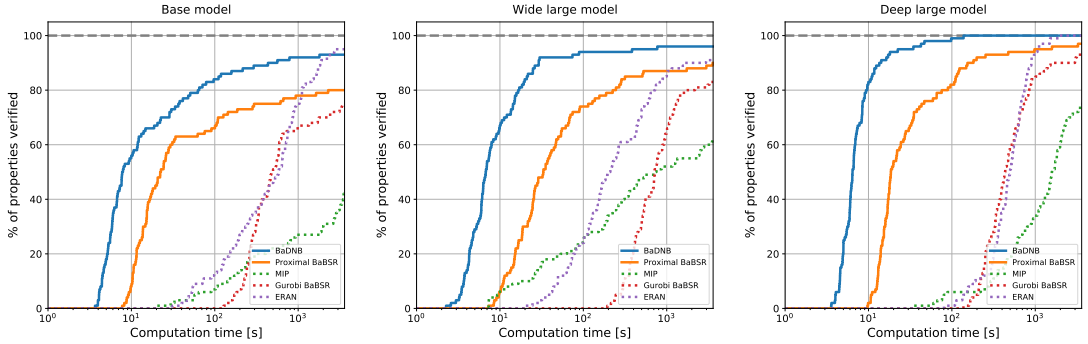
- **ERAN** is the complete verification toolbox by Singh et al. (2020), based on several works combining abstract interpretation, propagation-based methods, LP and MILP solvers (Singh et al., 2018, 2019a,b,c). Results are taken from the recent VNN-COMP competition (VNN-COMP, 2020). On the OVAL dataset, ERAN was executed on a 2.6 GHz Intel Xeon CPU E5-2690, using 14 cores. On the COLT dataset, it was run on a 10 Core Intel i9-7900X Skylake CPU.

#### 2.8.5.1 OVAL Dataset

Figure 2.7 reports results for the OVAL dataset. MIP, which relies on a black-box MIP solver, is the slowest verification method in all three cases, highlighting the importance of specialized algorithms for neural network verification. Gurobi BaBSR improves upon MIP, thus demonstrating the benefits of a customized branch-and-bound framework. However, as seen in §2.8.2, the use of Gurobi as bounding algorithm severely hinders scalability. Proximal BaBSR showcases the benefits of specialized solvers for the Planet relaxation: it significantly outperforms Gurobi BaBSR and is faster than ERAN on the larger networks. Furthermore, BaDNB manages to further cut verification times, yielding speed-ups to Proximal BaBSR

| | Base | | | Wide | | | Deep | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout |
| BaDNB | **309.30** | 38 496.52 | 7.00 | **165.54** | 11 258.56 | **4.00** | **10.50** | 368.16 | **0.00** |
| Proximal BaBSR | 808.62 | 166 478.44 | 20.00 | 498.69 | 80 703.96 | 10.00 | 206.54 | 18 139.62 | 3.00 |
| MIP | 2582.30 | 13 929.44 | 57.00 | 1702.88 | 4170.95 | 38.00 | 1831.44 | 6268.08 | 25.00 |
| Gurobi BaBSR | 1352.15 | 7013.72 | 25.00 | 1236.54 | 2737.36 | 17.00 | 782.84 | 848.32 | 7.00 |
| ERAN | 805.89 | - | **5.00** | 632.12 | - | 9.00 | 545.72 | - | **0.00** |

**(a)** Comparison of average runtime, average number of solved subproblems and the percentage of timed out properties. The best performing method is highlighted in bold.




**(b)** Cactus plots: percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

**Figure 2.7:** Performance of different complete verification algorithms, on the OVAL dataset.

that reach an order of magnitude on the Deep network. This testifies the advantages of Filtered Smart Branching and of the other design choices presented in §2.5.1. In addition to the results presented in §2.8.3, and §2.8.4, a comparison of BaDNB in table 2.7(a) with WK + CROWN @root in table 2.6(a) shows that, by automatically adjusting the number of dual iterations (§2.5.1.2), we can obtain a further 40% reduction in average verification time on the Wide network.

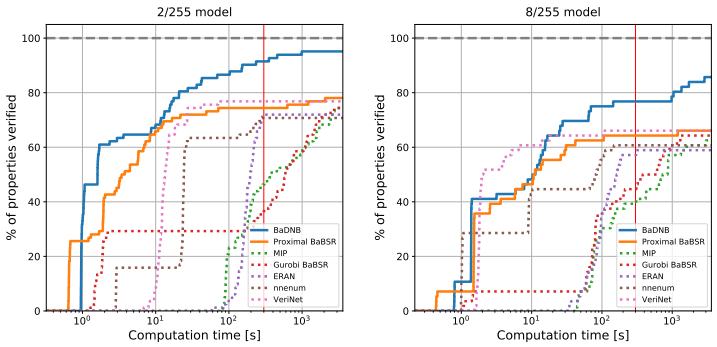
#### 2.8.5.2 COLT Dataset

We now report results for the COLT dataset, which offer insight into the scalability of the various complete verifiers on larger networks. Owing to the wider participation to this benchmark within VNN-COMP (VNN-COMP, 2020), we additionally report results for the two best-performing algorithms within the competition:

- **nnenum** by Bak et al. (2020), which pairs propagation-based methods and LP solvers with path enumeration techniques (Tran et al., 2019). nenum was executed on an Amazon EC2 `m4.10xlarge` cloud instance, with a 40-core 2.4 GHz Intel Xeon E5-2676 v3 CPU.

| | 2/255 | | | 8/255 | | |
|---|---|---|---|---|---|---|
| Method | time [s] | subproblems | %Timeout | time [s] | subproblems | %Timeout |
| BaDNB | **211.06** | 2064.02 | **4.88** | **667.29** | 17 254.61 | **14.29** |
| Proximal BaBSR | 842.00 | 17 979.17 | 21.95 | 1249.02 | 55 653.04 | 33.93 |
| MIP | 1353.07 | 103.11 | 25.61 | 1611.46 | 461.70 | 37.50 |
| Gurobi BaBSR | 1313.88 | 32.44 | 25.61 | 1461.48 | 310.18 | 35.71 |
| ERAN* | 1136.76 | - | 28.05 | 1541.98 | - | 41.07 |
| nnenum* | 1081.02 | - | 29.27 | 1431.46 | - | 39.29 |
| VeriNet* | 845.57 | - | 23.17 | 1225.24 | - | 33.93 |

**(a)** Comparison of average runtime, average number of solved subproblems and the percentage of timed out properties. The best performing method is highlighted in bold. *the method was run with a 5-minute time limit, within VNN-COMP (2020).




**(b)** Cactus plots: percentage of solved properties as a function of runtime. Baselines are represented by dotted lines. The red vertical line marks the time limit for the ERAN, nnenum and VeriNet experiments from VNN-COMP (VNN-COMP, 2020).

**Figure 2.8:** Performance of different complete verifiers on adversarial robustness verification of larger, COLT-trained convolutional, networks (Balunovic and Vechev, 2020).

- **VeriNet** from Henriksen and Lomuscio (2020), which presents modifications to the branch and bound algorithm from Neurify (Wang et al., 2018b). In particular, both the subproblem upper bounding strategy and the activation-based branching schemes are improved upon. VeriNet was run on a Ryzen 3700X 3.6 GHz 8-core CPU.

Due to the specifications of VNN-COMP the experiments for ERAN, nnenum and VeriNet were run with a 5-minute time limit on this dataset. In line with the time-limit for the other methods, Table 2.8(a) reports a runtime of 1 hour for all timed-out properties.

The performance of MIP in figure 2.8 demonstrates that black-box MIP solvers are unsuitable for larger networks. In fact MIP can rarely verify any property in less than hundreds of seconds. Furthermore, Gurobi's scaling problems are further evidenced by the results for Gurobi BaBSR, which hardly improves upon MIP's

performance. As seen on the OVAL dataset (figure 2.8), the use of our dual bounding algorithm greatly improves upon the original BaBSR implementation (Bunel et al., 2020b): Proximal BaBSR consistently outperforms ERAN and nnenum, and it is competitive with VeriNet. However, Proximal BaBSR still times out on a relatively large share of the considered properties, underscoring the limitations of BaBSR's design. The superior performance of BaDNB demonstrates that a scrupulous deployment of fast dual bounds throughout the branch and bound procedure is key to effective neural network verification.

## 2.9 Discussion

We have presented BaDNB, a novel branch and bound framework for neural network verification, and empirically demonstrated its advantages to state-of-the-art verification systems.

As part of BaDNB, we proposed a novel dual approach to neural network bounding, based on Lagrangian Decomposition. Our bounding algorithms provide significant benefits compared to off-the-shelf solvers and improve on both looser relaxations and on a previous method based on Lagrangian relaxation. While we have focused on the convex hull of element-wise activation functions, our dual approach is far more general. In fact, we believe that Lagrangian Decomposition has the potential to scale up tighter relaxations from the Mixed Integer Linear Programming literature (Sherali and Adams, 1994). We have furthermore shown that inexpensive dual algorithms can significantly speed up verification if employed to select branching decisions and to tighten intermediate bounds. We decided to keep costs low for these branch and bound components in order to obtain a well-rounded complete verifier. However, we are convinced that further verification improvements can be obtained by selectively employing solvers for Lagrangian Decomposition in this context.

# 3

# Scaling the Convex Barrier with Sparse Dual Algorithms

## Contents


3.1 Introduction . . . . . **61**
3.2 Preliminaries: Neural Network Relaxations . . . . . **63**
3.2.1 Planet Relaxation . . . . . 64
3.2.2 A Tighter Relaxation . . . . . 65
3.3 A Dual Formulation for the Tighter Relaxation . . . . . **67**
3.4 Active Set . . . . . **69**
3.4.1 Solver . . . . . 69
3.4.2 Extending the Active Set . . . . . 71
3.5 Saddle Point . . . . . **74**
3.5.1 Sparsity via Sufficient Statistics . . . . . 74
3.5.2 Solver . . . . . 77
3.6 Implementation Details, Technical Challenges . . . . . **80**
3.6.1 Parallelism, masked forward/backward passes . . . . . 81
3.6.2 Stratified bounding for branch and bound . . . . . 81
3.7 Related Work . . . . . **83**
3.8 Experiments . . . . . **84**
3.8.1 Experimental Setting . . . . . 85
3.8.2 Incomplete Verification . . . . . 86
3.8.3 Branch and Bound . . . . . 91
3.9 Discussion . . . . . **98**

# Abstract

Tight and efficient neural network bounding is crucial to the scaling of neural network verification systems. Many efficient bounding algorithms have been presented recently, but they are often too loose to verify more challenging properties. This is due 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 currently lacks an efficient customized solver. We alleviate this deficiency by presenting two novel dual algorithms: one operates a subgradient method on a small active set of dual variables, the other exploits the sparsity of Frank-Wolfe type optimizers to incur only a linear memory cost. Both methods recover the strengths of the new relaxation: tightness and a linear separation oracle. At the same time, they share 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 can obtain better bounds than off-the-shelf solvers in only a fraction of their running time, attaining significant formal verification speed-ups.

## 3.1 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 (Kurakin et al., 2017; Carlini and Wagner, 2017; Madry et al., 2018). 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., 2020; 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., 2019b). 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., 2018b; Bunel et al., 2020a) are not easily applicable, limiting the scaling of the new relaxation.

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

- Relying on an *active set* of dual variables, we present a unified dual treatment that includes both a linearly sized LP relaxation (Ehlers, 2017) and the tighter formulation. As a consequence, we obtain an inexpensive dual initializer, named Big-M, which is competitive with dual approaches on the looser relaxation (Dvijotham et al., 2018b; Bunel et al., 2020a). Moreover, by dynamically extending the active set, we obtain a subgradient-based solver, named Active Set, which rapidly overcomes the convex barrier and yields much tighter bounds if a larger computational budget is available.

- The tightness of the bounds attainable by Active Set depends on the memory footprint through the size of the active set. By exploiting the properties of Frank-Wolfe style optimizers (Frank and Wolfe, 1956), we present Saddle Point, a solver that deals with the exponentially many constraints of the relaxation by Anderson et al. (2020) while only incurring a linear memory cost. Saddle Point eliminates the dependency on memory at the cost of a potential reduction of the dual feasible space, but is nevertheless very competitive with Active Set in settings requiring tight bounds.

- Both solvers are *sparse* and recover the strengths of the original primal problem (Anderson et al., 2020) in the dual domain. In line with previous dual solvers, both methods yield valid bounds at anytime, leverage convolutional network structure and enjoy *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). Owing to this, we show that both solvers

can yield large complete verification gains compared to primal approaches (Anderson et al., 2020) and previous dual algorithms.

Code implementing our algorithms is available at `https://github.com/oval-group/oval-bab` as part of the OVAL neural network verification framework.

A preliminary version of this work appeared in the proceedings of the Ninth International Conference on Learning Representations (De Palma et al., 2021a). The present article significantly extends it by:

1. Presenting Saddle Point (§3.5), a second solver for the relaxation by Anderson et al. (2020), which is more memory efficient than both Active Set and the original cutting plane algorithm by Anderson et al. (2020).

2. Providing a detailed experimental evaluation of the new solver, both for incomplete and complete verification.

3. Presenting an adaptive and more intuitive scheme to switch from looser to tighter bounding algorithms within branch and bound (§3.6.2).

4. Investigating the effect of different speed-accuracy trade-offs from the presented solvers in the context of complete verification.

## 3.2 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, $\text{col}_i(W)$ and $\text{row}_i(W)$ respectively denote the $i$-th column and the $i$-th row of matrix $W$. Finally, 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 $\sum_i \text{col}_i(W)^T \mathbf{x}$.

Let $\mathcal{C}$ be the network input domain. Similar to Dvijotham et al. (2018b); Bunel et al. (2020a), we assume that the minimization of a linear function over $\mathcal{C}$ can be performed efficiently. For instance, this is the case for $\ell_\infty$ and $\ell_2$ norm perturbations.

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:

$$\min_{\mathbf{x},\hat{\mathbf{x}}} \quad \hat{x}_n \qquad \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \tag{3.1a}$$

$$\hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} \quad k \in [\![0, n-1]\!], \tag{3.1b}$$

$$\mathbf{x}_k = \sigma\left(\hat{\mathbf{x}}_k\right) \qquad k \in [\![1, n-1]\!], \tag{3.1c}$$

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 (3.1) is non-convex due to the activation function's non-linearity, that is, due to constraint (3.1c). As solving it is NP-hard (Katz et al., 2017), it is commonly approximated by a convex relaxation (see §3.7). 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 §3.2.1. We then consider a tighter one in §3.2.2.

### 3.2.1 Planet Relaxation

The so-called Planet relaxation (Ehlers, 2017) has enjoyed widespread use due to its amenability to efficient customised solvers (Dvijotham et al., 2018b; Bunel et al., 2020a) and is the "relaxation of choice" for many works in the area (Salman et al., 2019b; Singh et al., 2019c; Bunel et al., 2020b; Balunovic and Vechev, 2020; Lu and Kumar, 2020). Here, we describe it in its non-projected form $\mathcal{M}_k$, corresponding to the LP relaxation of the Big-M Mixed Integer Programming

(MIP) formulation (Tjeng et al., 2019). Applying $\mathcal{M}_k$ to problem (3.1) results in the following linear program:

$$\begin{aligned}
\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \quad \hat{x}_n \quad \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C} \\
& \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!], \\
& \left.\begin{array}{l} \mathbf{x}_k \geq \hat{\mathbf{x}}_k, \quad \mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k, \\ \mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k), \\ (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \times [\mathbf{0},\ \mathbf{1}] \end{array}\right\} := \mathcal{M}_k && k \in [\![1, n-1]\!],
\end{aligned} \tag{3.2}$$

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 B.5. Projecting out auxiliary variables $\mathbf{z}$ results in the Planet relaxation (cf. appendix B.2.1 for details), which replaces (3.1c) by its convex hull.

Problem (3.2), 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 specialized solvers. Customised dual solvers have been designed by relaxing constraints (3.1b), (3.1c) (Dvijotham et al., 2018b) or replacing (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 (3.2) in only a fraction of the runtime of off-the-shelf solvers.

### 3.2.2 A Tighter Relaxation

A much tighter approximation of problem (3.1) than the Planet relaxation (§3.2.1) can be obtained by representing the convex hull of the composition of constraints (3.1b) and (3.1c) rather than the convex hull of constraint (3.1c) alone. A formulation of this type was recently introduced by Anderson et al. (2020). In order to represent the interaction between $\mathbf{x}_k$ and all possible subsets of the activations of the previous layer, the formulation relies on a number of auxiliary matrices, which we will now introduce. Weight matrices are masked entry-wise via $I_k$, binary masks belonging to the following set: $2^{W_k} = \{0,1\}^{n_k \times n_{k-1}}$. We define $\mathcal{E}_k := 2^{W_k} \setminus \{0,1\}$, to exclude the

all-zero and all-one masks, which we treat separately (see §3.2.2.1). In addition, the formulation requires bounds on the subsets of $\mathbf{x}_{k-1}$ selected via the masking. In order to represent these bounds, the formulation exploits matrices $\check{L}_{k-1}, \check{U}_{k-1} \in \mathbb{R}^{n_k \times n_{k-1}}$, which are also masked via $I_k$ and computed via interval arithmetic as follows:

$$\check{L}_{k-1}[i,j] = \mathbf{l}_{k-1}[j]\mathbb{1}_{W_k[i,j]\geq 0} + \mathbf{u}_{k-1}[j]\mathbb{1}_{W_k[i,j]<0},$$
$$\check{U}_{k-1}[i,j] = \mathbf{u}_{k-1}[j]\mathbb{1}_{W_k[i,j]\geq 0} + \mathbf{l}_{k-1}[j]\mathbb{1}_{W_k[i,j]<0}.$$

The new representation results in the following primal problem:

$$\begin{aligned}
&\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \ \hat{x}_n \ \text{ s.t.} \\
&\quad \mathbf{x}_0 \in \mathcal{C} \\
&\quad \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!], \\
&\quad (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in \mathcal{M}_k \\
&\quad \mathbf{x}_k \leq \left.\begin{pmatrix} (W_k \odot I_k)\,\mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k + \\ -\left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k) + \\ +\left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \end{pmatrix} \ \forall I_k \in \mathcal{E}_k \right\} := \mathcal{A}_k && k \in [\![1, n-1]\!].
\end{aligned} \tag{3.3}$$

Both $\mathcal{M}_k$ and $\mathcal{A}_k$ yield valid MIP formulations for problem (3.1) 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 (3.3), one can efficiently compute the most violated constraint (if any) at that point. Denoting by $\mathcal{A}_{\mathcal{E},k} = \mathcal{A}_k \setminus \mathcal{M}_k$ the exponential family of constraints, the mask associated to the most violated constraint in $\mathcal{A}_{\mathcal{E},k}$ can be computed in linear-time (Anderson et al., 2020) as:

$$I_k[i,j] = \mathbb{1}^T_{\left((1-\mathbf{z}_k[i])\odot\check{L}_{k-1}[i,j] + \mathbf{z}_k[i]\odot\check{U}_{k-1}[i,j] - \mathbf{x}_{k-1}[j]\right)W_k[i,j]\geq 0}. \tag{3.4}$$

The most violated constraint in $\mathcal{A}_k$ is then obtained by comparing the constraint violation from the output of oracle (3.4) to those from the constraints in $\mathcal{M}_k$.

#### 3.2.2.1 Pre-activation bounds

Set $\mathcal{A}_k$ defined in problem (3.3) slightly differs from the original formulation of Anderson et al. (2020), as the latter does not exploit pre-activation bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ within the exponential family. In particular, constraints $\mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k$, and $\mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k)$, which we treat via $\mathcal{M}_k$, are replaced by looser counterparts. 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 (3.2). In appendix B.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 (3.3).

#### 3.2.2.2 Cutting plane algorithm

Owing to the exponential number of constraints, problem (3.3) cannot be solved as it is. As outlined by Anderson et al. (2020), the availability of a linear-time separation oracle (3.4) offers a natural primal *cutting plane* algorithm, which can then be implemented in off-the-shelf solvers: solve the Big-M LP (3.2), 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).

## 3.3 A Dual Formulation for the Tighter Relaxation

Inspired by the success of dual approaches on looser relaxations (Bunel et al., 2020a; Dvijotham et al., 2020), we show that the formal verification gains by Anderson et al. (2020) (see §3.2.2) 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 (3.3) 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* (3.3) *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 B.1, neither of the two dual derivations by Bunel et al. (2020a); Dvijotham et al. (2018b) readily satisfy all desiderata at once. Therefore, we need a completely different approach. Starting from primal (3.3), we relax all constraints in $\mathcal{A}_k$ except box constraints (see §3.2.1). In order to simplify notation, we employ dummy variables $\boldsymbol{\alpha}_0 = 0$, $\boldsymbol{\beta}_0 = 0$, $\boldsymbol{\alpha}_n = I$, $\boldsymbol{\beta}_n = 0$, obtaining the following dual problem (derivation in appendix B.4):

$$\begin{aligned}
&\max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\geq 0} d(\boldsymbol{\alpha},\boldsymbol{\beta}) \qquad \text{where:} \qquad d(\boldsymbol{\alpha},\boldsymbol{\beta}) := \min_{\mathbf{x},\mathbf{z}} \;\; \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}),\\
&\qquad \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}) = \left[\begin{array}{l} \sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k - \sum_{k=0}^{n-1} \boldsymbol{f}_k(\boldsymbol{\alpha},\boldsymbol{\beta})^T \mathbf{x}_k - \sum_{k=1}^{n-1} \boldsymbol{g}_k(\boldsymbol{\beta})^T \mathbf{z}_k \\ + \sum_{k=1}^{n-1} \left( \sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \square \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k) \right) \end{array}\right.\\
&\text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad k \in [\![1, n-1]\!],
\end{aligned} \tag{3.5}$$

where functions $\boldsymbol{f}_k, \boldsymbol{g}_k$ are defined as follows:

$$\begin{aligned}
\boldsymbol{f}_k(\boldsymbol{\alpha},\boldsymbol{\beta}) &= \;\; \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \textstyle\sum_{I_k} \boldsymbol{\beta}_{k,I_k} + \sum_{I_{k+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}},\\
\boldsymbol{g}_k(\boldsymbol{\beta}) &= \left[\begin{array}{l} \sum_{I_k \in \mathcal{E}_k} \left( W_k \odot (1 - I_k) \odot \check{U}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k \\ + \sum_{I_k \in \mathcal{E}_k} \left( W_k \odot I_k \odot \check{L}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \sum_{I_k \in \mathcal{E}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k. \end{array}\right.
\end{aligned} \tag{3.6}$$

We employ $\sum_{I_k}$ as a shorthand for $\sum_{I_k \in 2^{W_k}}$. Both functions therefore include sums over an exponential number of $\boldsymbol{\beta}_{k,I_k}$ variables.

This is again a challenging problem: the exponentially many constraints in the primal (3.3) are associated to an exponential number of variables, as $\boldsymbol{\beta} = \{\boldsymbol{\beta}_{k,I_k} \forall I_k \in 2^{W_k}, k \in [\![1, n-1]\!]\}$. Nevertheless, we show that the requirements of Fact 1 can be met by operating on restricted versions of the dual. We present two specialized algorithms for problem (3.5): one considers only a small active set of dual variables (§3.4), the other restricts the dual domain while exploiting the sparsity of Frank-Wolfe style iterates (§3.5). Both algorithms are sparse, anytime and yield bounds reflecting the tightness of the new relaxation.

**Algorithm 3** Active Set

**function** ACTIVESET_COMPUTE_BOUNDS(Problem (3.5))
  Initialise duals $\boldsymbol{\alpha}^0, \beta_0^0, \beta_1^0$ using Algorithm (5)
  Set $\beta^0_{k,I_k} = 0, \forall\ I_k \in \mathcal{E}_k$
  $\mathcal{B} = \emptyset$
  **for** `nb_additions` **do**
    **for** $t \in [\![0, T-1]\!]$ **do**
      $\mathbf{x}^{*,t}, \mathbf{z}^{*,t} \in \text{argmin}_{\mathbf{x},\mathbf{z}}\, \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^t, \boldsymbol{\beta}^t_{\mathcal{B}})$ using (3.8),(3.9) ▷ inner minimization
      **if** $t \leq$ `nb_vars_to_add` **then**
        For each layer $k$, add output of (3.4) called at $(\mathbf{x}^*, \mathbf{z}^*)$ to $\mathcal{B}_k$ ▷ active set extension
      $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{B}} \leftarrow (\boldsymbol{\alpha}^t, \boldsymbol{\beta}^t) + H(\nabla_{\boldsymbol{\alpha}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}), \nabla_{\boldsymbol{\beta}_{\mathcal{B}}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}))$ ▷ supergradient step, using (3.10)
      $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{B}} \leftarrow \max(\boldsymbol{\alpha}^{t+1}, 0), \max(\boldsymbol{\beta}^{t+1}_{\mathcal{B}}, 0)$ ▷ dual projection
  **return** $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \boldsymbol{\beta}^T_{\mathcal{B}})$

We conclude this section by pointing out that, as stated in §3.2, the only assumption for $\mathcal{C}$ in problem (3.5) is that it allows for efficient linear minimization. This is the case for both $\ell_\infty$ and $\ell_2$ norm perturbations, which are hence supported by our solvers.

## 3.4 Active Set

We present Active Set (Algorithm 3), a supergradient-based solver that operates on a small active set of dual variables $\boldsymbol{\beta}_{\mathcal{B}}$. Starting from the dual of problem (3.2), Active Set iteratively adds variables to $\boldsymbol{\beta}_{\mathcal{B}}$ and solves the resulting reduced version of problem (3.5). We first describe our solver on a fixed $\boldsymbol{\beta}_{\mathcal{B}}$ (§3.4.1) and then outline how to iteratively modify the active set (§3.4.2).

### 3.4.1 Solver

We want to solve a version of problem (3.5) for which $\mathcal{E}_k$, the exponentially-sized set of $I_k$ masks for layer $k$, is restricted to some constant-sized set[1] $\mathcal{B}_k \subseteq \mathcal{E}_k$, with $\mathcal{B} = \cup_{k \in [\![1, n-1]\!]} \mathcal{B}_k$. By keeping $\mathcal{B} = \emptyset$, we recover a novel dual solver for the Big-M relaxation (3.2) (explicitly described in appendix B.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 (3.6), (3.5) and removing these from the formulation, we obtain:

[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 $\boldsymbol{\beta}_{\mathcal{B}}$.

$$\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}}) = \begin{bmatrix} \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \sum_{I_k \in \mathcal{B}_k \cup \{0,1\}} \boldsymbol{\beta}_{k,I_k}, \\ + \sum_{I_{k+1} \in \mathcal{B}_{k+1} \cup \{0,1\}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}} \end{bmatrix}$$

$$\boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_{\mathcal{B}}) = \begin{bmatrix} \sum_{I_k \in \mathcal{B}_k} \left( W_k \odot (1 - I_k) \odot \check{U}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k \\ + \sum_{I_k \in \mathcal{B}_k} \left( W_k \odot I_k \odot \check{L}_{k-1} \right) \diamond \boldsymbol{\beta}_{k,I_k} + \sum_{I_k \in \mathcal{B}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k, \end{bmatrix}$$

along with the reduced dual problem:

$$\begin{aligned} \max_{(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}) \geq 0} d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}}) \qquad & \text{where:} \qquad d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}}) := \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}}), \\ \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}}) = & \begin{bmatrix} \sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k - \sum_{k=0}^{n-1} \boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})^T \mathbf{x}_k - \sum_{k=1}^{n-1} \boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_{\mathcal{B}})^T \mathbf{z}_k \\ + \sum_{k=1}^{n-1} \left( \sum_{I_k \in \mathcal{B}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \square \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k) \right) \end{bmatrix} \\ \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \qquad & (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0}, \ \mathbf{1}] \qquad k \in [\![1, n-1]\!]. \end{aligned} \tag{3.7}$$

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 (3.7), 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:

$$\mathbf{x}_k^* = \mathbb{1}_{\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}) \geq 0} \odot \hat{\mathbf{u}}_k + \mathbb{1}_{\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{B}}) < 0} \odot \hat{\mathbf{l}}_k, \qquad \mathbf{z}_k^* = \mathbb{1}_{\boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_{\mathcal{B}}) \geq 0} \odot \mathbf{1}. \tag{3.8}$$

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

$$\mathbf{x}_0^* \in \ \text{argmin}_{\mathbf{x}_0} \quad \boldsymbol{f}_{\mathcal{B},0}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})^T \mathbf{x}_0 \qquad \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}. \tag{3.9}$$

We assumed in §3.2 that (3.9) can be performed efficiently. We refer the reader to Bunel et al. (2020a) for descriptions of the minimisation when $\mathcal{C}$ is a $\ell_\infty$ or $\ell_2$ 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:

$$\begin{gathered}\nabla_{\boldsymbol{\alpha}_k} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = W_k \mathbf{x}^*_{k-1} + \mathbf{b}_k - \mathbf{x}^*_k, \qquad \nabla_{\beta_{k,0}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{x}^*_k - \mathbf{z}^*_k \odot \hat{\mathbf{u}}_k, \\ \nabla_{\beta_{k,1}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{x}^*_k - \left(W_k \mathbf{x}^*_{k-1} + \mathbf{b}_k\right) + (1 - \mathbf{z}^*_k) \odot \hat{\mathbf{l}}_k, \\ \nabla_{\beta_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \begin{pmatrix} \mathbf{x}^*_k - (W_k \odot I_k)\, \mathbf{x}^*_{k-1} + \\ + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}^*_k) + \\ -\mathbf{z}^*_k \odot \mathbf{b}_k + \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}^*_k \end{pmatrix} I_k \in \mathcal{B}_k,\end{gathered} \tag{3.10}$$

for each $k \in [\![1, n-1]\!]$ (dual "variables" $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_n, \boldsymbol{\beta}_0, \boldsymbol{\beta}_n$ are constants employed to simplify the notation: see appendix B.4). 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.

### 3.4.2 Extending the Active Set

We initialise the dual (3.5) with a tight bound on the Big-M relaxation by solving for $d_\emptyset(\boldsymbol{\alpha}, \boldsymbol{\beta}_\emptyset)$ in (3.7) (appendix B.2). 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.

#### 3.4.2.1 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 (3.5). 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

[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 $k$.

$\boldsymbol{\beta}_{k,I_k^*}$ maximising the corresponding entries of the supergradient of the full dual problem (3.5), excluding those pertaining to $\mathcal{M}_k$:

$$\boldsymbol{\beta}_{k,I_k^*} \in \underset{\boldsymbol{\beta}_{k,I_k} \in\ \boldsymbol{\beta}_k \setminus \boldsymbol{\beta}_{\emptyset,k}}{\operatorname{argmax}} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}\}. \tag{3.11}$$

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 (3.4) by Anderson et al. (2020).

**Proposition 3.** *$\boldsymbol{\beta}_{k,I_k^*}$ as defined in equation (3.11) represents the Lagrangian multipliers associated to the most violated constraints from $\mathcal{A}_{\mathcal{E},k}$ at*

$$(\mathbf{x}^*, \mathbf{z}^*) \in \underset{\mathbf{x},\mathbf{z}}{\operatorname{argmin}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}}),$$

*the primal minimiser of the current restricted Lagrangian.*

*Proof.* The result can be obtained by noticing that $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ (by definition of Lagrangian multipliers) quantifies the corresponding constraint's violation at $(\mathbf{x}^*, \mathbf{z}^*)$. Therefore, maximizing $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ will amount to maximizing constraint violation. We demonstrate analytically that the process will, in fact, correspond to a call to oracle (3.4). Recall the definition of $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ in equation (3.8), which applies beyond the current active set:

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

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

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

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:

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

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 (3.4) 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 $\mathcal{A}_{\mathcal{E},k}$ at $(\mathbf{x}^*, \mathbf{z}^*)$. □

#### 3.4.2.2 Frequency of addition

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

**Fact 2.** *Assume we obtained a dual solution* $(\boldsymbol{\alpha}^\dagger, \boldsymbol{\beta}_{\mathcal{B}}^\dagger) \in \operatorname{argmax} d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$ *using Active Set on the current* $\mathcal{B}$. *Then* $(\mathbf{x}^*, \mathbf{z}^*) \in \operatorname{argmin}_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^\dagger, \boldsymbol{\beta}_{\mathcal{B}}^\dagger)$ *is not in general an optimal primal solution for the primal of the current variable-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 (3.3). While the primal cutting plane algorithm by Anderson et al. (2020) calls the separation oracle (3.4) 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 (3.4) repeatedly for a number of iterations.

We conclude this section by pointing out that, provided the algorithm is run to dual convergence on each variable-restricted dual problem (3.7), primal optima can be recovered by suitable modifications of the optimization routine

(Sherali and Choi, 1996). Then, if the dual variables corresponding to the most violated constraint at the primal optima are added to $\mathcal{B}_k$, Active Set mirrors the primal cutting plane algorithm, guaranteeing convergence to the solution of problem (3.5). In practice, as the main advantage of dual approaches (Dvijotham et al., 2018b; Bunel et al., 2020a) is their ability to quickly achieve tight bounds (rather than formal optimality), we rely on the heuristic strategy in Algorithm 3.

## 3.5 Saddle Point

For the Active Set solver (§3.4), we only consider settings in which $\boldsymbol{\beta}_{\mathcal{B}}$ is composed of a (small) constant number of variables. In fact, both its memory cost and time complexity per iteration are proportional to the cardinality of the active set. This mirrors the properties of the primal cutting algorithm by Anderson et al. (2020), for which memory and runtime will increase with the number of added constraints. As a consequence, the tightness of the attainable bounds will depend both on the computational budget and on the available memory. We remove the dependency on memory by presenting Saddle Point (Algorithm 4), a Frank-Wolfe type solver. By restricting the dual feasible space, Saddle Point is able to deal with all the exponentially many variables from problem (3.5), while incurring only a linear memory cost. We first describe the rationale behind the reduced dual domain (§3.5.1), then proceed to describe solver details (§3.5.2).

### 3.5.1 Sparsity via Sufficient Statistics

In order to achieve sparsity (Fact 1) without resorting to active sets, it is crucial to observe that all the appearances of $\boldsymbol{\beta}$ variables in $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, the Lagrangian of the full dual (3.5), can be traced back to the following linearly-sized *sufficient statistics*:

$$\boldsymbol{\zeta}_k(\boldsymbol{\beta}_k) = \begin{bmatrix} \sum_{I_k} \boldsymbol{\beta}_{k,I_k} \\ \sum_{I_{K+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}} \\ \sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1} \odot (\hat{\mathbf{l}}_k - \mathbf{b}_k) \\ \sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{U}_{k-1}) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot (\hat{\mathbf{u}}_k - \mathbf{b}_k) \end{bmatrix}. \tag{3.12}$$

Therefore, by designing a solver that only requires access to $\boldsymbol{\zeta}_k(\boldsymbol{\beta}_k)$ rather than to single $\boldsymbol{\beta}_k$ entries, we can incur only a linear memory cost.

In order for the resulting algorithm to be computationally efficient, we need to meet the anytime requirement in Fact 1 with a low cost per iteration. Let us refer to the evaluation of a neural network at a given input point $\mathbf{x}_0$ as a forward pass, and the backpropagation of a gradient through the network as a backward pass.

**Fact 3.** *The full dual objective $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ can be computed at the cost of a backward pass over the neural network if sufficient statistics $\boldsymbol{\zeta}(\boldsymbol{\beta}) = \cup_{k \in [\![1, n-1]\!]} \boldsymbol{\zeta}_k(\boldsymbol{\beta}_k)$ have been pre-computed.*

*Proof.* If $\boldsymbol{\zeta}(\boldsymbol{\beta})$ is up to date, the Lagrangian $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ can be evaluated using a single backward pass: this can be seen by replacing the relevant entries of (3.12) in equations (3.6) and (3.5). Similarly to the gradient backpropagation through the network, the bottleneck of the Lagrangian computation is then the layer-wise use of transposed linear operators over the $\boldsymbol{\alpha}$ dual variables. The minimization of the Lagrangian over primals can then be computed in linear time (less than the cost of a backward pass) by using equations (3.8), (3.9) with $\mathcal{B}_k = \mathcal{E}_k$ for each layer $k$. □

From Fact 3, we see that the full dual can be efficiently evaluated via $\boldsymbol{\zeta}(\boldsymbol{\beta})$. On the other hand, in the general case, $\boldsymbol{\zeta}(\boldsymbol{\beta})$ updates have an exponential time complexity. Therefore, we need a method that updates the sufficient statistics while computing a minimal number of terms of the exponentially-sized sums in (3.12). In other words, we need *sparse updates* in the $\boldsymbol{\beta}$ variables. With this goal in mind, we consider methods belonging to the Frank-Wolfe family (Frank and Wolfe, 1956), whose iterates are known to be sparse (Jaggi, 2013). In particular, we now illustrate that sparse updates can be obtained by applying the Saddle-Point Frank-Wolfe (SP-FW) algorithm by Gidel et al. (2017) to a suitably modified version of problem (3.5). Details of SP-FW and the solver resulting from its application are then presented in §3.5.2.

**Fact 4.** *Dual problem* (3.5) *can be seen as a bilinear saddle point problem. By limiting the dual feasible region to a compact set, a dual optimal solution for this domain-restricted problem can be obtained via SP-FW (Gidel et al., 2017). Moreover, a valid lower bound to* (3.3) *can be obtained at anytime by evaluating* $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ *at the current dual point from SP-FW.*

We make the feasible region of problem (3.5) compact by capping the cumulative price for constraint violations at some constants $\boldsymbol{\mu}$. In particular, we bound the $\ell_1$ norm for the sets of $\boldsymbol{\beta}$ variables associated to each neuron. As the $\ell_1$ norm is well-known to be sparsity inducing (Candès et al., 2008), our choice reflects the fact that, in general, only a fraction of the $\mathcal{A}_k$ constraints will be active at the optimal solution. Denoting by $\triangle(\boldsymbol{\mu}) = \cup_{k \in [\![1, n-1]\!]} \triangle_k(\boldsymbol{\mu}_k)$ the resulting dual domain, we obtain domain-restricted dual $\max_{(\boldsymbol{\alpha},\boldsymbol{\beta}) \in \triangle(\boldsymbol{\mu})} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$, which can be written as the following saddle point problem:

$$
\begin{aligned}
&\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\
&\text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \\
&\qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad\qquad k \in [\![1, n-1]\!]\,, \\
&\qquad \left.\begin{array}{l} \boldsymbol{\alpha}_k \in [\mathbf{0}, \boldsymbol{\mu}_{\alpha,k}] \\ \boldsymbol{\beta}_k \geq \mathbf{0},\ \|\boldsymbol{\beta}_k\|_1 \leq \boldsymbol{\mu}_{\beta,k} \end{array}\right\} := \triangle_k(\boldsymbol{\mu}_k) \qquad k \in [\![1, n-1]\!]\,,
\end{aligned}
\tag{3.13}
$$

where $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ was defined in equation (3.5) and $\|\cdot\|_1$ denotes the $\ell_1$ norm. Frank-Wolfe type algorithms move towards the vertices of the feasible region. Therefore, the shape of $\triangle_k(\boldsymbol{\mu}_k)$ is key to the efficiency of $\boldsymbol{\zeta}_k$ updates. In our case, $\triangle_k(\boldsymbol{\mu}_k)$ is the Cartesian product of a box constraint on $\boldsymbol{\alpha}_k$ and $n_k$ exponentially-sized simplices: one for each set $\boldsymbol{\beta}_k[i] = \{\boldsymbol{\beta}_{k,\text{row}_i(I_k)}[i] \ \forall\ \text{row}_i(I_k) \in 2^{\text{row}_i(W_k)}\}$. As a consequence, each vertex of $\triangle_k(\boldsymbol{\mu}_k)$ is sparse in the sense that at most $n_k + 1$ variables out of exponentially many will be non-zero. In order for the resulting solver to be useful in practice, we need to efficiently select a vertex towards which to move: we show in section 3.5.2 that our choice for $\triangle_k(\boldsymbol{\mu}_k)$ allows us to recover the linear-time primal oracle (3.4) by Anderson et al. (2020).

Before presenting the technical details of our Saddle Point solver, it remains to comment on the consequences of the dual space restriction on the obtained bounds.

**Algorithm 4** Saddle Point

**function** SADDLEPOINT_COMPUTE_BOUNDS(Problem (3.5))
  Initialise duals $\boldsymbol{\alpha}^0, \beta^0_{\mathcal{B}}$ using algorithm (5) or algorithm (3)
  Set $\beta^0_{\bar{\mathcal{B}}} = 0$, $\beta^0 = \beta^0_{\mathcal{B}} \cup \beta^0_{\bar{\mathcal{B}}}$, and replace $\beta^0$ by its sufficient statistics $\boldsymbol{\zeta}(\beta^0_k)$ using (3.12)
  Initialise primals $\mathbf{x}^0, \mathbf{z}^0$ according to §B.3.2
  Set price caps $\boldsymbol{\mu}$ heuristically as outlined in §B.3.1
  **for** $t \in [\![0, T-1]\!]$ **do**
    $\bar{\mathbf{x}}^t, \bar{\mathbf{z}}^t \leftarrow$ using (3.8),(3.9) with $\mathcal{B}_k = \mathcal{E}_k$ ▷ compute primal conditional gradient
    $\bar{\boldsymbol{\alpha}}^t, \boldsymbol{\zeta}(\bar{\beta}^t) \leftarrow$ (3.14), (3.15) + (3.12) ▷ compute dual conditional gradient
    $\mathbf{x}^{t+1}, \mathbf{z}^{t+1}, \boldsymbol{\alpha}^{t+1}, \boldsymbol{\zeta}(\beta^{t+1}) = (1-\gamma_t)[\mathbf{x}^t, \mathbf{z}^t, \boldsymbol{\alpha}^t, \boldsymbol{\zeta}(\beta^t)] + \gamma_t[\bar{\mathbf{x}}^t, \bar{\mathbf{z}}^t, \bar{\boldsymbol{\alpha}}^t, \boldsymbol{\zeta}(\bar{\beta}^t)]$
  **return** $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \boldsymbol{\zeta}(\beta^T))$

Let us define $d^*_{\boldsymbol{\mu}} = \max_{(\boldsymbol{\alpha},\beta)\in\triangle(\boldsymbol{\mu})} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$, the optimal value of the restricted dual problem associated to saddle point problem (3.13). Value $d^*_{\boldsymbol{\mu}}$ is attained at the dual variables from a saddle point of problem (3.13). As we restricted the dual feasible region, $d^*_{\boldsymbol{\mu}}$ will in general be smaller than the optimal value of problem (3.5). However, owing to the monotonicity of $d^*_{\boldsymbol{\mu}}$ over $\boldsymbol{\mu}$ and the concavity of $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$, we can make sure $\triangle(\boldsymbol{\mu})$ contains the optimal dual variables by running a binary search on $\boldsymbol{\mu}$. In practice, we heuristically determine the values of $\boldsymbol{\mu}$ from our dual initialisation procedure (see appendix B.3.1).

### 3.5.2 Solver

Algorithms in the Frank-Wolfe family proceed by taking convex combinations between the current iterate and a vertex of the feasible region. This ensures feasibility of the iterates without requiring projections. For SP-FW (Gidel et al., 2017), the convex combination is performed at once, with the same coefficient, for both primal and dual variables.

In the general case, denoting primal variables as $\mathbf{x}$, and dual variables as $\mathbf{y}$, each iteration of the SP-FW algorithm proceeds as follows: first, we compute the vertex $[\bar{\mathbf{x}}, \bar{\mathbf{y}}]$ towards which we take a step (*conditional gradient*). This is done by maximizing the inner product between the gradient and the variables over the feasible region for the dual variables, and by minimizing the inner product between the gradient and the variables over the feasible region for the primal variables. This operation is commonly referred to as the linear maximization oracle for dual variables, and linear minimization oracle for primal variables. Second, a step

size $\gamma_t \in [0, 1]$ is determined according to the problem specification. Finally, the current iterate is updated as $[\mathbf{x}, \mathbf{y}] \leftarrow (1 - \gamma_t)[\mathbf{x}, \mathbf{y}] + \gamma_t[\bar{\mathbf{x}}, \bar{\mathbf{y}}]$. We will now provide details for the instantiation of SP-FW in the context of problem (3.13), along with information concerning the solver initialization.

While Saddle Point relies on a primal-dual method operating on problem (3.13), our main goal is to compute anytime bounds to problem (3.3). As explained in §3.3, this is typically achieved in the dual domain. Therefore, as per Fact 4, we discard the primal variables from SP-FW and use the current dual iterate to evaluate $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ from problem (3.5).

#### 3.5.2.1 Conditional gradient computations

Due to the bilinearity of $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, the computation of the conditional gradient for the primal variables coincides with the inner minimisation in equations (3.8)-(3.9) with $\mathcal{B}_k = \mathcal{E}_k \ \forall \ k \in [\![1, n-1]\!]$.

Similarly to the primal variables, the linear maximisation oracle for the dual variables decomposes over the layers. The gradient of the Lagrangian over the duals, $\nabla_{\boldsymbol{\alpha},\boldsymbol{\beta}} \ \mathcal{L}$, is given by the supergradient in equation (3.10) if $\mathcal{B}_k = \mathcal{E}_k$ and the primal minimiser $(\mathbf{x}^*, \mathbf{z}^*)$ is replaced by the primals at the current iterate. As dual variables $\boldsymbol{\alpha}$ are box constrained, the linear maximisation oracle will drive them to their lower or upper bounds depending on the sign of their gradient. Denoting conditional gradients by bars, for each $k \in [\![1, n-1]\!]$:

$$\bar{\boldsymbol{\alpha}}_k = \boldsymbol{\mu}_{\alpha,k} \odot \mathbb{1}_{(W_k \mathbf{x}_{k-1} + \mathbf{b}_k - \mathbf{x}_k) \geq 0}. \tag{3.14}$$

The linear maximization for the exponentially many $\boldsymbol{\beta}_k$ variables is key to the solver's efficiency and is carried out on a Cartesian product of simplex-shaped sub-domains (see definition of $\triangle_k(\boldsymbol{\mu}_k)$ in (3.13)). Therefore, conditional gradient $\bar{\boldsymbol{\beta}}_k$ can be non-zero only for the entries associated to the largest gradients of each

simplex sub-domain. For each $k \in [\![1, n-1]\!]$, we have:

$$\bar{\boldsymbol{\beta}}_k = \left\{ \begin{array}{l} \bar{\boldsymbol{\beta}}_{k,I_k^\dagger} = \boldsymbol{\mu}_{\beta,k} \odot \mathbb{1}_{\nabla_{\boldsymbol{\beta}_{k,I_k^\dagger}} \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}) \geq 0} \\ \bar{\boldsymbol{\beta}}_{k,I_k} = \mathbf{0} \quad \forall\ I_k \in 2^{W_k} \setminus I_k^\dagger \end{array} \right\}, \tag{3.15}$$
$$\text{where:} \quad \boldsymbol{\beta}_{k,I_k^\dagger} \in \underset{\boldsymbol{\beta}_{k,I_k} \in \boldsymbol{\beta}_k}{\operatorname{argmax}} \left\{ \nabla_{\boldsymbol{\beta}_{k,I_k}} \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta})^T \mathbf{1} \right\}.$$

We can then efficiently represent $\bar{\boldsymbol{\beta}}_k$ through sufficient statistics as $\bar{\boldsymbol{\zeta}}_k = \boldsymbol{\zeta}_k(\bar{\boldsymbol{\beta}}_k)$, which will require the computation of a single term of the sums in (3.12).

**Proposition 4.** *$\boldsymbol{\beta}_{k,I_k^\dagger}$ as defined in* (3.15) *represents the Lagrangian multipliers associated to the most violated constraints from $\mathcal{A}_k$ at $(\mathbf{x}, \mathbf{z})$, the current SP-FW primal iterate. Moreover, the conditional gradient $\bar{\boldsymbol{\beta}}_k$ can be computed at the cost of a single call to the linear-time oracle* (3.4) *by Anderson et al. (2020).*

*Proof.* Let us define $\boldsymbol{\beta}_{k,I_k^*}$ as:

$$\boldsymbol{\beta}_{k,I_k^*} \in \underset{\boldsymbol{\beta}_{k,I_k} \in\ \boldsymbol{\beta}_k \setminus \boldsymbol{\beta}_{\emptyset,k}}{\operatorname{argmax}} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta})^T \mathbf{1}\}.$$

Proceeding as the proof of proposition 3, with $(\mathbf{x}, \mathbf{z})$ in lieu of $(\mathbf{x}^*, \mathbf{z}^*)$, we obtain that $\boldsymbol{\beta}_{k,I_k^*}$ is the set of Lagrangian multipliers for the most violated constraint from $\mathcal{A}_{\mathcal{E},k}$ at $(\mathbf{x}, \mathbf{z})$ and can be computed through the oracle (3.4) by Anderson et al. (2020).

Then, $\boldsymbol{\beta}_{k,I_k^\dagger}$ is computed as:

$$\boldsymbol{\beta}_{k,I_k^\dagger} \in \underset{\boldsymbol{\beta}_{k,I_k} \in\ \{\boldsymbol{\beta}_{k,I_k^*},\ \boldsymbol{\beta}_{k,0},\ \boldsymbol{\beta}_{k,1}\}}{\operatorname{argmax}} \nabla_{\boldsymbol{\beta}_{k,I_k}} \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta})^T \mathbf{1}.$$

As pointed out in the proof of proposition 3, the dual gradients of the Lagrangian correspond (by definition of Lagrangian multiplier) to constraint violations. Hence, $\boldsymbol{\beta}_{k,I_k^\dagger}$ is associated to the most violated constraint in $\mathcal{A}_k$ . □

#### 3.5.2.2 Convex combinations

The $(t+1)$-th SP-FW iterate will be given by a convex combination of the $t$-th iterate and the current conditional gradient. Due to the linearity of $\boldsymbol{\zeta}(\boldsymbol{\beta})$, we can

perform the operation via sufficient statistics. Therefore, all the operations of Saddle Point occur in the linearly-sized space of $(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\zeta}(\boldsymbol{\beta}))$:

$$[\mathbf{x}^{t+1}, \mathbf{z}^{t+1}, \boldsymbol{\alpha}^{t+1}, \boldsymbol{\zeta}(\boldsymbol{\beta}^{t+1})] = (1-\gamma_t)[\mathbf{x}^t, \mathbf{z}^t, \boldsymbol{\alpha}^t, \boldsymbol{\zeta}(\boldsymbol{\beta}^t)] + \gamma_t[\bar{\mathbf{x}}^t, \bar{\mathbf{z}}^t, \bar{\boldsymbol{\alpha}}^t, \bar{\boldsymbol{\zeta}}(\boldsymbol{\beta}^t)],$$

where, for our bilinear objective, SP-FW prescribes $\gamma_t = \frac{1}{1+t}$ (Gidel et al., 2017, section 5). Extensions of SP-FW, such as its away or pairwise step variants, require a worst-case memory cost that is linear in the number of iterations. In other words, as for Active Set, the attainable tightness would depend on the available memory, voiding one of the main advantages of Saddle Point (we provide empirical evidence of its memory efficiency in §3.8.2.2). Furthermore, the worst-case memory cost would increase more rapidly than for Active Set, which infrequently adds a few variables to $\mathcal{B}$ (in our experiments, $\mathcal{B}$ contains at most 7 variables per neuron: see §3.8.2). Finally, due to the bilinearity of the objective, these SP-FW variants do not correspond to an improved convergence rate (Gidel et al., 2017).

#### 3.5.2.3 Initialisation

As for the Active Set solver (§3.4), dual variables can be initialised via supergradient ascent on the set of dual variables associated to the Big-M relaxation (cf. appendix B.2). Additionally, if the available memory permits it, the initialization can be tightened by running Active Set (algorithm 3) for a small fixed number of iterations.

We mirror this strategy for the primal variables, which are initialized by performing subgradient descent on the primal view of saddle point problem (3.13). Analogously to the dual case, the primal view of problem (3.13) can be restricted to the Big-M relaxation for a cheaper initialization. Our primal initialization strategy is detailed in in appendix B.3.2.

## 3.6 Implementation Details, Technical Challenges

In this section, we present details concerning the implementation of our solvers. In particular, we first outline our parallelisation scheme and the need for a specialised convolutional operator (§3.6.1), then describe how to efficiently employ our solvers within branch and bound (§3.6.2).

### 3.6.1 Parallelism, masked forward/backward passes

Analogously to previous dual algorithms (Dvijotham et al., 2018b; Bunel et al., 2020a), our solvers 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 solvers rely on standard linear algebra operations employed during the forward and backward passes of neural networks, we can exploit the highly optimized implementations commonly found in modern deep learning frameworks.

An exception are what we call “masked" forward and backward passes. Writing convolutional operators in the form of their equivalent linear operator (as done in previous sections, see §3.2), masked passes take the following form:

$$(W_k \odot I_k)\, \mathbf{a}_k, \quad (W_k \odot I_k)^T\, \mathbf{a}_{k+1},$$

where $\mathbf{a}_k \in \mathbb{R}^{n_k}, \mathbf{a}_{k+1} \in \mathbb{R}^{n_{k+1}}$. Both operators are needed whenever dealing with constraints from $\mathcal{A}_k$. In fact, they appear in both Saddle Point (for instance, in the sufficient statistics (3.12)) and Active Set (see equations (3.7), (3.10)), except for the dual initialization procedure based on the easier Big-M problem (3.2).

Masked passes can be easily implemented for fully connected layers via Hadamard products. However, a customized lower-level implementation is required for a proper treatment within convolutional layers. In fact, from the high-level perspective, masking a convolutional pass corresponds to altering the content of the convolutional filter while it is being slided through the image. Details of our implementation can be found in appendix B.7.

### 3.6.2 Stratified bounding for branch and bound

In complete verification, we aim to solve the non-convex problem (3.1) directly, rather than an approximation like problem (3.3). In order to do so, we rely on branch and bound, which operates by dividing the problem domain into subproblems

(branching) and bounding the local minimum over those domains. The lower bound on the minimum is computed via a bounding algorithm, such as our solvers (§3.4, §3.5). The upper bound, instead, can be obtained by evaluating the network at an input point produced by the lower bound computation[3]. Any domain that cannot contain the global lower bound is pruned away, whereas the others are kept and branched over. The graph describing branching relations between subproblems is referred to as the enumeration tree. As tight bounding is key to pruning the largest possible number of domains, the bounding algorithm plays a crucial role. Moreover, it usually acts as a computational bottleneck for branch and bound (Lu and Kumar, 2020).

In general, tighter bounds come at a larger computational cost. The overhead can be linked to the need to run dual iterative algorithms for more iterations, or to the inherent complexity of tighter relaxations like problem (3.3). For instance, such complexity manifests itself in the masked passes described in appendix B.7, which increase the cost per iteration of Active Set and Saddle Point with respect to algorithms operating on problem (3.2). These larger costs might negatively affect performance on easier verification tasks, where a small number of domain splits with loose bounds suffices to verify the property. Therefore, as a general complete verification system needs to be efficient regardless of problem difficulty, we employ a *stratified* bounding system within branch and bound. Specifically, we devise a simple adaptive heuristic to determine whether a given subproblem is "easy" (therefore looser bounds are sufficient) or whether it is instead preferable to rely on tighter bounds.

Given a bounding algorithm $a$, let us denote its lower bound for subproblem $s$ as $l_a(s)$. Assume two different bounding algorithms, $a_l$ and $a_t$, are available: one inexpensive yet loose, the other tighter and more costly. At the root $r$ of the branch and bound procedure, we estimate $l_{a_t - a_l} = l_{a_t}(r) - l_{a_l}(r)$, the extent to

[3]For subgradient-type methods like Active Set, we evaluate the network at $\mathbf{x}_0^{*,T}$ (see algorithm 3), while for Frank-Wolfe-type methods like Saddle Point at $\mathbf{x}_0^T$ (see algorithm 4). 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 (3.1) with maximisation instead of minimisation.

which the lower bounds returned by $a_l$ can be tightened by $a_t$. While exploring the enumeration tree, we keep track of the lower bound increase from parent to child (that is, after splitting the subdomain) through an exponential moving average. We write $i(s)$ for the average parent-to-child tightening until subproblem $s$. Then, under the assumption that each subtree is complete, we can estimate $|s|_{a_l}$ and $|s|_{a_t}$, the sizes of the enumeration subtrees rooted at $s$ that would be generated by each bounding algorithm. Recall that, for verification problems the canonical form (Bunel et al., 2018), subproblems are discarded when their lower bound is positive. Given $p$, the parent of subproblem $s$, we perform the estimation as: $|s|_{a_l} = 2^{\frac{-l_{a_l}(p)}{i(s)}+1} - 1$, $|s|_{a_t} = 2^{\frac{-\left(l_{a_l}(p)+l_{a_t-a_l}\right)}{i(s)}+1} - 1$. Then, relying on $c_{a_t/a_l}$, a rough estimate of the relative overhead of running $a_t$ over $a_l$, we mark the subtree rooted at $s$ as hard if the reduction in tree size from using $a_t$ exceeds its overhead. That is, if $\frac{|s|_{a_l}}{|s|_{a_t}} > c_{a_t/a_l}$, the lower bound for $s$ and its children will be computed via algorithm $a_t$ rather than $a_l$.

## 3.7 Related Work

In addition to those described in §3.2, 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., 2018a), specific dual assignments (Wong and Kolter, 2018), dual formulations based on Lagrangian Relaxation (Dvijotham et al., 2018b) or Lagrangian Decomposition (Bunel et al., 2020a). The degree of tightness varies greatly: from looser relaxations associated to closed-form methods (Gowal et al., 2018a; 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 (§3.2.1) (Singh et al., 2018; Wang et al., 2018b). 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 B.6.2). The relaxation by Anderson et al. (2020) detailed in §3.2.2 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 §3.2.2. Therefore, the relationship between the two formulations mirrors the one between the Planet and Big-M relaxations (see appendix B.2.1). Our dual derivation and solvers can be adapted to operate on the projected relaxations. Furthermore, the formulation by Tjandraatmadja et al. (2020) allows for a propagation-based method ("FastC2V"). However, such an algorithm tackles only two constraints per neuron at once and might hence yield looser bounds than the Planet relaxation. In this work, we are interested in designing solvers that can operate on strict subsets of the feasible region from problem (3.2).

Specialized 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., 2020). Therefore, the design of similar solvers for other tight relaxations is an interesting line of future research. We contribute with two specialized dual solvers for the relaxation by Anderson et al. (2020). In what follows, we demonstrate empirically that by meeting the requirements of Fact 1 we can obtain large incomplete and complete verification improvements.

## 3.8 Experiments

We empirically demonstrate the effectiveness of our methods under two settings. First, we assess the speed and quality of our bounds compared to other bounding algorithms on incomplete verification (§3.8.2). Then, we examine whether our speed-accuracy trade-offs pay off within branch and bound (§3.8.3). Our implementation, which is available at `https://github.com/oval-group/oval-bab` as

part of the OVAL neural network verification framework, is based on Pytorch (Paszke et al., 2017).

### 3.8.1 Experimental Setting

We compare both against dual iterative methods and Gurobi, which we use as gold standard for LP solvers. The latter is employed for the following two baselines:

- **Gurobi Planet** means solving the Planet Ehlers (2017) relaxation of the network (a version of problem (3.2) for which $\mathbf{z}$ have been projected out).
- **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).

Both Gurobi-based methods make use of LP incrementalism (warm-starting) when possible. In the experiments of §3.8.2, 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. In addition, our experimental analysis comprises the following dual iterative methods:

- **BDD+**, the recent proximal-based solver by Bunel et al. (2020a), operating on a Lagrangian Decomposition dual of the Planet relaxation.
- **Active Set** denotes our supergradient-based solver for problem (3.3) (§3.4).
- **Saddle Point**, our Frank-Wolfe-based solver for problem (3.3) (§3.5).
- By keeping $\mathcal{B} = \emptyset$, Active Set reduces to **Big-M**, a solver for the non-projected Planet relaxation (appendix B.2), which is employed as dual initialiser to both Active Set and Saddle Point.
- **AS-SP** is a version of Saddle Point whose dual initialization relies on a few iterations of Active Set rather than on the looser Big-M solver, hence combining both our dual approaches.

As we operate on the same datasets employed by Bunel et al. (2020a), we omit both their supergradient-based approach and the one by Dvijotham et al. (2018b), 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., 2018a) and the one by Wong and Kolter (2018). 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. While dual iterative algorithms are specifically designed to take advantage of GPU acceleration (see §3.6.1), we additionally provide a CPU implementation of our solvers in order to complement the comparison with Gurobi-based methods.

Unless otherwise stated, experiments were run under the following setup: Ubuntu 16.04.2 LTS, on a single Nvidia Titan Xp GPU, except those based on Gurobi and the CPU version of our solvers. The latter were run on i7-6850K CPUs, utilising 4 cores for the incomplete verification experiments, and 6 cores for the more demanding complete verification setting.

### 3.8.2 Incomplete Verification

We evaluate the efficacy of our bounding algorithms in an incomplete verification setting by upper bounding the vulnerability to adversarial perturbations (Szegedy et al., 2014), measured as the difference between the logits associated to incorrect classes and the one corresponding to the ground truth, on the CIFAR-10 test set (Krizhevsky and Hinton, 2009). If the upper bound is negative, we can certify the network's robustness to adversarial perturbations.

#### 3.8.2.1 Speed-Accuracy Trade-Offs

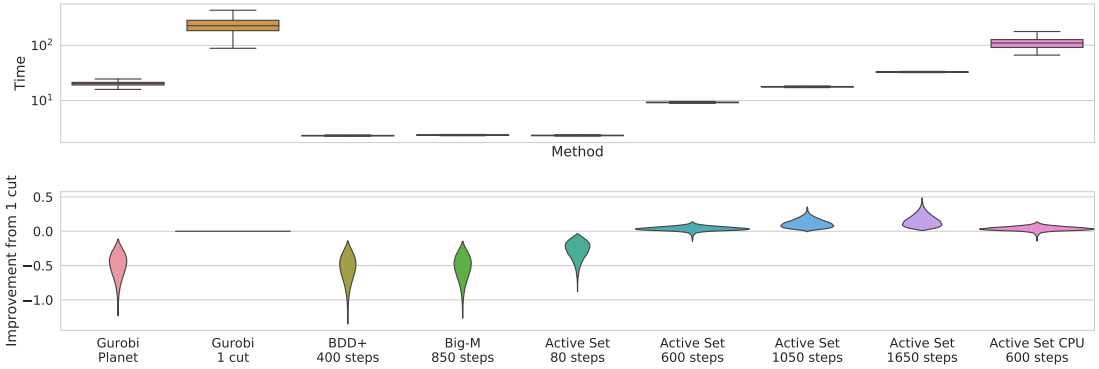
Here, we replicate the experimental setting from Bunel et al. (2020a). The networks correspond to the small network architecture from Wong and Kolter (2018), and to the "Wide" architecture, also employed for complete verification experiments in §3.8.3.1, found in Table 3.1. Due to the additional computational cost of bounds obtained via the tighter relaxation (3.3), we restricted the experiments to the first

2567 CIFAR-10 test set images for the experiments on the SGD-trained network (Figures 3.1, 3.2), and to the first 4129 images for the network trained via the method by Madry et al. (2018) (Figures B.3, B.4).

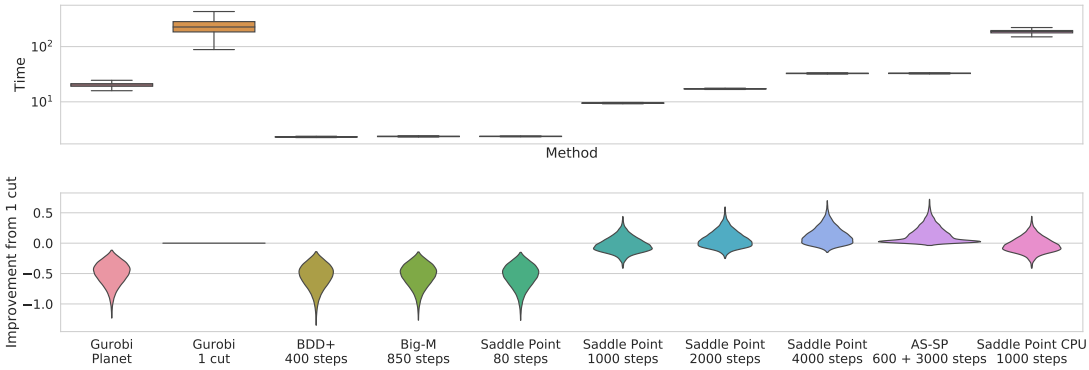
Here, we present results for a network trained via standard SGD and cross entropy loss, with no modification to the objective for robustness. Perturbations for this network lie in a $\ell_\infty$ norm ball with radius $\epsilon_{ver} = 1.125/255$ (which is hence lower than commonly employed radii for robustly trained networks). In appendix B.8, 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 trained with the verified training algorithm by Wong and Kolter (2018).

Solver hyper-parameters were tuned on a small subset of the CIFAR-10 test set. 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 algorithm 3) and we cap the maximum number of cuts to 7. This was found to be a good compromise between fast bound improvement and computational cost. For Saddle Point, we use $1/(t + 10)$ as step size to stay closer to the initialization points. The primal initialization algorithm (see appendix B.3.2) is run for 100 steps on the Big-M variables, with step size linearly decreased from $10^{-2}$ to $10^{-5}$.

Figure 3.1 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




**(a)** Speed-accuracy trade-offs of Active Set for different iteration ranges and computing devices.




**(b)** Speed-accuracy trade-offs of Saddle Point for different iteration ranges and computing devices.

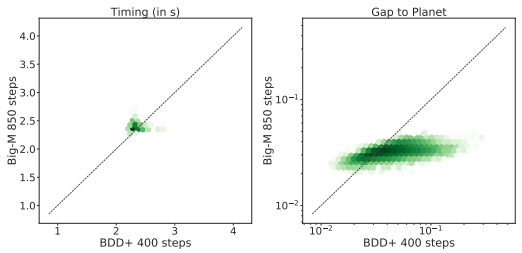
**Figure 3.1:** Upper bounds to the adversarial vulnerability for the SGD-trained network from Bunel et al. (2020a). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better, the width at a given value represents the proportion of problems for which this is the result. On average, both Active Set and Saddle Point achieve bounds tighter than Gurobi 1 cut with a smaller runtime.

relaxation (3.2), 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 a fraction of the time. Similar observations hold for Saddle Point which, especially when using fewer iterations, also exhibits a larger variance in terms of bounds tightness. In appendix B.8, we empirically demonstrate that the tightness of the Active Set bounds is strongly linked to our active set strategy (see §3.4.2). Remarkably, even if our solvers are specifically designed to take advantage

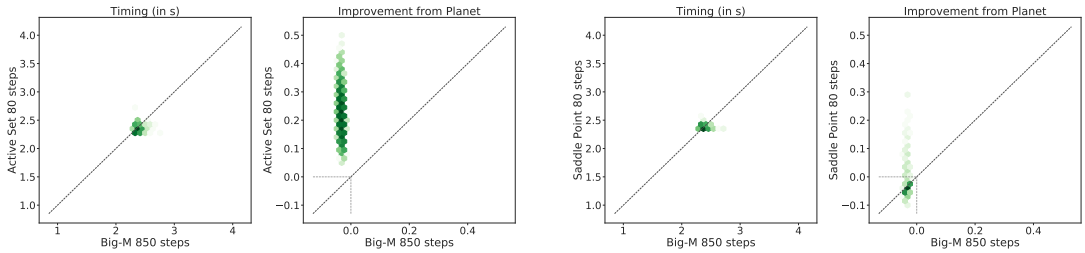
of GPU acceleration, executing them on CPU proves to be strongly competitive with Gurobi cut. Active Set produces better bounds in less time for the benchmark of Figure 3.1, while Saddle Point yields comparable speed-accuracy trade-offs.

**Small computational budget** Figure 3.2 shows pointwise comparisons for the less expensive methods from figure 3.1, on the same data. Figure 3.2(a) 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 3.2(b) shows the improvement over Planet bounds for Big-M compared to those of few (80) Active Set and Saddle Point iterations. Active Set returns markedly better bounds than Big-M in the same time, demonstrating the benefit of operating (at least partly) on the tighter dual (3.5). On the other hand, Saddle Point is rarely beneficial with respect to Big-M when running it for a few iterations.

**Larger computational budget** Figure 3.3 compares the performance of Active Set and Saddle Point for different runtimes, once again on the data from figure 3.1.




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




**(b)** Comparison of runtime (Timing) and difference with the Gurobi Planet bounds (Improvement from Planet). For the latter, higher is better.

**Figure 3.2:** Pointwise comparison for a subset of the methods on the data presented in figure 3.1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

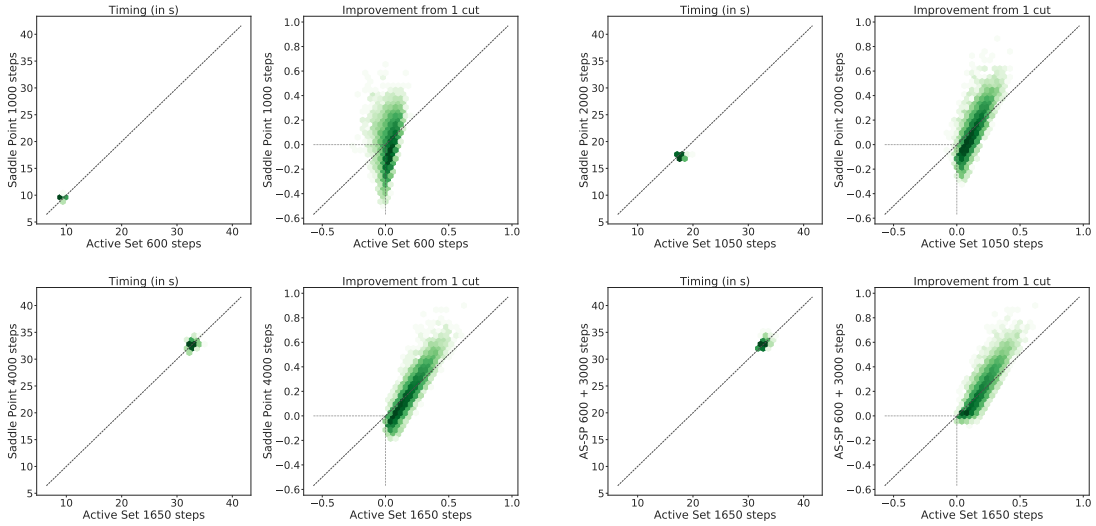



**Figure 3.3:** Pointwise comparison between our proposed solvers on the data presented in figure 3.1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

While Active Set yields better average bounds when fewer iterations are employed, the performance gap shrinks with increasing computational budgets, and AS-SP (Active Set initialization to Saddle Point) yields tighter average bounds than Active Set in the same time. Differently from Active Set, the memory footprint of Saddle Point does not increase with the number of iterations (see §3.5). Therefore, we believe the Frank-Wolfe-based algorithm is particularly convenient in incomplete verification settings that require tight bounds.

#### 3.8.2.2 Memory efficiency

As stated in §3.5, one of the main advantages of Saddle Point is its memory efficiency. In fact, differently from Active Set, the attainable bounding tightness will not depend on the available memory. In order to illustrate this, we present results on a large fully connected network with two hidden layers of width 7000. The network was adversarially (Madry et al., 2018) trained against perturbations of size $\epsilon = 2/255$, which is the same radius that we employ at verification time. On the Nvidia Titan Xp GPU employed for our experiments, Active Set is only able to include a single constraint from $\mathcal{A}_{\mathcal{E},k}$ per neuron without running out of memory. Except the maximum allowed number of cuts for Active Set, we run all algorithms

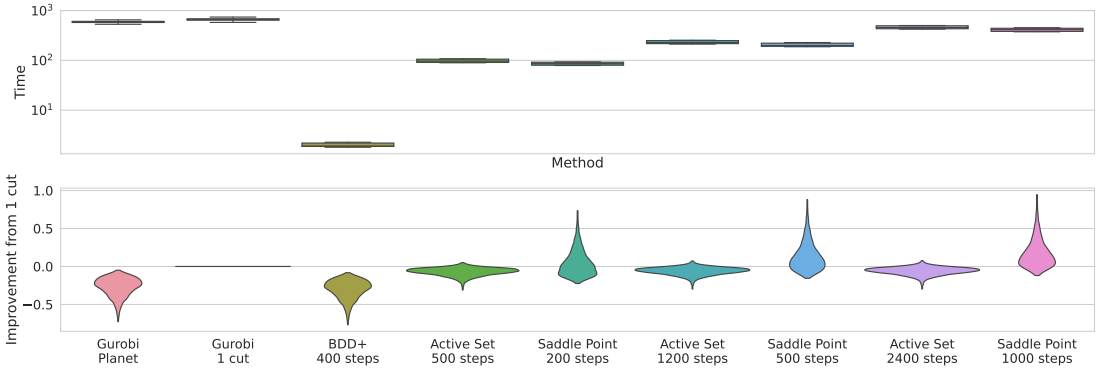



**Figure 3.4:** Upper bounds to the adversarial vulnerability of a fully connected network with two hidden layers of width 7000. Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. When run for enough iterations, Saddle Point achieves bounds tighter than both Gurobi 1 cut and Active Set, whose tightness is constrained by memory, in less time.

with the same hyper-parameters as in §3.8.2.1. We conduct the experiment on the first 500 images of the CIFAR-10 test set. Figure 3.4 shows that, while Active Set is competitive with Saddle Point when both are run for a few iterations, Saddle Point yields significantly better speed-accuracy trade-offs when both algorithms are run for longer. Indeed, the use of a single tightening constraint per neuron severely limits the tightness attainable by Active Set, which yields looser bounds than Gurobi cut on average. This is dissimilar from Figure 3.1, where Active Set rapidly overcomes Gurobi cut in terms of bounding tightness. On the other hand, Saddle Point returns bounds that are markedly tighter than those from the primal baselines in a fraction of their runtime, highlighting its benefits in memory-intensive settings.

### 3.8.3 Branch and Bound

We now assess the effectiveness of our algorithms within branch and bound (see §3.6.2). In particular, we will employ them within BaBSR (Bunel et al., 2020b). In BaBSR, branching is carried out by splitting an unfixed ReLU into its passing and blocking phases (see §3.6.2 for a description of branch and bound). In order to determine which ReLU to split on, BaBSR employs an inexpensive heuristic based on the bounding algorithm by Wong and Kolter (2018). The goal of the

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

**Table 3.1:** 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.

heuristic is to assess which ReLU induces maximum change in the domain's lower bound when made unambiguous.

#### 3.8.3.1 Complete Verification

We evaluate the performance on complete verification by verifying the adversarial robustness of a network to perturbations in $\ell_\infty$ norm on a subset of the dataset by Lu and Kumar (2020). We replicate the experimental setting from Bunel et al. (2020a).

**Dataset** Lu and Kumar (2020) provide, for a subset of the CIFAR-10 test set, a verification radius $\epsilon_{ver}$ 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 $\epsilon_{ver}$ radius was tuned to meet a certain "problem difficulty" via binary search, employing a Gurobi-based bounding algorithm (Lu and Kumar, 2020). In particular, Lu and Kumar (2020) chose perturbation radii to rule out properties for which an adversarial example can be rapidly found, and properties for which Gurobi Planet would be able to prove robustness without any branching. This characteristic makes the dataset an appropriate testing ground for tighter relaxations like the one by Anderson et al. (2020) (§3.2.2). The networks are robust on all the properties we employed. Three different network architectures of different sizes are used, all robustly trained for $\epsilon_{train} = 2/255$ with the algorithm by Wong and Kolter (2018). A "Base" network with 3172 ReLU activations, and two networks with roughly twice as many activations: one "Deep", the other "Wide". Details can be found in Table 3.1. We restricted the original dataset to 100 properties per network so as to mirror the setup of the recent VNN-COMP competition (VNN-COMP, 2020). The properties have an average perturbation radius of $\epsilon_{ver} = {}^{10.1}/_{255}$, $\epsilon_{ver} = {}^{6.9}/_{255}$, $\epsilon_{ver} = {}^{7.1}/_{255}$ for the Base, Wide, and Deep networks, respectively.

**Complete Verifiers** We compare the effect on final verification time of using the different bounding methods in §3.8.2 within BaBSR. When stratifying two bounding algorithms (see §3.6.2) we denote the resulting method by the names of both the looser and the tighter bounding method, separated by a plus sign (for instance, **Big-M + Active Set**). In addition, we compare against the following complete verification algorithms:

- **MIP** $\mathcal{A}_k$ encodes problem (3.1) as a Big-M MIP (Tjeng et al., 2019) and solves it in Gurobi by adding cutting planes from $\mathcal{A}_k$. This mirrors the original experiments from Anderson et al. (2020).
- **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[4] (VNN-COMP, 2020).

[4]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.

| | Base | | | Wide | | | Deep | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | time(s) | sub-problems | %Timeout | time(s) | sub-problems | %Timeout | time(s) | sub-problems | %Timeout |
| BDD+ BaBSR | 883.55 | 82 699.40 | 22.00 | 568.25 | 43 751.88 | 13.00 | 281.47 | 10 763.48 | 5.00 |
| Big-M BaBSR | 826.60 | 68 582.00 | 19.00 | 533.79 | 35 877.24 | 12.00 | 253.37 | 9346.78 | 4.00 |
| A. Set 100 it. BaBSR | 422.32 | **9471.90** | **7.00** | 169.73 | **1873.36** | **3.00** | 227.26 | 2302.16 | 2.00 |
| Big-M + A. Set 100 it. BaBSR | **415.20** | 10 449.10 | **7.00** | **163.02** | 2402.28 | **3.00** | 199.70 | 2709.60 | 2.00 |
| G. Planet + G. 1 cut BaBSR | 949.06 | 1572.10 | 15.00 | 762.42 | 514.02 | 6.00 | 799.71 | 391.70 | 2.00 |
| MIP $\mathcal{A}_k$ | 3227.50 | 226.24 | 82.00 | 2500.70 | 100.93 | 64.00 | 3339.37 | 434.57 | 91.00 |
| ERAN | 805.89 | - | 5.00 | 632.12 | - | 9.00 | 545.72 | - | 0.00 |
| Fast-and-Complete | 711.63 | 9801.22 | 16.00 | 350.57 | 8699.74 | 8.00 | **56.52** | **2238.52** | **1.00** |

**Table 3.2:** 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.

- **Fast-and-Complete** (Xu et al., 2021): a recent complete verifier based on BaBSR that pairs fast dual bounds with Gurobi Planet to obtain state-of-the-art performance.

We use 100 iterations for BDD+ (as done by Bunel et al. (2020a)) and 180 for Big-M, which was tuned to employ roughly the same time per bounding computation as BDD+. We re-employed the same hyper-parameters for Big-M, Active Set and Saddle Point, except the number of iterations. For dual iterative algorithms, we solve 300 subproblems at once for the base network and 200 for the deep and wide networks (see §3.6.1). 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.

**Inexpensively overcoming the convex barrier** Figure 3.2(b) in the previous section shows that Active Set can yield a relatively large improvement over Gurobi Planet bounds, the convex barrier as defined by Salman et al. (2019b), in a few

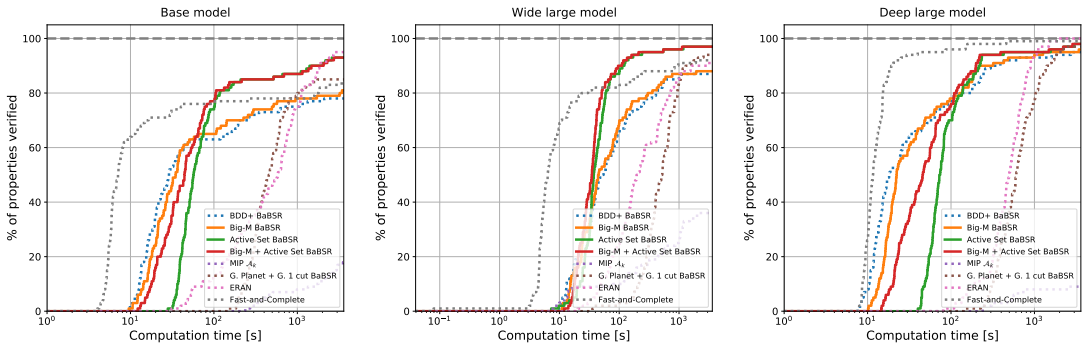



**Figure 3.5:** Cactus plots on properties from Lu and Kumar (2020), displaying the percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

iterations. In light of this, we start our experimental evaluation by comparing the performance of 100 iterations of Active Set (within BaBSR) with the relevant baselines. Figure 3.5 and Table 3.2 show that Big-M performs competitively with BDD+. With respect to BDD+ and Big-M, which operate on the looser formulation (3.2), Active Set verifies a larger share of properties and yields faster average verification. This demonstrates the benefit of tighter bounds (§3.8.2) 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 some of the speed of Big-M on easier properties, without sacrificing Active Set's gains on the harder ones. 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. Fast-and-Complete performs particularly well on the easier properties and on the Deep model, where it is the fastest algorithm. Nevertheless, it struggles on the harder properties, leading to larger average verification times and more timeouts than Active Set on the Base and Wide models. We believe that stratifying Active Set on the inexpensive dual bounds from Fast-and-Complete, which fall short of the convex barrier yet jointly tighten the intermediate bounds, would preserve the advantages of both methods. 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.

**Varying speed-accuracy trade-offs** The results in Figure 3.5 demonstrate that a small yet inexpensive tightening of the Planet bounds yields large complete verification improvements. We now investigate the effect of employing even tighter, yet more costly, bounds from our solvers. To this end, we compare 100

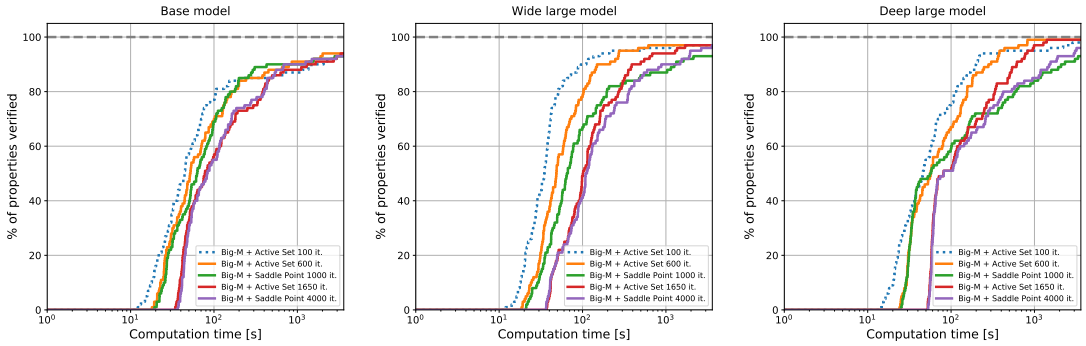



**Figure 3.6:** Cactus plots on properties from Lu and Kumar (2020), displaying the percentage of solved properties as a function of runtime. Comparison of best performing method from figure 3.5 with tighter bounding schemes.

iterations of Active Set with more expensive bounding schemes from our incomplete verification experiments (§3.8.2). All methods were stratified along Big-M to improve performance on easier properties (see §3.6.2) and employed within BaBSR. Figure 3.6 and Table 3.3 show results for two different bounding budgets: 600 iterations of Active Set or 1000 of Saddle Point, 1650 iterations of Active Set or 4000 of Saddle Point (see figure 3.3). Despite their ability to prune more subproblems, due to their large computational cost, tighter bounds do not necessarily correspond to shorter overall verification times. Running Active Set for 600 iterations of Active Set leads to faster verification of the harder properties while slowing it down for the easier ones. On the base and deep model, the benefits lead to a smaller average runtime (Table 3.3). This does not happen on the wide network, for which not enough subproblems are pruned. On the other hand, 1650 Active Set iterations do not prune enough subproblems for their computational overhead, leading to slower formal verification. The behavior of Saddle Point mirrors what was seen for incomplete

| | Base | | | Wide | | | Deep | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | time(s) | sub-problems | %Timeout | time(s) | sub-problems | %Timeout | time(s) | sub-problems | %Timeout |
| Big-M + Active Set 100 it. | 415.20 | 10 449.10 | 7.00 | **163.02** | 2402.28 | **3.00** | 199.70 | 2709.60 | 2.00 |
| Big-M + Active Set 600 it. | **360.16** | **4806.14** | **6.00** | 181.27 | **1403.90** | **3.00** | **148.06** | **1061.90** | **1.00** |
| Big-M + Saddle Point 1000 it. | 382.15 | 5673.04 | 7.00 | 417.41 | 1900.90 | 7.00 | 540.68 | 2551.62 | 7.00 |
| Big-M + Active Set 1650 it. | 463.00 | 7484.54 | **6.00** | 285.99 | 1634.98 | **3.00** | 250.52 | 1119.00 | **1.00** |
| Big-M + Saddle Point 4000 it. | 434.92 | 4859.68 | 7.00 | 402.86 | 1703.48 | **3.00** | 482.93 | 1444.20 | 4.00 |

**Table 3.3:** 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 result is highlighted in bold. Comparison of best performing method from figure 3.5 with tighter bounding schemes within BaBSR.

verification: while it does not perform as well as Active Set for small computational budgets, the gap shrinks when the algorithms are run for more iterations and it is very competitive with Active Set on the base network. Keeping in mind that the memory cost of each variable addition to the active set is larger in complete verification due to subproblem batching (see 3.6.1), Saddle Point constitutes a valid complete verification alternative in settings where memory is critical.

#### 3.8.3.2 Branch and Bound for Incomplete Verification

When run for a fixed number of iterations, branch and bound frameworks can be effectively employed for incomplete verification. As shown in §3.8.3.1, one of the main advantages of our algorithms is their performance within branch and bound. We provide further evidence of this by comparing them with previous algorithms that overcame the convex barrier, but which were not designed for employment within branch and bound: **kPoly** by Singh et al. (2019a), **Fast2CV** and **Opt2CV** by Tjandraatmadja et al. (2020). In particular, we compute the number of verified images on the first 1000 examples of the CIFAR-10 test set for the `ConvSmall` network from the ERAN (Singh et al., 2020) dataset, excluding misclassified images. We test different speed-accuracy trade-offs within branch and bound: BDD+ BaBSR is run for 400 iterations, and at most 5 branch and bound batches. Active Set and Saddle Point are run for 200 and 4000 iterations, respectively, with at most 10 batches within BaBSR. Hyper-parameters are kept as in §3.8.2. This experiment is run on a single Nvidia Titan V GPU. Table 3.4 shows that, regardless of the employed speed-accuracy trade-off, the use of branch and bound is beneficial with respect to kPoly, Fast2CV, and Opt2CV. Therefore, the use of relaxations tighter than

| | kPoly | FastC2V | OptC2V | BDD+ BaBSR | Active Set BaBSR | Saddle Point BaBSR |
|---|---|---|---|---|---|---|
| Verified Properties | 399 | 390 | 398 | 401 | 434 | 408 |
| Average Runtime [s] | 86 | 15.3 | 104.8 | 2.5 | 7.0 | 43.16 |

**Table 3.4:** Number of verified properties and average runtime on the adversarially trained (Madry et al., 2018) `ConvSmall` network from the ERAN (Singh et al., 2020) dataset, on the first 1000 images of the CIFAR-10 test set. Results for FastC2V and OptC2V are taken from Tjandraatmadja et al. (2020), results for kPoly are taken from Singh et al. (2019a).

Planet does not necessarily improve incomplete verification performance. Increasing the computational budget of branch and bound increases the number of verified properties, as demonstrated by the performance of Active Set and Saddle Point.

## 3.9 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 the complexity of the standard solvers for such relaxations hinders their applicability. We have presented two different sparse dual solvers for one such relaxation, and empirically demonstrated that they can yield significant formal verification speed-ups. Our results show that tightness, when paired with scalability, 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.

# 4

# IBP Regularization for Verified Adversarial Robustness via Branch-and-Bound

## Contents


4.1 Introduction . . . . . **101**
4.2 Background . . . . . **102**
4.2.1 Neural Network Verification . . . . . 103
4.2.2 Training via the Robust Loss . . . . . 104
4.2.3 Hybrid Training Methods . . . . . 105
4.3 Training via IBP Regularization . . . . . **106**
4.4 Verification Framework . . . . . **108**
4.4.1 Branch–and-Bound Setup . . . . . 108
4.4.2 UPB Branching . . . . . 108
4.5 Related Work . . . . . **109**
4.6 Experiments . . . . . **111**
4.6.1 Verified Training . . . . . 111
4.6.2 Branching . . . . . 112
4.7 Conclusions . . . . . **113**

# Abstract

Recent works have tried to increase the verifiability of adversarially trained networks by running the attacks over domains larger than the original perturbations and adding various regularization terms to the objective. However, these algorithms either underperform or require complex and expensive stage-wise training procedures, hindering their practical applicability. We present IBP-R, a novel verified training algorithm that is both simple and effective. IBP-R induces network verifiability by coupling adversarial attacks on enlarged domains with a regularization term, based on inexpensive interval bound propagation, that minimizes the gap between the non-convex verification problem and its approximations. By leveraging recent branch-and-bound frameworks, we show that IBP-R obtains state-of-the-art verified robustness-accuracy trade-offs for small perturbations on CIFAR-10 while training significantly faster than relevant previous work. Additionally, we present UPB, a novel branching strategy that, relying on a simple heuristic based on $\beta$-CROWN, reduces the cost of state-of-the-art branching algorithms while yielding splits of comparable quality.

## 4.1 Introduction

The existence of adversarial examples (Szegedy et al., 2014; Goodfellow et al., 2015) has raised widespread concerns on the robustness of neural networks. As a consequence, many authors have promptly devised algorithms to formally prove the robustness of trained networks (Katz et al., 2017; Ehlers, 2017; Bunel et al., 2018; Zhang et al., 2018; Raghunathan et al., 2018). At the same time, a number of works have focused on training networks for adversarial robustness: first by defending against specific attacks (adversarial training) (Madry et al., 2018), then providing formal guarantees about attack-independent robustness (verified training) (Dvijotham et al., 2018a; Wong and Kolter, 2018; Mirman et al., 2018). The vast majority of verified training methods operate by backpropagating over over-approximations of the network's loss under adversarial perturbations, and obtain state-of-the-art results for large perturbations (Zhang et al., 2020; Xu et al., 2020; Lyu et al., 2021). However, these training schemes are typically unable to benefit from tight over-approximations at verification time, hence requiring relatively large networks to perform at their best. A recent line of work has better leveraged network capacity by enhancing the verifiability of adversarially-trained networks. These algorithms can exploit tighter over-approximations but they either underperform (Xiao et al., 2019) or require expensive procedures in order to reach state-of-the-art performance on small perturbations (Balunovic and Vechev, 2020).

The recent VNN-COMP-21, an international competition on neural network verification (Bak et al., 2021) highlighted significant scaling improvements in exact verification algorithms (Henriksen and Lomuscio, 2021; Serre et al., 2021; De Palma et al., 2021c; Wang et al., 2021a). We aim to leverage these developments by presenting IBP-R, a novel and inexpensive verified training algorithm that induces network verifiability by: (i) running adversarial attacks over domains that are significantly larger than the target perturbations, (ii) exploiting IBP (Mirman et al., 2018; Gowal et al., 2018b) to minimize the area of the convex hull of the activations, a commonly employed relaxation within recent verification frameworks. We show that, in spite of its speed and conceptual simplicity, IBP-R yields state-of-the-art results under

small perturbations on CIFAR-10. In particular, under $\ell_\infty$ perturbations of radius $\epsilon_{\text{ver}} = 2/255$, networks trained via IBP-R attain, on average: a verified accuracy of 61.97%, a robust accuracy of 66.39% under MI-FGSM attacks (Dong et al., 2018) and a natural accuracy of 78.19%. In this setting, IBP-R trains in less than a third of the runtime of COLT (Balunovic and Vechev, 2020). Furthermore, for $\epsilon_{\text{ver}} = 8/255$, IBP-R performs competitively with COLT while almost halving its runtime.

Finally, motivated by the task of evaluating the verifiability of networks trained via IBP-R, we present a simple and novel branching strategy, named UPB, for complete verification via branch-and-bound (Bunel et al., 2018). UPB leverages dual information from the recent $\beta$-CROWN algorithm (Wang et al., 2021a) to heuristically rank the quality of the possible branching decisions. We show that, at a cost equivalent to a single gradient backpropagation through the network, UPB obtains a verification performance comparable to the more expensive and state-of-the-art FSB strategy (De Palma et al., 2021c).

## 4.2 Background

In the following, we will use boldface letters to denote vectors (for example, $\mathbf{x}$), uppercase letters to denote matrices (for example, $W_k$), and brackets for intervals (for example, $[\mathbf{l}_k, \mathbf{u}_k]$). Furthermore, we will write $\odot$ for the Hadamard product, $\mathbb{1}_{\mathbf{c}}$ for the indicator vector on condition $\mathbf{c}$, $[\![\cdot]\!]$ for integer ranges, and employ the following shorthand: $[\mathbf{x}]_+ := \max(\mathbf{x}, \mathbf{0})$.

Let us define the data distribution $\mathcal{D}$, yielding points $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^d \times \mathbb{R}^o$ and let us denote by $\boldsymbol{\theta} \in \mathbb{R}^S$ the network parameters. Robust training is concerned with training a neural network $f : \mathbb{R}^S \times \mathbb{R}^d \to \mathbb{R}^o$ so that a given property $P : \mathbb{R}^o \times \mathbb{R}^o \to \{0, 1\}$ is satisfied in a region around each input point $\mathbf{x}$, denoted $\mathcal{C}(\mathbf{x})$. In other words, the parameters $\boldsymbol{\theta}$ must satisfy the following:

$$\mathbf{x}_0 \in \mathcal{C}(\mathbf{x}) \implies P(f(\boldsymbol{\theta}, \mathbf{x}_0), \mathbf{y}) \quad \forall\ (\mathbf{x}, \mathbf{y}) \in \mathcal{D}. \tag{4.1}$$

In this work, we will focus on robustness to adversarial perturbations around the input images. Specifically, $\mathcal{C}(\mathbf{x}) := \{\mathbf{x}_0 : \|\mathbf{x}_0 - \mathbf{x}\|_p \leq \epsilon_{\text{ver}}\}$ and $P$ amounts to checking that the predicted and ground truth classification labels match.

### 4.2.1 Neural Network Verification

Before delving into the task of training a robust network, we first consider the problem of determining whether a given network is robust or not. This involves the formal verification of condition (4.1) on the given network, which is generally NP-HARD (Katz et al., 2017). Therefore, its exact verification is often replaced by less expensive approximations, which nevertheless provide formal guarantees for a subset of the properties (incomplete verification). By means of simple transformations, one can represent both $f$ and $P$ from condition (4.1) via a single network $f'$ of depth $n$ (Bunel et al., 2020b), so that incomplete verification corresponds to the following optimization problem:

$$\begin{aligned} \min_{\mathbf{x},\hat{\mathbf{x}}} \quad \hat{x}_n \qquad \text{s.t.} \qquad & \mathbf{x}_0 \in \mathcal{C}(\mathbf{x}), \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} \quad k \in [\![0, n-1]\!], \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in \text{Rel}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) \quad k \in [\![1, n-1]\!], \\ & \hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \qquad\qquad k \in [\![1, n-1]\!], \end{aligned} \tag{4.2}$$

where $\sigma$ denotes the activation function, $W_k$ the weight matrix of the $k$-th layer of $f'$, $\mathbf{b}_k$ its bias. The use of $\text{Rel}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$, a convex relaxation of $\sigma$, ensures that problem (4.2) is convex, greatly simplifying its solution. In general, $\text{Rel}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ is a function of intermediate bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$, which provide ranges on the network pre-activation variables (for details, see appendix C.3). In the context of ReLU networks, which are the focus of this work, a popular relaxation choice is the convex hull of the activation, commonly referred to as the Planet relaxation (Ehlers, 2017). If $\hat{\mathbf{l}}_k < 0$ and $\hat{\mathbf{u}}_k > 0$ (ambiguous ReLU), its shape is given by Figure 4.1. If either $\hat{\mathbf{l}}_k > 0$ or $\hat{\mathbf{u}}_k < 0$, the activation is said to be stable and its convex hull can be represented by a line, greatly improving the tightness of the overall network approximation.

When verifying all properties is a requirement (complete verification), problem (4.2) is employed as a sub-routine for a global optimization algorithm equivalent to

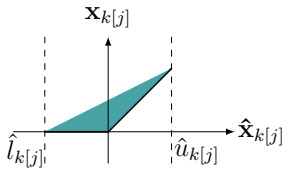



**Figure 4.1:** Convex hull for an ambiguous ReLU (Ehlers, 2017).

branch-and bound (Bunel et al., 2018). The goal is to find the sign of the minimum of a non-convex problem (reported in appendix C.2) whose domain is a subset of the feasible region from problem (4.2). Complete verifiers hence proceed by recursively splitting the domain (branching) and solving the resulting convex sub-problems (bounding) until a definite answer can be provided. For ReLU activations, the branching is usually performed by splitting the domain of an ambiguous ReLU into its two stable subdomains.

### 4.2.2 Training via the Robust Loss

In order to enforce condition (4.1) during training, one typically defines a surrogate loss $\mathcal{L} : \mathbb{R}^o \times \mathbb{R}^o \to \mathbb{R}$, and seeks to minimize the worst-case empirical risk within $\mathcal{C}(\mathbf{x})$, referred to as the robust loss:

$$\min_{\boldsymbol{\theta}} \; \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left[ \max_{\mathbf{x}'\in\mathcal{C}(\mathbf{x})} \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}'), \mathbf{y}) \right]. \tag{4.3}$$

The exact computation of $\max_{\mathbf{x}'\in\mathcal{C}(\mathbf{x})} \mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}'), \mathbf{y})$ entails the use of a complete verification algorithm (§4.2.1), which is too expensive to be employed during training. Therefore, the robust loss is typically replaced by an approximation: adversarial training algorithms (Madry et al., 2018) rely on lower bounds, while certified training algorithms (Gowal et al., 2018b; Zhang et al., 2020) employ upper bounds. Lower bounds are computed by using so-called adversarial attacks: algorithms, such as PGD (Madry et al., 2018), that heuristically search for misclassified (adversarial) examples in the input space. Upper bounds are instead computed by solving an instance of problem (4.2) where $\text{Rel}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ typically represents the IBP hyper-rectangle (Mirman et al., 2018) or linear bounds on the activation (Wong and Kolter, 2018; Zhang et al., 2018). Adversarial training yield models with strong standard accuracy and empirical robustness. However, such

robustness is often hard to demonstrate via a formal verification method, and might potentially break under stronger attacks. Stronger robustness guarantees are instead provided by certified training algorithms, at the expense of the standard network accuracy and with longer training times.

### 4.2.3 Hybrid Training Methods

A line of recent work seeks to bridge the gap between adversarial and certified training by modifying the regions over which the attacks are performed and adding specialized regularization terms.

Xiao et al. (2019) demonstrate that the verified robust accuracy of PGD-trained networks (Madry et al., 2018) can be increased by adding $\ell_1$ regularization and a term encouraging ReLU stability to problem (4.3):

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left[ \max_{\mathbf{x}'\in\mathcal{C}(\mathbf{x})} \mathcal{L}(f(\boldsymbol{\theta},\mathbf{x}'),\mathbf{y}) + \lambda \left\|\boldsymbol{\theta}\right\|_1 + \rho \sum_{j=1}^{k} \tanh(1 - \hat{\mathbf{l}}_k(\boldsymbol{\theta}) \odot \hat{\mathbf{u}}_k(\boldsymbol{\theta}))^T \mathbf{1} \right],$$

where $\hat{\mathbf{l}}_k(\boldsymbol{\theta})$ and $\hat{\mathbf{u}}_k(\boldsymbol{\theta})$, which depend on the network parameters, are computed via a tightened version of IBP.

Balunovic and Vechev (2020) propose to employ adversarial training layer-wise, running the attacks over convex outer-approximations of frozen subsets of the network. Let us denote by $f^j$ a subset of network $f$ that starts at the $j$-th layer, and by $\boldsymbol{\theta}^j$ its parameters. Furthermore, let $\mathbb{C}_j(\mathbf{x})$ represent an outer-approximation of the $j$-th latent space obtained via the zonotope relaxation (Zhang et al., 2018), relying on zonotope intermediate bounds approximated via Cauchy random projections (Li et al., 2007). Convex Layer-wise Adversarial Training (COLT), operates on the following objective at the $j$-th stage of the training:

$$\min_{\boldsymbol{\theta}^j} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \begin{cases} \kappa \begin{bmatrix} \max_{\mathbf{x}'\in\mathbb{C}_j(\mathbf{x})} \mathcal{L}(f^j(\boldsymbol{\theta}^j,\mathbf{x}'),\mathbf{y}) \\ +\rho_j \left[-\hat{\mathbf{l}}_{j+1}(\boldsymbol{\theta}^j)\right]_+^T \left[\hat{\mathbf{u}}_{j+1}(\boldsymbol{\theta}^j)\right]_+ \end{bmatrix} \\ +(1-\kappa) \max_{\mathbf{x}'\in\mathbb{C}_{j-1}(\mathbf{x})} \mathcal{L}(f^{j-1}(\boldsymbol{\theta}^{j-1},\mathbf{x}'),\mathbf{y}) \\ +\lambda \left\|\boldsymbol{\theta}^j\right\|_1, \end{cases} \tag{4.4}$$

where $[-\hat{\mathbf{l}}_{j+1}]_+^T [\hat{\mathbf{u}}_{j+1}]_+$ is a regularizer for the $(j+1)$-th latent space, inducing ReLU stability and minimizing the area of the zonotope relaxation for ambiguous

ReLUs. As when computing intermediate bounds for $\mathbb{C}_j(\mathbf{x})$, the regularizer is computed via approximate zonotope bounds. In order to gradually transition from one training stage to the other, $\kappa$ is linearly increased from 0 to 1 in the first phase of the training. At the first stage ($j = 0$), the loss transitions from the natural loss (without any adversarial component) to the regularized PGD loss. COLT performs particularly well for smaller perturbation radii, for which it yields state-of-the-art results. However, its complexity and stage-wise nature make it relatively hard to deploy in practice. For instance, Balunovic and Vechev (2020) employ a different value for both $\rho_j$ and the train-time perturbation radius, which affects both $\mathbb{C}_j(\mathbf{x})$ and intermediate bounds, at each training stage. Further details are provided in appendix C.1.

## 4.3 Training via IBP Regularization

Certified training algorithms that directly employ upper bounds to the robust loss (4.3) do not typically benefit from the use of more accurate verification algorithms than those they were trained with Zhang et al. (2020). On the other hand, the hybrid training methods from §4.2.3 are designed to be verified with tighter bounds, potentially encoding (part of) the network as a MILP (Tjeng et al., 2019). In light of the recent scaling improvements of complete verifiers (Bak et al., 2021), we present a robust training method designed for recent branch-and-bound frameworks (De Palma et al., 2021c; Wang et al., 2021a), capable of preserving COLT's effectiveness while simplifying its training procedure.

**Training objective** Intuitively, verification is easier if the network is robust by a large margin, and if the employed relaxation accurately represents the network, Therefore, we propose a simple training scheme revolving around the following two features: (i) adversarial attacks run over significantly larger domains than those employed at test time, (ii) a term minimizing the gap between the relaxations used for verification and the original neural network domain. Given its flexibility, small cost, and widespread usage, we aim to exclusively rely on IBP for bounding

computations. Details for IBP can be found in appendix C.4. We name the resulting method IBP Regularization (IBP-R), which operates on the following objective:

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left\{ \kappa \left[ \begin{array}{l} \max_{\mathbf{x}\in\mathcal{C}_+(\mathbf{x})} \mathcal{L}(f(\boldsymbol{\theta},\mathbf{x}),\mathbf{y}) + \\ \frac{\rho}{2}\sum_{j=1}^{n} \left[-\hat{\mathbf{l}}_j(\boldsymbol{\theta})\right]_+^T \left[\hat{\mathbf{u}}_j(\boldsymbol{\theta})\right]_+ \end{array} \right] + (1-\kappa)\mathcal{L}(f(\boldsymbol{\theta},\mathbf{x}),\mathbf{y}) + \lambda \left\|\boldsymbol{\theta}\right\|_1 , \tag{4.5}$$

where $\frac{1}{2}[-\hat{\mathbf{l}}_j(\boldsymbol{\theta})]_+^T [\hat{\mathbf{u}}_j(\boldsymbol{\theta})]_+$ represents the area of the widely-employed ReLU convex hull represented in Figure 4.1, and $\mathcal{C}_+(\mathbf{x})$ is a superset of the original input specification from condition (4.1). For adversarial robustness specifications, $\mathcal{C}_+(\mathbf{x}) := \{\mathbf{x}_0 | \left\|\mathbf{x}_0 - \mathbf{x}\right\|_p \leq \alpha\epsilon_{\text{ver}}\}$, with $\alpha \geq 1.6$ in our experiments (see §4.6). Note that we regularize over all the activations of the network at once, with the same regularization coefficient. An in-depth comparison with COLT is available in appendix C.1.

**Regularization masking** When a property holds, the tightness of the employed relaxations is fundamental in order to swiftly provide a formal guarantee via branch-and-bounds methods. On the other hand, verifying that the property does not hold typically implies finding counter-examples via adversarial attacks (Bak et al., 2021). As a consequence, there is no need to encourage tightness by minimizing $\frac{1}{2}[-\hat{\mathbf{l}}_j(\boldsymbol{\theta})]_+^T [\hat{\mathbf{u}}_j(\boldsymbol{\theta})]_+$ when $P(f(\boldsymbol{\theta},\mathbf{x}_0),\mathbf{y})\ \forall\, \mathbf{x}_0 \in \mathcal{C}(\mathbf{x})$ is unlikely to hold after training. In order to take this observation into account, we propose to mask the convex hull regularizer when a counter-example is found for the current sample. Denoting by $\tilde{\mathbf{x}}$ the point found by the train-time adversarial attack, we perform the following substitution in objective (4.5):

$$\frac{1}{2}[-\hat{\mathbf{l}}_j(\boldsymbol{\theta})]_+^T [\hat{\mathbf{u}}_j(\boldsymbol{\theta})]_+ \rightarrow \frac{\mathbb{1}_{P(f(\boldsymbol{\theta},\tilde{\mathbf{x}}),\mathbf{y})}}{2}[-\hat{\mathbf{l}}_j(\boldsymbol{\theta})]_+^T [\hat{\mathbf{u}}_j(\boldsymbol{\theta})]_+ .$$

**Training details** As common for recent certified training algorithms (Balunovic and Vechev, 2020; Zhang et al., 2020), $\kappa$ is linearly increased from 0 to 1 at the beginning of training (mixing). Similarly to IBP Gowal et al. (2018b) and CROWN-IBP (Zhang et al., 2020), we also linearly increase the effective perturbation radius from 0 to $\alpha\epsilon_{\text{ver}}$ while mixing the objectives.

## 4.4 Verification Framework

As explained in §4.3, IBP-R is designed to facilitate the verification of trained networks via recent branch-and-bound frameworks. We will now first present the details of the employed complete verifier (§4.4.1), then present a novel branching strategy in §4.4.2.

### 4.4.1 Branch–and-Bound Setup

Owing to its modularity and performance on large COLT-trained networks (Bak et al., 2021, `cifar2020` benchmark), we base our verifier on the OVAL branch-and-bound framework (Bunel et al., 2020a,b; De Palma et al., 2021c) from VNN-COMP-2021 (Bak et al., 2021). Given that IBP-R explicitly seeks to minimize the area of the ReLU convex hull (see Figure 4.1), we instantiate the framework so as to use $\beta$-CROWN (Wang et al., 2021a) for the bounding (see §4.2.1), a state-of-the-art solver for the employed relaxation, designed for use within branch-and-bound. In line with De Palma et al. (2021c), intermediate bounds are never updated after branching, and they are individually computed via $\alpha$-CROWN Xu et al. (2021). Furthermore, the dual variables of each bounding computation are initialized to the values associated to the parent node (that is, the bounding performed before the last split), and the number of dual iterations is dynamically adjusted to reduce the bounding time (De Palma et al., 2021c). Counter-examples are found using the MI-FGSM Dong et al. (2018) adversarial attack, which is run repeatedly for each property, using a variety of hyper-parameter settings. Verification is run with a timeout of 1800 seconds, and terminated early when the property is likely to time out. We now present the employed branching strategy.

### 4.4.2 UPB Branching

In spite of its strong empirical performance on COLT-trained networks (Bak et al., 2021), the FSB branching strategy (De Palma et al., 2021c), commonly employed for $\beta$-CROWN (Wang et al., 2021a), requires $O(n)$ CROWN-like bounding computations per split. In order to reduce the branching overhead, we present Upper

Planet Bias (UPB), a novel and simpler branching strategy that yields splitting decisions of comparable quality at the cost of a single gradient backpropagation through the network.

The popular SR (Bunel et al., 2020b) and FSB branching strategies partly rely on estimates of the sensitivity of CAP bounds (Wong and Kolter, 2018) to the splitting of an ambiguous ReLUs (De Palma et al., 2021c). As the employed relaxation for the output bounding is typically much tighter, these estimates are often unreliable. In order to improve branching performance, FSB couples these estimates with an expensive bounding step. We propose to remove the need to compute bounds at branching time by re-using dual information from branch-and-bound's bounding step, which we perform using $\beta$-CROWN (see §4.4.1). Specifically, we propose to score branching decisions according to a dual term associated to the bias of the upper linear constraint from the Planet relaxation (Figure 4.1) for each ambiguous neuron:

$$\mathbf{s}_{\text{UPB},k} = \frac{\left[-\hat{\mathbf{l}}_j\right]_+^T [\hat{\mathbf{u}}_j]_+}{\left(\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k\right)} \odot [\bar{\boldsymbol{\lambda}}_k]_+ \quad k \in [\![1, n-1]\!]\,, \tag{4.6}$$

where $[\bar{\boldsymbol{\lambda}}_k]_+$ is computed by evaluating equation (C.8) in appendix C.5 on the dual variables computed for the branch-and-bound bounding step. Therefore, the cost of computing scores (4.6) for all neurons corresponds to that of a single gradient backpropagation. Intuitively, as $\mathbf{s}_{\text{UPB},k}$ will disappear from the dual objective after splitting, we employ it as a proxy for a branching decision's potential bounding improvement. See appendix C.5 for further details.

## 4.5 Related Work

Many popular certified training algorithms work by upper bounding the robust loss (4.3) via some combination of IBP and linear bounds on the activation function. IBP (Gowal et al., 2018b; Mirman et al., 2018), CAP (Wong and Kolter, 2018; Wong et al., 2018), and CROWN-IBP (Zhang et al., 2020) all fall in this category. Shi et al. (2021) recently proposed a series of techniques to shorten the usually long training schedules of these algorithms. Xu et al. (2020) provide minor improvements on CROWN-IBP by changing the way the loss function is incorporated into problem

(4.2). The above family of methods produce state-of-the-art results for larger perturbation radii. Regularization-based techniques, instead, tend to perform better on smaller radii (see §4.2.3). Further works have focused on achieving robustness via specialized network architectures (Lyu et al., 2021; Zhang et al., 2021), Lipschitz constant estimation for perturbations in the $\ell_2$ norm (Huang et al., 2021), or under randomized settings (Cohen et al., 2019; Salman et al., 2019a): these methods are out of the scope of the present work.

As outlined in §4.2, widely-employed neural network relaxations model the convex hull of the activation function, referred to as the convex barrier due to its popularity (Salman et al., 2019b), or on even looser convex outer-approximations such as CAP (Wong and Kolter, 2018), CROWN (Zhang et al., 2018), DeepZ (Singh and Shawe-Taylor, 2018), or DeepPoly (Singh et al., 2019b). These relaxations are relatively inexpensive yet very effective when adapted for complete verification via branch-and-bound, and are hence at the core of the $\alpha$-$\beta$-CROWN (Xu et al., 2021; Wang et al., 2021a) and OVAL frameworks (Bunel et al., 2020a; De Palma et al., 2021c). A number of works have recently focused on devising tighter neural network relaxations (Singh et al., 2019a; Anderson et al., 2020; Tjandraatmadja et al., 2020; Müller et al., 2022). These have been integrated into recent branch-and-bound verifiers (De Palma et al., 2021a,b; Ferrari et al., 2022) and yield strong results for harder verification properties on medium-sized networks.

While the employed relaxations are a fundamental component of a complete verifier, the overall speed-accuracy trade-offs are greatly affected by the employed branching strategy. Small networks with few input dimensions can be quickly verified by recursively splitting the input region (Bunel et al., 2018; Royo et al., 2019). On the other hand, activation splitting (Ehlers, 2017; Katz et al., 2017) (see §4.2.1, §4.4.2) performs better for larger convolutional networks (Bunel et al., 2020b; De Palma et al., 2021c) and enjoyed recent developments based on graph neural networks (Lu and Kumar, 2020) or for use with tighter relaxations (Ferrari et al., 2022).

| Perturbation | Method | Standard accuracy [%] | Robust accuracy [%] | Verified accuracy [%] | Runtime [s] |
|---|---|---|---|---|---|
| $\epsilon_{\text{ver}} = 2/255$ | COLT | **78.37 ± 0.24** | 65.66 ± 0.13 | 61.88 ± 0.11 | $3.05 \times 10^4 \pm 1.21 \times 10^2$ ‡ |
| | IBP-R | 78.19 ± 0.52 | **66.39 ± 0.12** | **61.97 ± 0.18** | $9.34 \times 10^3 \pm 2.95 \times 10^1$ |
| | IBP-R w/ Masking | 78.22 ± 0.26 | 66.28 ± 0.17 | 61.69 ± 0.29 | $9.63 \times 10^3 \pm 4.42 \times 10^1$ |
| | Literature Results | | | | |
| | (Shi et al., 2021)* | 66.84 | / | 52.85 | |
| | (Zhang et al., 2020)* | 71.52 | 59.72 | 53.97 | $9.13 \times 10^4$ † |
| | (Balunovic and Vechev, 2020) | 78.4 | / | 60.50 | |
| | (Xiao et al., 2019) | 61.12 | 49.92 | 45.93 | |
| $\epsilon_{\text{ver}} = 8/255$ | COLT | 51.94 ± 0.14 | 31.68 ± 0.23 | 28.73 ± 0.23 | $1.03 \times 10^4 \pm 1.70 \times 10^1$ ‡ |
| | IBP-R | 51.43 ± 0.21 | 31.89 ± 0.11 | 27.87 ± 0.01 | $5.92 \times 10^3 \pm 2.95 \times 10^1$ |
| | IBP-R w/ Masking | 52.74 ± 0.30 | 32.78 ± 0.33 | 27.55 ± 0.22 | $5.89 \times 10^3 \pm 3.38 \times 10^1$ |
| | Literature Results | | | | |
| | (Shi et al., 2021)* | 48.28 ± 0.40 | / | **34.42 ± 0.32** | $9.51 \times 10^3$ ◇ |
| | (Zhang et al., 2020)* | **54.50** | **34.26** | 30.50 | $9.13 \times 10^4$ † |
| | (Balunovic and Vechev, 2020) | 51.70 | / | 27.50 | |
| | (Xiao et al., 2019) | 40.45 | 26.78 | 20.27 | |

* the employed 7-layer network has $17.2 \times 10^6$ parameters, as opposed to the $2.1 \times 10^6$ parameters of the 5-layer and 4-layer networks respectively used for our $\epsilon_{\text{ver}} = 2/255$ and $\epsilon_{\text{ver}} = 8/255$ experiments. Differently from our work, data augmentation (random horizontal flips and croppings) is used.
◇ the training is performed on an Nvidia RTX 2080 Ti GPU, which is significantly faster than that employed in our experiments.
† the training is performed on 4 Nvidia RTX 2080 Ti GPUs.
‡ on the same setup, the original PyTorch implementation runs in $1.74 \times 10^5$ seconds for $\epsilon_{\text{ver}} = 2/255$, and $4.49 \times 10^4$ seconds for $\epsilon_{\text{ver}} = 8/255$.

**Table 4.1:** Performance of different verified training algorithms under $\ell_\infty$ norm perturbations on the CIFAR10 dataset. The table reports mean and sample standard deviation over 3 repetitions for our experiments, over 5 repetitions for (Shi et al., 2021) on $\epsilon_{\text{ver}} = 8/255$. The remaining results from the literature were executed with a single seed. The method with the best average performance for each perturbation radius is highlighted in bold.

## 4.6 Experiments

In this section, we present an experimental evaluation of the IBP-R certified training algorithm (§4.6.1), then evaluate the performance of our UPB branching strategy (§4.6.2). The implementation of our training algorithm is based on Jax (Bradbury et al., 2018), while verification is performed post-training by using a modified version (see §4.4) of the OVAL framework, implemented in PyTorch (Paszke et al., 2019).

### 4.6.1 Verified Training

We evaluate the efficacy of our IBP-R algorithm (§4.3) by replicating the CIFAR-10 (Krizhevsky and Hinton, 2009) experiments from Balunovic and Vechev (2020) and comparing against COLT, which we ported to Jax for fairness (resulting in significant speed-ups as shown in table 4.1, see appendix C.6.2 for details). We focus our comparison on COLT, as it is the best-performing instance of the hybrid training algorithms detailed in §4.2.3, family to which IBP-R belongs. Furthermore, to the best of our knowledge, it yields state-of-the-art results for small perturbations and ReLU networks. Details concerning the employed architecture, hyper-parameters,

and the computational setup can be found in appendix C.6. Timing results were executed on a Nvidia Titan V GPU.

Table 4.1 reports the results of our experiment, as well as relevant results from the literature. Specifically, we report results for: (i) CROWN-IBP (Zhang et al., 2020) and the improved IBP algorithm by Lyu et al. (2021), representing state-of-the-art certified training algorithms based on upper bounding the robust loss (see §4.2.2), (ii) Xiao et al. (2019), sharing many similarities with IBP-R (see §4.2.3), and (iii) the original COLT experiments from Balunovic and Vechev (2020). IBP-R excels on the smaller $\ell_\infty$ perturbation radius ($\epsilon_{\text{ver}} = 2/255$), displaying verified and standard accuracies comparable (given the experimental variability) to COLT, and a larger empirically robust accuracy (for details on the employed attack, see §4.4.1). Furthermore, IBP-R training is more than three times faster than COLT, making its use particularly convenient on this setup. The masking (see §4.3) does not appear to be beneficial on smaller perturbations. Relevant results from the literature all underperform compared to IBP-R, which hence achieves state-of-the-art results on this benchmark. On the larger perturbation radius ($\epsilon_{\text{ver}} = 8/255$), masking the regularization has a positive effect. The masked version of IBP-R performs comparably with COLT in $\approx 57\%$ of its runtime, attaining larger standard and empirically robust accuracies, and smaller verified accuracy. However, in this context, all regularization-based methods (including IBP-R) are outperformed by algorithms upper bounding the robust loss (Zhang et al., 2020; Lyu et al., 2021). Nevertheless, these works employ data augmentation and significantly larger networks than the one used in our experiments. We therefore conjecture that IBP-R, which scales better than COLT with the network depth (see runtimes in table 4.1), will become more competitive when evaluated on comparable settings.

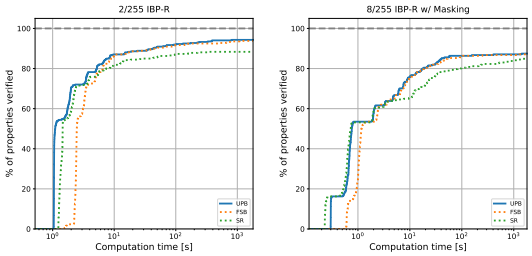
### 4.6.2 Branching

In order to test the efficacy of our UPB branching strategy (§4.4.2), we time verification for the first 500 CIFAR-10 test samples on two IBP-R-trained networks (one per perturbation radius) from the experiment of §4.6.1. Images that are

| | $\epsilon_{\text{ver}} = 2/255$ IBP-R | | | $\epsilon_{\text{ver}} = 8/255$ IBP-R w/ Masking | | |
|---|---|---|---|---|---|---|
| Method | time [s] | subproblems* | %Timeout | time [s] | subproblems* | %Timeout |
| UPB | **113.42** | **930.83** | **5.70** | **237.24** | 935.80 | **12.55** |
| FSB | 127.65 | 1244.01 | 6.22 | 243.29 | **795.22** | 12.92 |
| SR | 215.89 | 5185.27 | 11.66 | 303.76 | 6744.72 | 15.13 |

*computed on the properties that did not time out for neither UPB nor FSB. The inclusion of timed-out results in the average leads to an overestimation of the number of subproblems for the less expensive branching strategy.

**(a)** Comparison of average runtime, average number of solved subproblems and the percentage of timed out properties. The best performing method is highlighted in bold.




**(b)** Cactus plots: percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

**Figure 4.2:** Complete verification performance of different branching strategies on two IBP-R-trained CIFAR-10 networks from §4.6.1.

incorrectly classified are discarded. We keep the branch-and-bound settings fixed to those of §4.4.1, and benchmark against the following branching strategies: FSB (De Palma et al., 2021c), and SR (Bunel et al., 2020b). Appendix C.7 replicates the experiment on two COLT-trained networks.

Figure 4.2 shows that UPB improves average verification times compared to FSB (by roughly 13% and 2.5% for the $\epsilon_{\text{ver}} = 2/255$ and $\epsilon_{\text{ver}} = 8/255$ networks, respectively). On the larger perturbation radius, this holds in spite of a larger average number of visited subproblems, highlighting the cost of FSB's bounding-based selection step. On the other hand, UPB verifies more properties within the timeout on both networks. Furthermore, it reduces the number of visited subproblems on the smaller perturbation, testifying the efficacy of the selected domain splits. Both UPB and FSB yield significantly faster verification than SR on the considered problems. The results show that UPB is less expensive than the state-of-the-art FSB algorithm, while producing branching decisions of comparable quality.

## 4.7 Conclusions

Many state-of-the-art verified training algorithms require very large networks to obtain good robustness-accuracy trade-offs. Methods designed to exploit tight bounds at verification-time better exploit network capacity but they either underperform or involve extremely complex training procedures. We introduced IBP-R, a simple and intuitive robust training algorithm designed to induce verifiability via recent

branch-and-bound algorithms. We show that, by minimizing the area of the Planet relaxation via IBP bounds for all network activations, and using PGD attacks over larger perturbations, one can obtain state-of-the-art certified accuracy results on small perturbations without large compromises in standard accuracy. Finally, in order to ease the task of verifying the trained networks, we presented UPB, a straightforward and inexpensive branching strategy that yields branching decisions as effective as the state-of-the-art. We believe our results could be further improved by leveraging recent improvements on standard IBP training (Shi et al., 2021), as well as larger network architectures: these are interesting avenues for future work.

# 5

# In Defense of the Unitary Scalarization for Deep Multi-Task Learning

## Contents


5.1 Introduction . . . . . **117**
5.2 Related Work . . . . . **118**
5.3 Multi-Task Learning Optimizers . . . . . **120**
5.4 Experimental Evaluation . . . . . **124**
5.4.1 Supervised Learning . . . . . 125
5.4.2 Reinforcement Learning . . . . . 129
5.5 Regularization in Specialized MTL Optimizers . . . . . **131**
5.5.1 Ablation Study . . . . . 131
5.5.2 Technical Results . . . . . 132
5.5.3 Under-Optimization: Empirical Study . . . . . 136
5.6 Conclusions . . . . . **137**

# Abstract

Recent multi-task learning research argues against *unitary scalarization*, where training simply minimizes the sum of the task losses. Several ad-hoc multi-task optimization algorithms have instead been proposed, inspired by various hypotheses about what makes multi-task settings difficult. The majority of these optimizers require per-task gradients, and introduce significant memory, runtime, and implementation overhead. We show that unitary scalarization, coupled with standard regularization and stabilization techniques from single-task learning, matches or improves upon the performance of complex multi-task optimizers in popular supervised and reinforcement learning settings. We then present an analysis suggesting that many specialized multi-task optimizers can be partly interpreted as forms of regularization, potentially explaining our surprising results. We believe our results call for a critical reevaluation of recent research in the area.

## 5.1 Introduction

Multi-Task Learning (MTL) (Caruana, 1997a) exploits similarities between tasks to yield models that are more accurate, generalize better and require less training data. Owing to the success of MTL on traditional machine learning models (Heskes, 2000; Bakker and Heskes, 2003; Evgeniou and Pontil, 2004) and of deep single-task learning across a variety of domains, a growing body of research has focused on deep MTL. The most straightforward way to train a neural network for multiple tasks at once is to minimize the sum of per-task losses. Adopting terminology from multi-objective optimization, we call this approach *unitary scalarization*.

While some work shows that multi-task networks trained via unitary scalarization exhibit superior performance to independent per-task models (Kokkinos, 2017; Kalashnikov et al., 2021), others suggest the opposite (Teh et al., 2017b; Kendall et al., 2018; Sener and Koltun, 2018). As a result, many explanations for the difficulty of MTL have been proposed, each motivating a new SMTO (Sener and Koltun, 2018; Liu et al., 2021c; Yu et al., 2020; Chen et al., 2020; Wang et al., 2021b). These works typically claim that the proposed SMTO outperforms unitary scalarization, in addition to relevant prior work. However, SMTOs usually require access to per-task gradients either with respect to the shared parameters, or to the shared representation. Therefore, their reported performance gain comes at significant computation and memory cost, the overhead scaling linearly with the number of tasks. By contrast, unitary scalarization requires only the average of the gradients across tasks, which can be computed via a single backpropagation.

Existing SMTOs were introduced to solve challenges related to the optimization of the deep MTL problem. We instead postulate that the reported weakness of unitary scalarization is linked to experimental variability or to a lack of regularization, leading to the following contributions:

- A comprehensive experimental evaluation (§5.4) of recent SMTOs on popular multi-task benchmarks, showing that no SMTO consistently outperforms unitary scalarization in spite of the added complexity and overhead. In particular, either

the differences between unitary scalarization and SMTOs are not statistically significant, or they can be bridged by standard regularization and stabilization techniques from the single-task literature. Our RL experiments include optimizers previously applied only to supervised learning.

- An empirical and technical analysis of the considered SMTOs, suggesting that they reduce overfitting on the multi-task problem and hence act as regularizers (§5.5). We conduct an ablation study and provide a collection of novel and existing technical results that support this hypothesis.

- Code to reproduce the experiments, including a unified PyTorch (Paszke et al., 2019) implementation of the considered SMTOs, is available at `https://github.com/yobibyte/unitary-scalarization-dmtl`.

We believe that our results suggest that the considered SMTOs can be often replaced by less expensive techniques. We hope that these surprising results stimulate the search for a deeper understanding of MTL.

## 5.2 Related Work

Before diving into details of specific SMTOs in Section 5.5, we provide a high-level overview of the deep MTL research. Seminal work in MTL includes *hard parameter sharing* (Caruana, 1997b): sharing neural network parameters between all tasks with, possibly, a separate part of the model for each task. Hard parameter sharing is still the major MTL approach adopted in natural language processing (Collobert and Weston, 2008; Chen et al., 2021), computer vision (Misra et al., 2016), and speech recognition (Seltzer and Droppo, 2013). In this work, we implicitly assume that each parameter update employs information from all tasks. However, not all works satisfy this assumption, either due to a large number of tasks (Cappart et al., 2021; Kurin et al., 2020), or simply as an implementation decision (Huang et al., 2020; Kurin et al., 2021). In this setting, MTL resembles other problems dealing with multiple tasks, i.e., continual (Khetarpal et al., 2020), curriculum (Narvekar et al., 2020), and meta-learning (Hospedales et al., 2020), which are not the focus of this work.

Many works strive to improve the performance of deep multi-task models. One line of research hypothesizes that conflicting per-task gradient directions lead to suboptimal models, and focuses on explicitly removing such conflicts Yu et al. (2020); Chen et al. (2020); Liu et al. (2021c); Wang et al. (2021b); Javaloy and Valera (2022); Liu et al. (2021a). Some authors postulate that loss imbalances across tasks hinder learning, proposing loss reweighting methods (Kendall et al., 2018; Chen and Gu, 2018; Lin et al., 2022). Sener and Koltun (2018) and Navon et al. (2022) propose that tasks compete for model capacity and interpret MTL as multi-objective optimization in order to cope with inter-task competition. Here, we focus on algorithms that explicitly rely on per-task gradients to try to outperform unitary scalarization (§5.5). Research on multi-task architectures (Misra et al., 2016; Guo et al., 2020) or MTL algorithms exclusively motivated by deterministic loss reweighting (Kendall et al., 2018; Guo et al., 2018; Liu et al., 2019) are orthogonal to our work. Both topics are investigated by a recent survey on pixel-level multi-task computer vision problems (Vandenhende et al., 2021), which found that the minimization of tuned weighted sums of losses (scalarizations) is empirically competitive with deterministic loss reweighting and MGDA in the considered settings. These results are extended to popular SMTOs by a critical review from Xin et al. (2022), concurrent to our work, which argues that the optimization and generalization performance of SMTOs can be matched by tuning scalarization coefficients. Our work reaches a similar conclusion, demonstrating that unitary scalarization performs on par with SMTOs when coupled with standard and inexpensive regularization or stabilization techniques. In other words, Xin et al. (2022) provide complementary support for the link between SMTOs and regularization by showing that tuning scalarization weights positively affects generalization.

In addition to the common supervised settings, we also consider multi-task RL, whose research can be grouped into three categories: the first adds auxiliary tasks providing additional inductive biases to speed up learning (Jaderberg et al., 2017) on a target task. The second, based on policy distillation, uses per-task teacher models to provide labels for a multi-task model or per-task policies as

regularizers (Rusu et al., 2016; Parisotto et al., 2016; Teh et al., 2017a). The third directly learns a shared policy (Kalashnikov et al., 2021), possibly via an SMTO (Yu et al., 2020). We focus on the third category, whose literature reports varying performance for unitary scalarization (better (Kalashnikov et al., 2021) or worse (Yu et al., 2020) than per-task models), indicating confounding factors in evaluation pipelines and further motivating our work. PopArt (van Hasselt et al., 2016; Hessel et al., 2019) performs scale-invariant value function updates in order to address differences in returns across environments, showing improvements in the multi-task setting while still using unitary scalarization. PopArt does not require per-task gradients but introduces additional hyperparameters. In our work, we address the differences in rewards by normalizing them at the replay buffer level. However, we believe both unitary scalarization and SMTOs might equally benefit from PopArt.

## 5.3 Multi-Task Learning Optimizers

We will now describe the deep MTL training problem and popular algorithms employed for its solution. Let $(X, Y) \in \mathbb{R}^{d\times n} \times \mathbb{R}^{o\times n}$ be the training set, composed of $n$ $d$-dimensional points and $o$-dimensional labels. In addition, $\mathcal{L}_i : \mathbb{R}^{o\times n} \times \mathbb{R}^{o\times n} \to \mathbb{R}$ denotes the loss for the $i$-th task, $\boldsymbol{\theta} \in \mathbb{R}^S$ the parameter space, $\mathcal{T} := \{1, \dots, m\}$ the set of $m$ tasks. The goal of MTL is to learn a single (generally task-aware) parametrized model $f : \mathbb{R}^S \times \mathbb{R}^{d\times n} \times \mathcal{T} \to \mathbb{R}^{o\times n}$ that performs well on all tasks $\mathcal{T}$. The parameter space is often split into a set of shared parameters across tasks (generally the majority of the architecture), denoted $\boldsymbol{\theta}_{\|}$, and (possibly empty) task-specific parameters, denoted $\boldsymbol{\theta}_{\perp}$, so that $\boldsymbol{\theta} := [\boldsymbol{\theta}_{\|}, \boldsymbol{\theta}_{\perp}]^T$. In this context, the model $f$ often takes on an encoder-decoder architecture, where the encoder $g$ learns a shared representation across tasks, and the decoders $h_i$ are task-specific predictive heads: $f(\boldsymbol{\theta}, X, i) = h_i(g(\boldsymbol{\theta}_{\|}, X), \boldsymbol{\theta}_{\perp})$. In this case, we denote by $\mathbf{z} = g(\boldsymbol{\theta}_{\|}, X) \in \mathbb{R}^{r\times n}$ the $r$-dimensional shared representation of $X$.

The training problem for MTL is typically formulated as the sum of the per-task losses (Sener and Koltun, 2018; Yu et al., 2020; Chen et al., 2020):

$$\min_{\boldsymbol{\theta}} \left[ \; \mathcal{L}^{\mathrm{MT}}(\boldsymbol{\theta}) := \textstyle\sum_{i\in\mathcal{T}} \mathcal{L}_i(f(\boldsymbol{\theta}, X, i), Y) \; \right]. \tag{5.1}$$

**Unitary Scalarization** The obvious way to minimize the multi-task training objective in equation (5.1) is to rely on a standard gradient-based algorithm. While, for simplicity, we focus on standard gradient descent rather than mini-batch stochastic gradient descent, the notation can be adapted by replacing the dataset size $n$ by the mini-batch size $b$. Equation (5.1) corresponds to a linear scalarization with unitary weights under a multi-objective interpretation of MTL; hence, we call the direct application of gradient descent on equation (5.1) *unitary scalarization*. For vanilla gradient descent, this corresponds to taking a step in the opposite direction as the one given by the sum of per-task gradients: $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathrm{MT}} = \sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i$. Per-task gradients are not required, as it suffices to directly compute the gradient of the sum $\mathcal{L}^{\mathrm{MT}}$. Hence, when relying on deep learning frameworks based on reverse-mode differentiation, such as PyTorch (Paszke et al., 2019), the backward pass is performed once per iteration (rather than $m$ times). Furthermore, the memory cost is a factor $m$ less than most SMTOs, which require access to each $\nabla_{\boldsymbol{\theta}} \mathcal{L}_i$. As a consequence, unitary scalarization is simple, fast, and memory efficient. Our experiments demonstrate that, when possibly coupled with single-task regularization such as early stopping, $\ell_2$ penalty or dropout layers (Srivastava et al., 2014), this simple optimizer is strongly competitive with SMTOs.

**MGDA** Sener and Koltun (2018) point out that equation (5.1) can be cast as a multi-objective optimization problem with the following objective: $\boldsymbol{\mathcal{L}}^{\mathrm{MT}}(\boldsymbol{\theta}) := [\mathcal{L}_1(\boldsymbol{\theta}), \dots, \mathcal{L}_m(\boldsymbol{\theta})]^T$. A commonly employed solution concept in multi-objective optimization is Pareto optimality. A point $\boldsymbol{\theta}^*$ is called Pareto-optimal if, for any another point $\boldsymbol{\theta}^\dagger$ such that $\exists i \in \mathcal{T} : \mathcal{L}_i(\boldsymbol{\theta}^\dagger) < \mathcal{L}_i(\boldsymbol{\theta}^*)$, then $\exists j \in \mathcal{T} : \mathcal{L}_j(\boldsymbol{\theta}^\dagger) > \mathcal{L}_j(\boldsymbol{\theta}^*)$. A necessary condition for Pareto optimality at a point is Pareto stationarity, defined as the lack of a shared descent direction across all losses at that point. Sener and Koltun (2018) rely on MGDA (Désidéri, 2012) to reach a Pareto-stationary point for shared parameters $\boldsymbol{\theta}_{\|}$. Intuitively, MGDA proceeds by repeatedly stepping in a shared descent direction (Fliege and Svaiter, 2000; Désidéri, 2012), which can

be found by solving the following optimization problem:

$$\min_{\mathbf{g},\epsilon} \left[\epsilon + 1/2 \left\|\mathbf{g}\right\|_2^2\right] \quad \text{s.t.} \quad \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i^T \mathbf{g} \leq \epsilon \quad \forall\, i \in \mathcal{T}, \tag{5.2}$$

whose dual takes the following form (corresponding to the formulation from Désidéri (2012)):

$$\max_{\boldsymbol{\alpha} \geq 0} -1/2 \left\|\mathbf{g}\right\|_2^2 \quad \text{s.t.} \quad \sum_i \alpha_i \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i = -\mathbf{g}, \quad \sum_{i \in \mathcal{T}} \alpha_i = 1. \tag{5.3}$$

In other words, MGDA takes a step in a direction $\mathbf{g}$ given by the negative convex combination of per-task gradients, whose coefficients are given by solving equation (5.3). In practice, per-task gradients are rescaled before applying MGDA: the original authors' implementation (Sener and Koltun, 2018) relies on $\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i \leftarrow {}^{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i} / {\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|}_{\mathcal{L}_i(\boldsymbol{\theta})}$. The convergence of MGDA to a Pareto-stationary point is still guaranteed after normalization (Désidéri, 2012).

**IMTL** IMTL (Liu et al., 2021c) is presented as an SMTO that is not biased against any single task. It is composed of two complementary algorithmic blocks: IMTL-L, acting on task losses, and IMTL-G, acting on per-task gradients. IMTL-G follows the intuition that a multi-task optimizer should proceed along a direction $\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i$ that equally represents per-task gradients. This is formulated analytically by requiring that the cosine similarity between $\mathbf{g}$ and each $\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i$ be the same. To prevent the resulting problem from being underdetermined, Liu et al. (2021c) add the constraint $\sum_{i \in \mathcal{T}} \alpha_i = 1$, resulting in a problem that admits a closed-form solution for $\mathbf{g}$:

$$\begin{aligned} &\mathbf{g}^T \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1\right\|} = \mathbf{g}^T \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|} \;\; \forall\, i \in \mathcal{T} \setminus \{1\}, \\ &\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i, \quad \sum_{i \in \mathcal{T}} \alpha_i = 1. \end{aligned} \tag{5.4}$$

IMTL-L, instead, aims to reweight task losses so that they are all constant over time, and equal to 1. In order to limit oscillations of the scaling factors, the authors propose to learn them jointly with the network by minimizing a common objective via gradient descent. In particular, given $s_i \in \mathbb{R}\ \forall i \in \mathcal{T}$,

Liu et al. (2021c) derive the following form for the joint minimization problem: $\min_{\mathbf{s},\boldsymbol{\theta}} \left[\sum_i \left(e^{s_i}\mathcal{L}_i(\boldsymbol{\theta}) - s_i\right)\right]$. As proved by Liu et al. (2021c), IMTL-L only has a rescaling effect on the update direction of IMTL-G. Unlike IMTL-G and the other SMTOs presented in this section, IMTL-L rescaling is designed to affect the updates for task-specific parameters $\boldsymbol{\theta}_\perp$ as well.

**PCGrad** Let us write $\cos(\mathbf{x}, \mathbf{z})$ for the cosine similarity between vectors $\mathbf{x}$ and $\mathbf{z}$. Yu et al. (2020) postulate that multi-task convergence is severely slowed down if the following three conditions (named the *tragic triad*) hold at once: (i) conflicting gradient directions: $\cos(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i, \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j) < 0$ for some $i, j \in \mathcal{T}$; (ii) differing gradient magnitudes: $\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i\right\| \gg \left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j\right\|$ for some $i, j \in \mathcal{T}$; and (iii) the unitary scalarization $\mathcal{L}^{\text{MT}}$ has high curvature along $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{MT}}$. The PCGrad (Yu et al., 2020) SMTO is presented as a solution to the tragic triad, targeted at the first condition. Consistent with the previous sections, let us denote the update direction by $\mathbf{g}$. Furthermore, let $[\mathbf{x}]_+ := \max(\mathbf{x}, \mathbf{0})$. Given per-task gradients $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i$, PCGrad iteratively projects each task gradient onto the normal plane of all the gradients with which it conflicts:

$$\begin{aligned}&\left[\mathbf{g}_i \leftarrow \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i,\ \ \mathbf{g}_i \leftarrow \mathbf{g}_i + \left[\frac{-\mathbf{g}_i^T \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j\right\|^2}\right]_+ \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j \ \ \forall j \in \mathcal{T} \setminus \{i\}\right] \forall i \in \mathcal{T}, \\ &\mathbf{g} = -\sum_{i \in \mathcal{T}} \mathbf{g}_i,\end{aligned} \tag{5.5}$$

where the iterative updates of $\mathbf{g}_i$ with respect to task gradient $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j$ are performed in random order.

**GradDrop** Chen et al. (2020) focus on conflicting signs across task gradient entries, arguing that such conflicts lead to gradient "tug-of-wars". The GradDrop SMTO (Chen et al., 2020), presented as a solution to this problem, proposes to randomly mask per-task gradients $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i$ so as to minimize such conflicts. Specifically, GradDrop computes the "positive sign purity" $p_j$ for the task gradient's $j$-th entry and then masks the $j$-th entry of each per-task gradient with probability increasing with $p_j$, if the entry is negative, or decreasing with $p_j$, if the entry

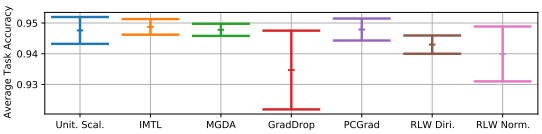
is positive. Let us write $\mathbf{p} := [p_1, \dots, p_S]$, where $S$ is the dimensionality of the parameter space (see §5.3), $\odot$ for the Hadamard product and $\mathbb{1}_{\mathbf{a}}$ for the indicator vector on condition $\mathbf{a}$. Given a vector $\mathbf{u}_i$, uniformly sampled in $[\mathbf{0}, \mathbf{1}]$ at each iteration, GradDrop takes a step in the direction given by:

$$\mathbf{g} = \sum_{i \in \mathcal{T}} \begin{pmatrix} -\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \odot \mathbb{1}_{\left(\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i > 0\right)} \odot \mathbb{1}_{(\mathbf{u}_i > \mathbf{p})} \\ -\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \odot \mathbb{1}_{\left(\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i < 0\right)} \odot \mathbb{1}_{(\mathbf{u}_i < \mathbf{p})} \end{pmatrix}, \quad \text{with} \quad \mathbf{p} = \frac{1}{2}\left(1 + \frac{\sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}{\sum_{i \in \mathcal{T}} \left|\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i\right|}\right). \tag{5.6}$$

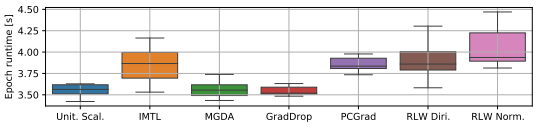
## 5.4 Experimental Evaluation

Relying on a unified experimental pipeline, we present an empirical evaluation on common MTL benchmarks of unitary scalarization (§5.3), of the popular SMTOs presented in §5.3, and of the recent RLW algorithms (Lin et al., 2022) due to their similarities with PCGrad and GradDrop (see §5.5.2). We benchmark against the two RLW instances that showed the best average performance in the original paper: RLW with weights sampled from a Dirichlet distribution ("RLW Diri."), and RLW with weights sampled from a Normal distribution ("RLW Norm."). The goal of this section is to assess the efficacy of a popular line of previous work, focusing on a few representative or well-established optimizers. Therefore, we forego comparison with more recent SMTOs (Navon et al., 2022; Javaloy and Valera, 2022; Liu et al., 2021a). Nevertheless, we point out that these algorithms often lack significant enough improvements over the optimizers we consider, or may have substantial commonalities with them (see §5.5.2 for Nash-MTL (Navon et al., 2022), which was published concurrently to the finalization of this work). Whenever appropriate, we employ "Unit. Scal." as shorthand for unitary scalarization. We first present supervised learning experiments (§5.4.1), and then evaluate on a popular reinforcement learning benchmark (§5.4.2).

Our experiments indicate that the performance of unitary scalarization has been consistently underestimated in the literature. By showing the variability between




**(a)** Avg. task test accuracy: mean and 95% CI (10 runs).




**(b)** Box plots for the training time of an epoch (10 runs).

**Figure 5.1:** No algorithm outperforms unitary scalarization on the Multi-MNIST dataset.

runs and by relying on standard regularization and stabilization techniques from the single-task literature, we demonstrate that *no SMTO consistently outperforms unitary scalarization across the considered settings*. This result holds in spite of the added complexity and computational overhead associated with most SMTOs. We provide a potential explanation of our results in §5.5.

### 5.4.1 Supervised Learning

All the architectures employed in the supervised learning experiments conform to the encoder-decoder structure detailed in §5.3. Whenever suggested by the original authors for this context, the SMTO implementations rely on per-task gradients with respect to the last shared activation, $\nabla_{\mathbf{z}}$, rather than on the usually more expensive $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i$. In particular, this is the case for MGDA, IMTL and GradDrop. See appendix D.2 for details concerning each individual algorithm. Surprisingly, several MTL works (Yu et al., 2020; Chen et al., 2020; Liu et al., 2021c; Lin et al., 2022) report validation results, making it easier to overfit. Instead, following standard machine learning practice, we select a model on the validation set, and later report test metrics for all benchmarks. Validation results are also available in appendix D.4. Appendix D.3.1 reports the computational setup, hyperparameter and tuning details.

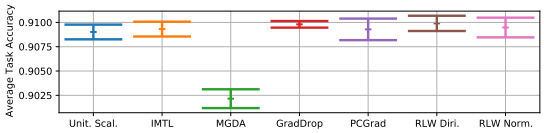
#### 5.4.1.1 Multi-MNIST

We present results on the Multi-MNIST dataset, a simple two-task supervised learning benchmark. Multi-MNIST, originally introduced by Sabour et al. (2017) and as modified by Sener and Koltun (2018), is constructed by uniformly sampling MNIST (LeCun et al., 1998) images, and placing one in the top-left corner, the other in the bottom-right corner. Each of the two overlaid images corresponds to a

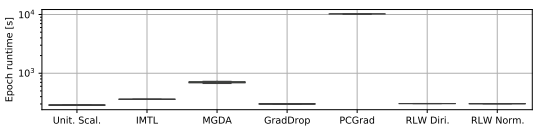
10-class (one per digit) classification task. Using the above procedure, we generate the Multi-MNIST training set from the first 50000 MNIST training images, the validation set from the last 10000 training images, and the test set from the original MNIST test set. We employ a popular architecture from previous work (Sener and Koltun, 2018; Yu et al., 2020) (see appendix D.3.1), where a single dropout layer (Srivastava et al., 2014) (with dropout probability 0.5) is employed in both the encoder and the decoder. $\ell_2$ regularization did not improve validation performance and was therefore omitted. Figure 5.1 reports the average task test accuracy, and the training time per epoch. For each run, the test model was selected as the model with the largest average task validation accuracy across the training epochs. Appendix D.4 presents the results of Figure 5.1 in tabular form, as well as the average task validation accuracy per epoch. As seen from the overlapping confidence intervals, none of the considered algorithms clearly outperforms the others. However, GradDrop displays higher experimental variability. Finally, Figure 5.1(b) shows that unitary scalarization also has among the lowest training times.

#### 5.4.1.2 CelebA

We now show results for the CelebA (Liu et al., 2015) dataset, a challenging 40-task multi-label classification problem. The dataset consists of 200,000 celebrity headshots (with standard training, validation and test splits) labelled for the presence or absence of 40 separate attributes (e.g., beard, eyeglasses), each corresponding to a binary classification task in the MTL literature. We employ the same architecture as many previous studies (Sener and Koltun, 2018; Yu et al., 2020; Lin et al., 2022; Liu et al., 2021c) (see appendix D.3.1). We tuned $\ell_2$ regularization terms $\lambda$ for all




**(a)** Avg. task test accuracy: mean and 95% CI (3 runs).



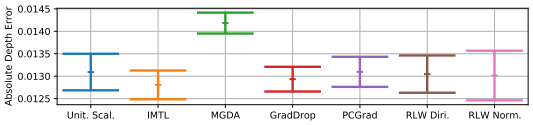

**(b)** Box plots for the training time of an epoch (10 runs).

**Figure 5.2:** While SMTOs display larger runtimes, none of them outperforms the unitary scalarization on the CelebA dataset.

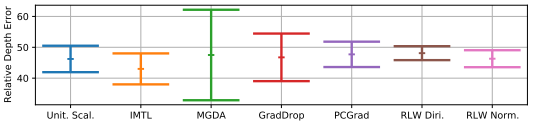
SMTOs in the following grid: $\lambda \in \{0, 10^{-4}, 10^{-3}\}$. The best validation performance was attained with $\lambda = 10^{-3}$ for unitary scalarization, IMTL and PCGrad, and with $\lambda = 10^{-4}$ for MGDA, GradDrop, and RLW. Validation performance was further stabilized by the addition of several dropout layers (see Figure 5.5), with dropout probabilities from 0.25 to 0.5. We present an ablation study on the effect of regularization on this experiment in §5.5.1. Figure D.4 (appendix D.4.2) shows that regularization improves the peak average validation performance for all the considered methods. Analogously to our Multi-MNIST results, Figure 5.2 plots the distribution of the training time per epoch, and the average test task accuracy. As with Multi-MNIST, the test model for each run was the one with maximal average validation task accuracy across epochs. In other words, if the peak is attained before the last epoch, we perform early stopping: as shown in Figure D.2(a) in appendix D.4 this is the case for most methods. Due to the large number of tasks, Figure 5.2(b) shows relatively large runtime differences across methods. PCGrad is the slowest (roughly 35 times slower than unitary scalarization). In fact, amongst the considered algorithms, it is the only one that computes per-task gradients over the parameters ($\nabla_{\boldsymbol{\theta}} \mathcal{L}_i \ \forall i \in \mathcal{T}$) at each iteration. GradDrop, MGDA and IMTL have overhead factors (compared to unitary scalarization) ranging from roughly 1.05 to 2.4 due to the relatively small size of $\mathbf{z}$ for the employed architecture. The overhead of RLW is negligible: roughly 5%. Nevertheless, due to largely overlapping confidence intervals in Figure 5.2(a), none of the methods consistently outperforms unitary scalarization. In fact, owing to our adoption of explicit regularization techniques (see §5.5.1) its average performance is superior to that reported in the literature (Sener and Koltun, 2018; Liu et al., 2021c).

#### 5.4.1.3 Cityscapes

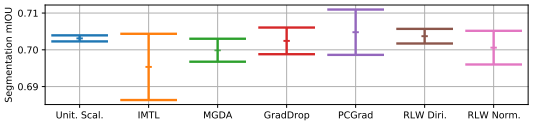
In order to complement the multi-task classification experiments for Multi-MNIST and CelebA, we present results for Cityscapes (Cordts et al., 2016), a dataset for semantic understanding of urban street scenes. We rely on the version of the dataset pre-processed by Liu et al. (2019), which consists of $2,975$ training and 500 test



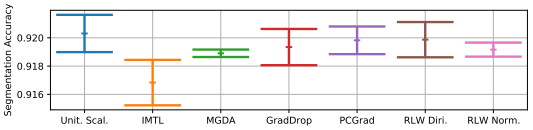

**(a)** Absolute depth test error: lower is better.



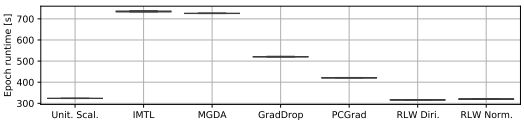

**(b)** Relative depth test error: lower is better.




**(c)** Test segmentation mIOU: higher is better.




**(d)** Test segmentation accuracy: higher is better.




**(e)** Box plots for the training time of an epoch (10 runs).

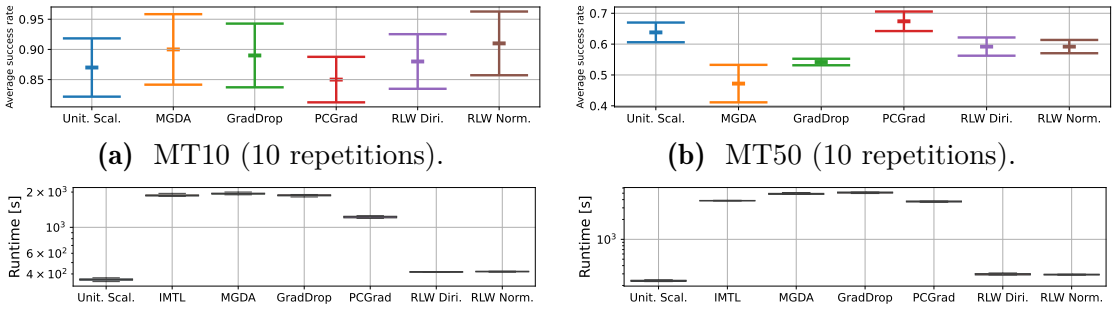
**Figure 5.3:** On Cityscapes, none of the SMTOs outperforms unitary scalarization, which proves to be the most cost-effective algorithm. Subfigures (a)-(d) report means for three runs, and their 95% CIs.

images and presents two tasks: semantic segmentation on 7 classes (e.g., human, vehicle), and depth estimation. We further split the original training set into a validation set of 595 images, employed to tune hyper-parameters, and a training set of 2380 images. We rely on a common encoder architecture from the literature (Liu et al., 2021c; Lin et al., 2022) (see appendix D.3.1), with a single dropout layer in the task-specific heads (Lin et al., 2022). As for CelebA, unitary scalarization, IMTL, and PCGrad benefit from more regularization than the other optimizers: we employ $\lambda = 10^{-5}$ for these three algorithms, as it resulted in better validation performance on the majority of metrics, and $\lambda = 0$ for the remaining methods. Cityscapes is a heterogeneous MTL problem: it contains tasks of different types whose validation metrics cannot be averaged to perform model selection. Considering the lack of an established procedure in this context, we potentially evaluate a different model for each metric, chosen as the one with the best (maximal or minimal, depending on the metric) validation performance across epochs (we perform per-run early stopping). This procedure maximizes per-task performance, at the cost of increased inference time. If inference time is a priority, an alternative model selection procedure could rely on relative task improvement (Javaloy and Valera, 2022; Navon et al., 2022; Liu et al., 2021a), assuming that per-metric improvements are to be weighted linearly.

Nevertheless, any consistently applied model selection scheme serves the main goal of our work: evaluating all SMTOs on a fair ground. Figure 5.3 shows test results for two metrics per task, and the distribution of the training time per epoch. As with Multi-MNIST and CelebA, no training algorithm clearly outperforms unitary scalarization (significant overlaps across confidence intervals exist), which is again the least expensive method. In contrast with a popular hypothesis (Kendall et al., 2018; Chen and Gu, 2018; Liu et al., 2021c), this holds in spite of relatively large loss imbalances. In fact, the loss for the depth task is roughly 10 times smaller than that of the segmentation task: see figures D.10(f)-D.10(g). Unlike CelebA (see Figure 5.2(b)), IMTL, MGDA and GradDrop are significantly slower than unitary scalarization (factors from 1.6 to 2.3), due to the relatively (compared to the parameter space) large size of $\mathbf{z}$ in the employed architecture. PCGrad, instead, appears to be less expensive (30% more than the baseline), demonstrating the benefits of working on $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i$ on this model.

### 5.4.2 Reinforcement Learning

For RL experiments, we use Meta-World (Yu et al., 2019), which consists of ten or fifty tasks (respectively MT10 and MT50) in which a simulated robot manipulator has to perform various actions, e.g., pressing a button, opening a door, or pushing a block. We rely on the Soft Actor-Critic (Haarnoja et al., 2018) implementation from (Sodhani et al., 2021). Unlike §5.4.1, the employed network architecture (see appendix D.3.1) is fully shared across tasks. Therefore, all SMTO implementations for these experiments rely on per-task gradients with respect to network parameters $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i$ (see §5.5). Among the SMTOs we consider, PCGrad is the only one developed with the RL setting in mind. For fairness and completeness, we add all the other SMTOs from the supervised learning experiments, and are the first to test these optimizers in the RL setting. To stabilize learning, we increase the replay buffer size, a well known technique in single-task RL, add actor $l_2$ regularization, and modify the reward normalization employed by Sodhani et al. (2021). The unitary scalarization performance reported by Yu et al. (2020)




**(a)** MT10 (10 repetitions). **(b)** MT50 (10 repetitions).

**(c)** MT10 (10 repetitions). **(d)** MT50 (10 repetitions).

**Figure 5.4:** On Metaworld, none of the SMTOs significantly outperforms Unit. Scal., which is the least expensive method. Subfigures (a)-(b) report mean and 95% CI for the best (over the updates) average success rate. Subfigures (c)-(d) show box plots for the training time of 10,000 updates.

is considerably lower than that of Sodhani et al. (2021), which we believe is due to the lack of reward normalization in the former. Sodhani et al. (2021) keep a moving average of rewards in the environment, with a hyperparameter controlling the speed of the moving average. As we show in Figure D.11, the learning algorithm is sensitive to that hyperparameter. Moreover, such normalization might make similar transitions have drastically different rewards stored in the replay buffer. To alleviate these issues, we store the raw rewards in the buffer, and normalize only when a mini-batch is sampled.

Figure 5.4 reports the best average success rate across the updates and the runtime for 10,000 updates. In addition to these summary statistics, reported for consistency with §5.4.1, the learning curves are shown in appendix D.5. Our MT10 (10 tasks) results in Figure 5.4(a) show that by stabilizing the baseline using standard RL techniques, unitary scalarization performs on par with other SMTOs, mirroring our findings in §5.4.1. This is in contrast with the previous literature, which reported that PCGrad outperforms unitary scalarization (Yu et al., 2020; Sodhani et al., 2021). Figure 5.4(b) presents results on MT50 (50 tasks): similarly to MT10, none of the SMTOs significantly outperforms unitary scalarization, with PCGrad's average being slightly above unitary scalarization. We speculate that the stochastic loss rescaling performed by PCGrad (see Proposition 7) reduces the differences in task return scales, and expect that methods like PopArt (van Hasselt et al., 2016) would have a similar effect without requiring access to per-task

gradients. While we did not tune hyperparameters for MT50 (we employed those found for MT10), it would be much easier to do that for unitary scalarization due to its lower runtime. In fact, Figure 5.4(d) shows that a single unitary scalarization run takes roughly 15 hours, whereas PCGrad, MGDA and GradDrop require more than a week. Similarly to MT10, actor regularization pushes the average performance of unitary scalarization higher (see in appendix D.5.2). Overall, as in the supervised learning setting, unitary scalarization performs comparably to SMTOs despite being simpler and less demanding in both memory and compute. IMTL was unstable on this RL benchmark and all of the runs crashed due to numerical overflow. We hence omit IMTL results from this chapter and show its results in Figure D.7 in appendix D.5, which also describes a possible explanation. We hypothesize that the instability of IMTL is due to lack of bounds on scaling coefficients. See appendix D.3.2 for hyperparameter settings and ablation studies.

## 5.5 Regularization in Specialized MTL Optimizers

The empirical results presented in §5.4 motivate the need to carefully analyze existing SMTOs. We make an initial attempt in this direction by viewing their effects through the lens of regularization. Let us define a regularizer as a technique to reduce overfitting (Dietterich, 1995). We first show that the SMTOs considered in §5.4 empirically act as regularizers via an ablation study (§5.5.1). We then take a closer look at their behavior, presenting technical results that support their alternative interpretation as regularizers (§5.5.2). Finally, §5.5.3 provides additional empirical backing for some of the technical results. Unless otherwise stated, we assume that MTL methods apply only to $\boldsymbol{\theta}_{\parallel}$ and that standard gradient-based updates are employed for tasks-specific parameters $\boldsymbol{\theta}_{\perp}$. We furthermore adopt the following shorthands: $\mathcal{L}_i(\boldsymbol{\theta})$ for $\mathcal{L}_i(f(\boldsymbol{\theta}, X, i), Y)$, and $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i$ for $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i(f(\boldsymbol{\theta}, X, i), Y)$.

### 5.5.1 Ablation Study

We repeat the experiment from §5.4.1.2 and remove explicit regularization: no dropout layers are added to the encoder-decoder architecture, and $\lambda = 0$ for all

optimizers. In addition, we examine the behavior of two different $\ell_2$-regularized instances of unitary scalarization: $\lambda = 10^{-4}$ for "Unit. Scal. $\ell_2$", $\lambda = 2 \times 10^{-3}$ for "Unit. Scal. $\ell_2$+". Figure 5.5 shows that SMTOs behave similarly to an $\ell_2$-penalized unitary scalarization. Importantly, SMTOs delay overfitting, requiring less early stopping compared to unitary scalarization to obtain comparable performance. In other words, early stopping is sufficient for unitary scalarization to perform on par with SMTOs. Moreover, overfitting is further reduced by "Unit. Scal. Reg.", which plots the regularized unitary scalarization from §5.4.1.2, with dropout layers and a weight decay of $\lambda = 10^{-3}$. Finally, Figure D.3(a) shows that unregularized unitary scalarization and most SMTOs rapidly drive the training loss of each task towards its global optimum. This suggests that the main difficulty of MTL is not associated with the optimization of its training objective, but rather to incorporating adequate regularization. Additional results are presented in appendix D.4.2.

### 5.5.2 Technical Results

All the methods considered in §5.5.1 regularize more than unitary scalarization. While RLW was shown to reduce overfitting by the original authors (Lin et al., 2022, theorem 2), we now provide a collection of novel and existing technical results that potentially explain the regularizing behavior of each of the other algorithms, complementing the presentation from §5.3. In particular, we show that MGDA, IMTL and PCGrad have a larger convergence set than unitary scalarization,

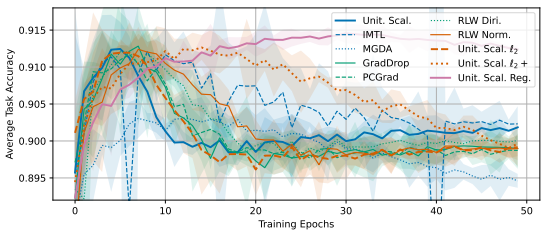



**Figure 5.5:** Mean and 95% CI (3 runs) avg. task validation accuracy over epochs on CelebA. SMTOs postpone the onset of overfitting, mirroring the effect of $\ell_2$ regularization on unitary scalarization.

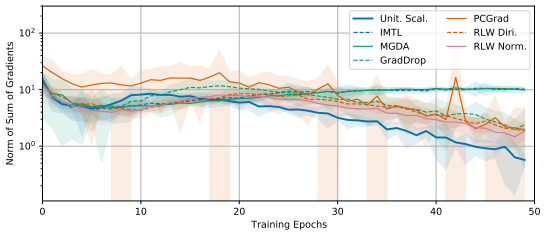



**Figure 5.6:** Mean and 95% CI (3 runs) for $\left\|\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|_2$ on CelebA. MGDA and IMTL converge away from stationary points of unitary scalarization, indicating under-optimization.

reducing the chances to land on sharp local minima (Dietterich, 1995). Furthermore, GradDrop and PCGrad introduce significant stochasticity, which is often linked to the same effect (Keskar et al., 2017; Kleinberg et al., 2018). We hope these observations will steer further research.

**MGDA** Let us denote the convex hull of a set $\mathcal{A}$ by $\text{Conv}(\mathcal{A})$. We now recall a well-known property of MGDA (Désidéri, 2012) and relate it to the behavior of unitary scalarization.

**Proposition 5.** *The* MGDA SMTO *(Sener and Koltun, 2018) converges to a superset of the convergence points of unitary scalarization. More specifically, it converges to any point* $\boldsymbol{\theta}_{\|}^*$ *such that:* $\mathbf{0} \in \text{Conv}(\{\nabla_{\boldsymbol{\theta}_{\|}^*} \mathcal{L}_i \mid i \in \mathcal{T}\})$.

See appendix D.2.1 for a simple proof. As a consequence of Proposition 5, MGDA does not necessarily reach a stationary point for $\mathcal{L}^{\text{MT}}$ (that is, a point for which $\sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i = \mathbf{0}$) or for any of the losses $\mathcal{L}_i$ ($\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i = \mathbf{0}$). For example, any point $\boldsymbol{\theta}_{\|}$ for which two per-task gradients point in opposite directions is Pareto stationary. On account of the well-known (Dietterich, 1995) relationship between under-optimizing (e.g., early stopping (Caruana et al., 2000; Li et al., 2020)) and overfitting, proposition 5 supports the interpretation of MGDA as a regularizer for equation (5.1). Empirical evidence that MGDA under-optimizes is provided in §5.5.3, Figure D.3(a), and Figure 5.5, which shows over-regularization. Proposition 5 can be trivially extended to the recent Nash-MTL, which shares the same convergence set (Navon et al., 2022, Theorem 5.4).

**IMTL** We now show that aggregating per-task gradients so that their cosine similarity is the same (equation (5.4)) yields a constrained steepest-descent algorithm (Proposition 6). This view on the update step of IMTL leads to a novel analysis of its convergence points (corollary 1). Proofs can be found in appendix D.2.2. We will denote by $\text{Aff}(\mathcal{A})$ the affine hull of a set $\mathcal{A}$.

**Proposition 6.** *IMTL by Liu et al. (2021c) updates* $\boldsymbol{\theta}_{\|}$ *by taking a step in the* steepest descent direction whose *cosine similarity with per-task gradients is the same across tasks.*

**Corollary 1.** IMTL *by Liu et al. (2021c) converges to a superset of the Pareto-stationary points for* $\boldsymbol{\theta}_{\|}$ *(and hence of the convergence points of the unitary scalarization). More specifically, it converges to any point* $\boldsymbol{\theta}_{\|}^*$ *such that:*

$$\mathbf{0} \in Aff\left(\left\{\nabla_{\boldsymbol{\theta}_{\|}^*}\mathcal{L}_i / \left\|\nabla_{\boldsymbol{\theta}_{\|}^*}\mathcal{L}_i\right\| \mid i \in \mathcal{T}\right\}\right).$$

As seen for MGDA, corollary 1 implies that, even if the employed model $f$ has the capacity to reach the minimal loss on $\mathcal{L}^{\text{MT}}$, IMTL may stop before reaching a stationary point. Recalling the relationship between under-optimizing and overfitting (Dietterich, 1995), this supports the interpretation of IMTL as a regularizer for equation (5.1). This is empirically shown in §5.5.3, Figures 5.5, D.3(a). In particular, unitary scalarization reaches the same average performance of IMTL but requires earlier stopping.

**PCGrad** We provide an alternative characterization of the PCGrad update rule, highlighting its stochasticity in the context of its interpretation as loss rescaling (Liu et al., 2021c; Lin et al., 2022). See appendix D.2.3 for a proof.

**Proposition 7.** *PCGrad is equivalent to a dynamic, and possibly stochastic, loss rescaling for* $\boldsymbol{\theta}_{\|}$*. At each iteration, per-task gradients are rescaled as follows:*

$$\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i \leftarrow \left(1 + \textstyle\sum_{j \in \mathcal{T}\setminus\{i\}} d_{ji}\right) \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i,\ d_{ji} \in \left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|}\right].$$

*Furthermore, if* $|\mathcal{T}| > 2$*,* $d_{ji}$ *is a random variable, and the above range contains its support.*

The results from proposition 7 can be easily extended to GradVac (Wang et al., 2021b), which generalizes PCGrad's projection onto the normal vector to arbitrary target cosine similarities between per-task gradients. When $|\mathcal{T}| > 2$,

PCGrad corresponds to a stochastic loss re-weighting. As such, PCGrad bears many similarities with Random Loss Weighting (RLW) (Lin et al., 2022). RLW proposes to sample scalarization weights from standard probability distributions at each iteration, and proves that this leads the better generalization (Lin et al., 2022, theorem 2). Indeed, it is well-known that adding noise to stochastic gradient estimations leads the optimization towards flatter minima, and that such minima may reduce overfitting (Keskar et al., 2017; Kleinberg et al., 2018). In line with the main technical results by Yu et al. (2020), we now restrict our focus to two-task problems, which allow for an easy description of PCGrad's convergence points. The result is largely based on (Yu et al., 2020, theorem 1): we relax some of the assumptions and provide a proof in appendix D.2.3.

**Corollary 2.** *If* $|\mathcal{T}| = 2$*, PCGrad will stop at any point where* $\cos(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2) = -1$*. Furthermore, if* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *are differentiable, and* $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{MT}$ *is L-Lipschitz with* $L > 0$*, PCGrad with step size* $t < \frac{1}{L}$ *converges to a superset of the convergence points of the unitary scalarization.*

Corollary 2 implies that, when $|\mathcal{T}| = 2$, PCGrad may under-optimize equation (5.1) as MGDA and IMTL. In particular, if $\cos(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2) = -1$, then $\mathbf{0} \in \text{Conv}(\{\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2\})$ (see proposition 5). We believe that PCGrad's stochasticity and enlarged convergence set potentially explain its regularizing effect.

**GradDrop** While the motivation behind GradDrop is to avoid entry-wise gradient conflicts across tasks, the main property of the method is to drive the optimization towards "joint minima": points that are stationary for all the individual tasks at once (Chen et al., 2020, proposition 1). In other words: $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i = \mathbf{0}\ \forall\ i \in \mathcal{T}$. While this property is desirable, we show that it holds beyond GradDrop, and independently of the gradient directions. Under strong assumptions on the model capacity, the above property would trivially hold for unitary scalarization (proposition 10, appendix D.2.4). Proposition 8 shows that it holds for a simple randomized version of unitary scalarization, which we name Random Grad Drop (RGD).

**Proposition 8.** *Let* $\mathcal{L}^{RGD}(\boldsymbol{\theta}_{\|}) := \sum_{i\in\mathcal{T}} u_i\mathcal{L}_i(\boldsymbol{\theta}_{\|})$*, where* $u_i \sim$ *Bernoulli*$(p)$ $\forall i \in \mathcal{T}$ *and* $p \in (0, 1]$*. The gradient* $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{RGD}$ *is always zero if and only if* $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0}$ $\forall i \in \mathcal{T}$*. In other words, the result from (Chen et al., 2020, proposition 1) can be obtained without any information on the sign of per-task gradients.*

Proposition 8 (see appendix D.2.4 for a simple proof) shows that an inexpensive sign-independent stochastic scalarization shares GradDrop's main reported property. $\mathcal{L}^{\text{RGD}}$ can be directly cast an instance of RLW, and hence as a regularization method (Keskar et al., 2017; Kleinberg et al., 2018). Furthermore, Figure D.6 in appendix D.4.3 shows that the empirical results of GradDrop on CelebA (Liu et al., 2015) are closely matched by a sign-agnostic gradient masking, partly undermining the conflicting gradients assumption. We believe that the above results, along with the authors' original experiments showing that GradDrop delays overfitting on CelebA (Chen et al., 2020, figure 3), suggest that GradDrop behaves as a regularizer.

### 5.5.3 Under-Optimization: Empirical Study

As seen in §5.5.2, MGDA and IMTL might under-optimize equation (5.1) compared to unitary scalarization due to their larger convergence sets. In order to assess whether this is empirically the case, we estimate $\left\|\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|_2$, the norm of the unitary scalarization update on shared parameters $\boldsymbol{\theta}_{\|}$, for all optimizers throughout the unregularized CelebA experiment from §5.5.1. Large magnitudes for $\left\|\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|_2$ towards convergence would indicate that SMTOs steer optimization far from stationary points of unitary scalarization, resulting in under-optimization. We compute the update norm on the mini-batch loss every 100 updates, and report the per-epoch average in Figure 5.6. Compared with unitary scalarization, most SMTOs have smaller or comparable update magnitude in the first 15 epochs. However, towards convergence, SMTOs display larger $\left\|\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|_2$ compared to unitary scalarization. In particular, IMTL and MGDA have the largest norm, denoting significant empirical under-optimization. The additional stochasticity of RLW, PCGrad, and GradDrop also appears to lead to larger norm values than unitary scalarization, yet to a lesser degree. Given that MGDA and IMTL

incur a larger loss than unitary scalarization in later epochs (see Figure D.3(a) in appendix D.4.2), we can conclude that they guide optimization towards regions of the parameter space that under-optimize equation (5.1), providing empirical support for our analysis.

## 5.6 Conclusions

This chapter made two main contributions. First, we evaluated popular SMTOs using a single experimental pipeline, including previously unpublished results of MGDA, IMTL, RLW, and GradDrop in the RL setting. Surprisingly, our evaluation showed that none of the SMTOs consistently outperform unitary scalarization, the simplest and least expensive method. Second, in order to explain our surprising results, we postulate that SMTOs act as regularizers and present an analysis that supports our hypothesis. We believe our work calls for further reevaluation of progress in developing principled and efficient MTL algorithms.

We conclude by addressing the limitations of our work. While we covered a wide range of popular benchmarks, we do not exclude the existence of settings where unitary scalarization underperforms: discovering them is an interesting direction for future work. Furthermore, our experimental results were obtained via grid searches under limited compute resources: some of the methods might benefit from further fine-tuning. Nevertheless, we remark that fine-tuning will be easier for unitary scalarization due to its shorter runtimes. Finally, we presented the regularization hypothesis only as a partial explanation of our results: we hope it will steer further analysis and consequently improve the understanding of MTL.

# 6 Conclusions

## Contents


6.1 Summary . . . . . **139**
6.2 Discussion and Future Work . . . . . **140**
6.2.1 Neural Network Verification . . . . . 140
6.2.2 Training for Verifiability . . . . . 141
6.2.3 Deep Multi-Task Learning . . . . . 142

## 6.1 Summary

Despite their remarkable empirical success, deep learning systems are still far from being deployable in safety-critical applications. This thesis attempts to move in this direction by presenting contributions in the areas of neural network verification, whose implementations are publicly available as part of the OVAL verification framework at `https://github.com/oval-group/oval-bab`, and on two training problems related to trustworthiness.

In chapter 2, we present two incomplete verifiers based on Lagrangian Decomposition. The two algorithms solve a popular neural network relaxation, commonly referred to as the convex barrier, representing the convex hull of element-wise activation functions. We design a massively parallel Branch-and-Bound framework around the solvers, demonstrating their efficacy for complete verification.

In chapter 3, we overcome the convex barrier on piecewise-linear networks by designing two memory-efficient dual solvers for a recent tighter relaxation. We demonstrate that these algorithms can yield better bounds in the same time with respect to relevant primal approaches. Furthermore, we show their advantages for harder verification properties compared to our solvers from chapter 2.

In chapter 4, we introduce an intuitive and inexpensive algorithm for verified network training. Our approach successfully exploits tight bounds from Branch-and-Bound-based verifiers, and attains state-of-the-art results on small perturbations on CIFAR-10 without resorting to complex layer-wise training schemes (Balunovic and Vechev, 2020). We additionally present a branching strategy that, by re-using dual information from a recent bounding algorithm (Wang et al., 2021a), reduces branching costs without affecting the quality of the splits.

In chapter 5, we conduct a comprehensive experimental evaluation of optimizers for deep multi-task learning. We demonstrate that, in spite of the added complexity and overhead, none of these optimizers consistently outperforms a simple baseline. We provide a partial explanation of the results based on regularization, which we back theoretically and empirically.

## 6.2 Discussion and Future Work

We conclude with a critical discussion of the contributions presented in this thesis, possibly commenting on relevant work that appeared since their publication. We highlight open challenges in the fields of interest and promising avenues for future work.

### 6.2.1 Neural Network Verification

Bunel et al. (2020a) (presented in chapter 2) showed that dual iterative algorithms operating on the triangle relaxation (see §1.2.1), coupled with activation splitting and the parallelism of modern GPU architectures, could significantly improve on the state-of-the-art for complete neural network verification. As a result, our work spearheaded a successful line of work that replaced the decomposition-based bounding with methods based on linear bound propagation. In particular, by appropriately optimizing the slope of the bounding functions (which are usually kept constant, see §2.2.2.1), it was shown that one can obtain a solver for the triangle relaxation with better empirical speed-accuracy trade-offs than our methods based on Lagrangian decomposition (Xu et al., 2021). The drawbacks of this approach when applied for complete verification can be overcome by using Lagrangian multipliers for the activation splits, obtaining a state-of-the-art complete verifier (Wang et al., 2021a) when using our FSB branching strategy (see chapter 2) from De Palma et al. (2021c). Our solvers for the tighter relaxation by Anderson et al. (2020), presented in chapter 3 (De Palma et al., 2021a,b), show the benefits of using more accurate relaxations on harder verification properties. However, their speed-accuracy trade-offs could be potentially improved by adopting a formulation amenable to linear bound propagation: doing so is an interesting avenue for future work. In chapter 2, we demonstrate that our hand-designed heuristic branching strategy outperforms a splitting method based on deep learning (Lu and Kumar, 2020). The efficacy of our branching strategy and its adoption in the literature testify to the success of heuristic methods for activation splitting. However, while branching heuristics focus on short-term bounding improvement, an ideal splitting decision should take long-term

effects into account. Hence, given the cardinality of set of the possible branching decisions, we believe that large improvements could be potentially obtained via the principled use of learning algorithms. In addition, further scalability improvements could be obtained by leveraging machine learning in other components of complete verification frameworks, an example being automatically deciding on the tightness of the bounds required by a given subproblem.

Finally, most of the verification community so far has focussed on standard feedforward networks and on local robustness specifications, where the input domain is a $\ell_p$ ball around specific inputs and the output must satisfy a set of linear inequalities. However, the verification of semantic specifications, such as those that may occur in autonomous driving (for instance, ensuring right of way is appropriately given) or scientific applications, remains a challenge, as does the verification of global specifications (property holding for any possible network input). Likewise, we are still far from the efficient complete verification of models widespread in natural language processing, such as transformers. We believe that future work should focus on incorporating of a wider range of verification properties, which should be accompanied by the adoption of conceptually different benchmarks inspired by real-world applications.

### 6.2.2 Training for Verifiability

In order to bridge the gap between state-of-the-art network architectures and those commonly employed in the context of verification, progress in verifiers should be complemented by advances in verified training algorithms. The performance of current approaches (such as our work presented in chapter 4) suggests the existence of fundamental trade-offs between standard network performance and verifiability. The higher cost per-iteration compared to standard training approaches and the presence of additional hyper-parameters severely limit the scalability of these training schemes. To make things worse, robustness appears to have larger capacity requirements than standard performance metrics (Madry et al., 2018). Furthermore, the algorithms that perform well on small perturbations, based on enhancing adversarial training,

are fundamentally different from state-of-the-art methods on large perturbations, based on backpropagating through incomplete verifiers (see §4.6.1). The reasons for this discrepancy are yet to be fully understood. We believe that significant research efforts should be directed towards addressing these limitations, possibly resulting in radically different approaches to verified network training. These advances would likely require a deeper understanding of what makes a network easily verifiable by recent Branch-and-Bound-based approaches.

### 6.2.3 Deep Multi-Task Learning

By highlighting the competitive performance of a simple baseline, in line with prior work across various deep learning problems (Brockschmidt, 2020; Narang et al., 2021), our work in chapter 5 casts doubts on the validity of the progress on specialized multi-task optimizers. The concurrent presentation of similar results (Xin et al., 2022) at the same venue as our original work (Kurin et al., 2022) evidences the timeliness of our investigation. While our work relies on an analysis of the regularizing properties of SMTOs to explain the results, Xin et al. (2022) focus on the complementary aspect of scalarization coefficients. We hope that these findings will stimulate further research in the area, so as to increase trust in MTL pipelines.

We conclude by pointing out that training for multiple tasks at once has been demonstrated to improve a network's robustness to adversarial examples (Mao et al., 2020). Therefore, a natural extension of the work presented in this thesis would be the investigation of topics at the intersection of MTL and verifiability. Potential steps in this direction could include examining the effect, if any, of SMTOs on verifiability, or incorporating MTL within existing verified training pipelines.

# Appendices

# A

# Improved Branch and Bound for Neural Network Verification via Lagrangian Decomposition

**Contents**


A.1 Proof of theorem 1 . . . . . **145**
A.2 Proof of proposition 2 . . . . . **149**
A.3 Sigmoid Activation function . . . . . **152**
A.3.1 Convex hull computation . . . . . 153
A.3.2 Solving the $\mathcal{P}_k$ subproblems over sigmoid activation . . . 156
A.4 Momentum for the Proximal solver . . . . . **157**
A.5 Supplementary Incomplete Verification Experiments . . . . . **159**

## A.1 Proof of theorem 1

**Theorem 1.** *Let us assume $\sigma_k(\hat{\mathbf{x}}_k) = \max(\mathbf{0}, \hat{\mathbf{x}}_k)\ \forall\ k \in [\![1, n-1]\!]$. For dual point $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ of problem (2.6) by Dvijotham et al. (2018b) yielding bound $d(\boldsymbol{\mu}, \boldsymbol{\lambda})$, it holds that $q(\boldsymbol{\mu}) \geq d(\boldsymbol{\mu}, \boldsymbol{\lambda})$. Furthermore, if $\boldsymbol{\lambda}'_{n-1} = -W_n^T \mathbf{1}$ and $\boldsymbol{\lambda}'_{k-1} = W_k^T \boldsymbol{\mu}_k$ for $k \in [\![2, n-1]\!]$, then $q(\boldsymbol{\mu}) = d(\boldsymbol{\mu}, \boldsymbol{\lambda}')$.*

*Proof.* We will show the relation between the dual problem by Dvijotham et al. (2018b) and the one proposed in chapter 2. We will show that, in the context of ReLU activation functions, from any dual point of their dual providing a given bound, our dual provides a bound at least as tight. Moreover, our dual coincides with the one by Dvijotham et al. (2018b) if the latter is modified to include additional equality constraints.

Let us write $\sigma(\cdot) = \max(0, \cdot)$. We start from problem[1] (2.6) by Dvijotham et al. (2018b). Decomposing it, in the same way that Dvijotham et al. (2018b) do it in order to obtain their equation (7), and with the convention that $\boldsymbol{\mu}_n = -1$) we obtain:

$$\begin{aligned} b_n - \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T \mathbf{b}_k + \sum_{k=1}^{n-1} \min_{\hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k]} \left( \boldsymbol{\mu}_k^T \hat{\mathbf{x}}_k - \boldsymbol{\lambda}_k^T \sigma(\hat{\mathbf{x}}_k) \right) \\ + \sum_{k=1}^{n-1} \min_{\mathbf{x}_k \in [\sigma(\hat{\mathbf{l}}_k), \sigma(\hat{\mathbf{u}}_k)]} \left( -\boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k + \boldsymbol{\lambda}_k^T \mathbf{x}_k \right) + \min_{\mathbf{x}_0 \in \mathcal{C}} -\boldsymbol{\mu}_1^T W_1 \mathbf{x}_0. \end{aligned} \tag{A.1}$$

With the convention that $\boldsymbol{\rho}_n = -1$, our dual (2.11) can be decomposed as:

$$q(\boldsymbol{\rho}) = \sum_{k=1}^{n-1} \left( \min_{\substack{\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1} \\ \in \mathcal{P}_k(\cdot,\cdot,\cdot)}} \boldsymbol{\rho}_k^T \hat{\mathbf{x}}_{B,k} - \boldsymbol{\rho}_{k+1}^T \hat{\mathbf{x}}_{A,k+1} \right) + \min_{\substack{\mathbf{x}_0, \hat{\mathbf{x}}_{A,1} \\ \in \mathcal{P}_0(\cdot,\cdot)}} -\boldsymbol{\rho}_1^T \hat{\mathbf{x}}_{A,1}, \tag{A.2}$$

We will show that when we choose the dual variables such that

$$\boldsymbol{\rho}_k = \boldsymbol{\mu}_k, \tag{A.3}$$

we obtain a tighter bound than the ones given by (A.1).

[1] Our activation function $\sigma$ is denoted $h$ in their paper. The equivalent of $\mathbf{z}$ in their paper is $\mathbf{x}$ in ours, while the equivalent of their $\mathbf{x}$ is $\hat{\mathbf{x}}$. Also note that their paper use the computation of upper bounds as examples while ours use the computation of lower bounds.

We will start by showing that the term being optimized over $\mathcal{P}_0$ is equivalent to some of the terms in (A.1):

$$\min_{\substack{\mathbf{x}_0, \hat{\mathbf{x}}_{A,1} \\ \in \mathcal{P}_0(\cdot,\cdot)}} -\boldsymbol{\rho}_1^T \hat{\mathbf{x}}_{A,1} = \min_{\substack{\mathbf{x}_0, \hat{\mathbf{x}}_{A,1} \\ \in \mathcal{P}_0(\cdot,\cdot)}} -\boldsymbol{\mu}_1^T \hat{\mathbf{x}}_{A,1} = -\boldsymbol{\mu}_1^T \mathbf{b}_1 + \min_{\mathbf{x}_0 \in \mathcal{C}} -\boldsymbol{\mu}_1^T W_1 \mathbf{x}_0 \quad \text{(A.4)}$$

The first equality is given by the replacement formula (A.3), while the second one is given by performing the replacement of $\hat{\mathbf{x}}_{A,1}$ with his expression in $\mathcal{P}_0$.

We will now obtain a lower bound of the term being optimized over $\mathcal{P}_k$. Let's start by plugging in the values using the formula (A.3) and replace the value of $\hat{\mathbf{x}}_{A,k+1}$ using the equality constraint.

$$\begin{aligned} &\min_{\substack{\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1} \\ \in \mathcal{P}_k(\cdot,\cdot,\cdot)}} \left(\boldsymbol{\rho}_k^T \hat{\mathbf{x}}_{B,k} \; - \; \boldsymbol{\rho}_{k+1}^T \hat{\mathbf{x}}_{A,k+1}\right) = \\ &\quad = -\boldsymbol{\mu}_{k+1}^T \mathbf{b}_{k+1} + \min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \\ (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in \text{Conv}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(\boldsymbol{\mu}_k^T \hat{\mathbf{x}}_{B,k} \; - \; \boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k\right) \end{aligned} \quad \text{(A.5)}$$

Focusing on the second term that contains the minimization over the convex hull, we will obtain a lower bound. It is important, at this stage, to recall that, as seen in section 2.3.2.2, the minimum of the second term can either be one of the three vertices of the triangle in Figure 2.1 (ambiguous ReLU), the $\hat{\mathbf{x}}_{B,k} = \mathbf{x}_k$ line (passing ReLU), or the $(\hat{\mathbf{x}}_{B,k} = \mathbf{0}, \mathbf{x}_k = \mathbf{0})$ triangle vertex (blocking ReLU). We will write $(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k) \in \texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$.

We can add a term $\boldsymbol{\lambda}_k^T(\sigma(\hat{\mathbf{x}}_k) - \sigma(\hat{\mathbf{x}}_k)) = 0$ and obtain:

$$\min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \\ (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in \text{Conv}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(\boldsymbol{\mu}_k^T \hat{\mathbf{x}}_{B,k} \; - \; \boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k\right) \quad \text{(A.6)}$$

$$= \min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \\ \texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(\boldsymbol{\mu}_k^T \hat{\mathbf{x}}_{B,k} \; - \; \boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k\right)$$

$$= \min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \\ \texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(\boldsymbol{\mu}_k^T \hat{\mathbf{x}}_{B,k} \; - \; \boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k + \boldsymbol{\lambda}_k^T(\sigma(\hat{\mathbf{x}}_{B,k}) - \sigma(\hat{\mathbf{x}}_{B,k}))\right)$$

$$
\begin{aligned}
\geq & \min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \\ \texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(\boldsymbol{\mu}_k^T \hat{\mathbf{x}}_{B,k} - \boldsymbol{\lambda}_k^T \sigma\left(\hat{\mathbf{x}}_{B,k}\right)\right) \\
& + \min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \\ \texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(-\boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k + \boldsymbol{\lambda}_k^T \sigma\left(\hat{\mathbf{x}}_{B,k}\right)\right) \\
\geq & \min_{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k]} \left(\boldsymbol{\mu}_k^T \hat{\mathbf{x}}_{B,k} - \boldsymbol{\lambda}_k^T \sigma\left(\hat{\mathbf{x}}_{B,k}\right)\right) + \min_{\mathbf{x}_k \in [\sigma(\hat{\mathbf{l}}_k), \sigma(\hat{\mathbf{u}}_k)]} \left(-\boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k + \boldsymbol{\lambda}_k^T \mathbf{x}_k\right).
\end{aligned}
$$

Equality between the second line and the third comes from the fact that we are adding a term equal to zero. The inequality between the third line and the fourth is due to the fact that the sum of minimum is going to be lower than the minimum of the sum. For what concerns obtaining the final line, the first term comes from projecting $\mathbf{x}_k$ out of the feasible domain and taking the convex hull of the resulting domain. We need to look more carefully at the second term. Plugging in the ReLU formula:

$$
\begin{aligned}
& \min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \\ \texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(-\boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k + \boldsymbol{\lambda}_k^T \max\{0, \hat{\mathbf{x}}_{B,k}\}\right) \\
= & \min_{\substack{\hat{\mathbf{x}}_{B,k} \in [\sigma(\hat{\mathbf{l}}_k), \sigma(\hat{\mathbf{u}}_k)] \\ \texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)}} \left(-\boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k + \boldsymbol{\lambda}_k^T \hat{\mathbf{x}}_{B,k}\right) \\
\geq & \min_{\mathbf{x}_k \in [\sigma(\hat{\mathbf{l}}_k), \sigma(\hat{\mathbf{u}}_k)]} \left(-\boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k + \boldsymbol{\lambda}_k^T \mathbf{x}_k\right),
\end{aligned}
$$

as (keeping in mind the shape of `ReLU_sol` and for the purposes of this specific problem) excluding the negative part of the $\hat{\mathbf{x}}_{B,k}$ domain does not alter the minimal value. The final line then follows by observing that forcing $\hat{\mathbf{x}}_{B,k} = \mathbf{x}$ is a convex relaxation of the positive part of `ReLU_sol`. Summing up the lower bounds for all the terms in (A.2), as given by equations (A.4) and (A.6), we obtain the formulation of Problem (A.1). We conclude that the bound obtained by the original dual by Dvijotham et al. (2018b) is necessarily no larger than the bound derived using our dual. Given that we are computing lower bounds, this means that their bound is looser.

Finally, we now prove that our dual (2.11) coincides with the following modified version of problem (2.6):

$$\begin{aligned}
&\max_{\boldsymbol{\mu},\boldsymbol{\lambda}} \;\; d_O(\boldsymbol{\mu},\boldsymbol{\lambda}) \\
\text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\
& (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in [\sigma(\hat{\mathbf{l}}_k), \sigma(\hat{\mathbf{u}}_k)] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \qquad k \in [\![1, n-1]\!]\,, \\
& \boldsymbol{\lambda}'_{n-1} = -W_n^T \mathbf{1}, \quad \boldsymbol{\lambda}'_{k-1} = W_k^T \boldsymbol{\mu}_k \qquad k \in [\![2, n-1]\!]\,,
\end{aligned}$$

whose objective, keeping the convention that $\boldsymbol{\mu}_n = -1$, can be rewritten as:

$$\begin{aligned}
d'_O(\boldsymbol{\mu}) = \min_{\hat{\mathbf{x}}} \; & \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T (\hat{\mathbf{x}}_k - \mathbf{b}_k) - \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T W_k \sigma(\hat{\mathbf{x}}_k) - \boldsymbol{\mu}_1^T W_1 \mathbf{x}_0 + \mathbf{b}_n \\
\text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \qquad \hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \quad k \in [\![1, n-1]\!]\,.
\end{aligned} \tag{A.7}$$

Exploiting equations (A.4) and (A.5), and re-using the notation for $\texttt{ReLU\_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ and $\boldsymbol{\mu}_n = -1$, we obtain the following reformulation of $q(\boldsymbol{\mu})$:

$$\begin{aligned}
q(\boldsymbol{\mu}) = \min_{\mathbf{x},\hat{\mathbf{x}}} \; & -\sum_{k=1}^{n} \boldsymbol{\mu}_k^T \mathbf{b}_k + \sum_{k=1}^{n-1} \left( \boldsymbol{\mu}_k^T \hat{\mathbf{x}}_{B,k} \; - \; \boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k \right) - \boldsymbol{\mu}_1^T W_1 \mathbf{x}_0 \\
\text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\
& \hat{\mathbf{x}}_{B,k} \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \qquad\qquad k \in [\![1, n-1]\!]\,, \\
& (\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k) \in \texttt{ReLU_sol}(\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) \qquad k \in [\![1, n-1]\!]\,.
\end{aligned}$$

In order for the above equation to be equal to (A.7), noting that we can substitute $\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{B,k}$, we only need to prove the following for $k \in [\![1, n-1]\!]$:

$$\begin{aligned}
&\min_{\hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k]} \quad \left( \boldsymbol{\mu}_k^T \hat{\mathbf{x}}_k \; - \; \boldsymbol{\mu}_{k+1}^T W_{k+1} \mathbf{x}_k \right) \\
&(\hat{\mathbf{x}}_k, \mathbf{x}_k) \in \texttt{ReLU_sol}(\hat{\mathbf{x}}_k, \mathbf{x}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) \\
&\qquad\qquad = \min_{\hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k]} \left( \boldsymbol{\mu}_k^T \hat{\mathbf{x}}_k \; - \boldsymbol{\mu}_{k+1}^T W_{k+1} \sigma(\hat{\mathbf{x}}_k) \right),
\end{aligned}$$

which holds trivially for blocking or passing ReLUs. In the case of ambiguous ReLUs, instead, it suffices to point out that, due to the piecewise-linearity of the objective, the right hand side corresponds to: $\min\{\boldsymbol{\mu}_k^T[i]\hat{\mathbf{l}}_k[i], 0, \boldsymbol{\mu}_k^T[i]\hat{\mathbf{u}}_k[i] - (\boldsymbol{\mu}_{k+1}^T W_{k+1})[i]\hat{\mathbf{u}}_k[i]\}$. Looking at equation (2.17), we can see that this minimization is hence identical to the to the right hand side, proving the second part of the theorem. □

## A.2 Proof of proposition 2

**Proposition 2.** *Let $d_P$ be a lower bound to problem* (2.1) *obtained via a propagation-based bounding algorithm. Then, if $\sigma_k(\hat{\mathbf{x}}_k) = \max(\mathbf{0}, \hat{\mathbf{x}}_k)\ \forall\ k \in [\![1, n-1]\!]$, there exist some dual points $\bar{\boldsymbol{\rho}}$ and $\left(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}\right)$ such that $q(\bar{\boldsymbol{\rho}}) = d_O\left(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}\right) = \bar{d}_P$, and both $\bar{\boldsymbol{\rho}}$ and $\left(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}\right)$ can be computed at the cost of a backward pass through the network.*

*Proof.* We will show that, for ReLU activations, the solution generated by propagation-based methods (including amongst others, CROWN (Zhang et al., 2018), and the algorithm by Wong and Kolter (2018), see §2.2.2.1), can be used to provide initialization to both our dual (2.11) and dual (2.6) by Dvijotham et al. (2018b). This holds in spite of the differences between the three dual derivations, which consist of an application of Lagrangian Decomposition, and of the Lagrangian relaxations of two different problems.

Recall that $\underline{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k) = \underline{\mathbf{a}}_k \hat{\mathbf{x}}_k + \underline{\mathbf{b}}_k$ and $\bar{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k) = \bar{\mathbf{a}}_k \hat{\mathbf{x}}_k + \bar{\mathbf{b}}_k$ are two linear functions that bound $\sigma_k(\hat{\mathbf{x}}_k)$ respectively from below and above. Before proceeding with the proof, we point out that, relying on the definition of the convex hull of the ReLU in (2.8), the following is true for any $\underline{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k)$ and $\bar{\boldsymbol{\sigma}}_k(\hat{\mathbf{x}}_k)$ which are not unnecessarily loose (see Figure 2.1):

$$
\begin{aligned}
0 \leq \bar{\mathbf{a}}_k = \frac{\hat{\mathbf{u}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} \leq 1, \quad \bar{\mathbf{b}}_k = \frac{-\hat{\mathbf{l}}_k \odot \hat{\mathbf{u}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} \quad & 0 \leq \underline{\mathbf{a}}_k \leq 1 \qquad \text{if } \hat{\mathbf{l}}_k \leq 0 \text{ and } \hat{\mathbf{u}}_k \geq 0, \\
& \bar{\mathbf{a}}_k = \underline{\mathbf{a}}_k = 0 \qquad \text{if } \hat{\mathbf{u}}_k \leq 0, \\
& \bar{\mathbf{a}}_k = \underline{\mathbf{a}}_k = 1 \qquad \text{if } \hat{\mathbf{l}}_k \geq 0, \\
& \underline{\mathbf{b}}_k = 0 \qquad \text{in all cases.}
\end{aligned}
\tag{A.8}
$$

We start by proving that, by taking as our solution of (2.11):

$$
\boldsymbol{\rho} = \bar{\boldsymbol{\mu}}, \qquad \text{where } \bar{\boldsymbol{\mu}} \text{ is defined as per equation (2.4)}, \tag{A.9}
$$

we obtain exactly the same bound given by plugging equation (2.4) into equation (2.5).

Recall that, given a choice of $\boldsymbol{\rho}$, the bound that we generate is:

$$\begin{aligned}\min_{\mathbf{x},\hat{\mathbf{x}}}\ & \hat{x}_{A,n} + \sum_{k=1}^{n-1} \boldsymbol{\rho}_k^T \left(\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}\right) \\ \text{s.t.}\quad & \mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}), \\ & \mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}) \qquad k \in [\![1, n-1]\!]\,,\end{aligned}$$

which, using the dummy variable $\boldsymbol{\rho}_n = -\mathbf{1}$ for ease of notation, can be decomposed into several subproblems:

$$\sum_{k=1}^{n-1} \left( \min_{\substack{\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1} \\ \in \mathcal{P}_k\left(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}\right)}} \boldsymbol{\rho}_k^T \hat{\mathbf{x}}_{B,k} \ - \ \boldsymbol{\rho}_{k+1}^T \hat{\mathbf{x}}_{A,k+1} \right) + \min_{\substack{\mathbf{x}_0, \hat{\mathbf{x}}_{A,1} \\ \in \mathcal{P}_0\left(\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}\right)}} -\boldsymbol{\rho}_1^T \hat{\mathbf{x}}_{A,1}. \tag{A.10}$$

Let's start by replacing $\boldsymbol{\rho}_1$ by $\bar{\boldsymbol{\mu}}_1$, in accordance with (A.9). The problem over $\mathcal{P}_0$ becomes:

$$\begin{aligned}\min_{\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}}\ & -\bar{\boldsymbol{\mu}}_1^T \hat{\mathbf{x}}_{A,1} \\ \text{s.t.}\quad & \hat{\mathbf{x}}_1 = W_1 \mathbf{x}_0 + \mathbf{b}_1 \\ & \mathbf{x}_0 \in \mathcal{C}.\end{aligned}$$

Substituting the equality into the objective of the subproblem, we get equation (A.4):

$$\min_{\mathbf{x}_0 \in \mathcal{C}} \quad -\bar{\boldsymbol{\mu}}_1^T W_1 \mathbf{x}_0 - \bar{\boldsymbol{\mu}}_1^T \mathbf{b}_1.$$

We will now evaluate the values of the problem over $\mathcal{P}_k$. Recall that these subproblems take the following form:

$$\begin{aligned}[\hat{\mathbf{x}}^*_{B,k}, \hat{\mathbf{x}}^*_{A,k+1}] = \operatorname*{argmin}_{\hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}}\ & \boldsymbol{\rho}_k^T \hat{\mathbf{x}}_{B,k} - \boldsymbol{\rho}_{k+1}^T \hat{\mathbf{x}}_{A,k+1} \\ \text{s.t}\quad & (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in \mathrm{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k), \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in [\sigma_k(\hat{\mathbf{l}}_k), \sigma_k(\hat{\mathbf{u}}_k)] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k], \\ & \hat{\mathbf{x}}_{A,k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1}.\end{aligned}$$

Merging the last equality constraint into the objective function:

$$\begin{aligned}\min_{\hat{\mathbf{x}}_{B,k}, \mathbf{x}_k}\ & \boldsymbol{\rho}_k^T \hat{\mathbf{x}}_{B,k} - \boldsymbol{\rho}_{k+1}^T W_{k+1}\mathbf{x}_k - \boldsymbol{\rho}_{k+1}^T \mathbf{b}_{k+1} \\ \text{s.t}\quad & (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in \mathrm{Conv}(\sigma_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k), \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in [\sigma_k(\hat{\mathbf{l}}_k), \sigma_k(\hat{\mathbf{u}}_k)] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k].\end{aligned}$$

We now distinguish the different cases of ReLU: passing ($\mathcal{I}^+$), blocking $\mathcal{I}^-$, and ambiguous ($\mathcal{I}$), recalling that the problem above decomposes over the ReLUs of layer $k$, the $i$-th ReLU being associated to the $i$-th entries of $\mathbf{x}_k$ and $\hat{\mathbf{x}}_k$.

If $i \in \mathcal{I}^+$, we have $\text{Conv}(\sigma_k[i], \hat{\mathbf{l}}_k[i], \hat{\mathbf{u}}_k[i]) \equiv \{(\mathbf{x}_k[i], \hat{\mathbf{x}}_{B,k}[i]) | \hat{\mathbf{x}}_{B,k}[i] = \mathbf{x}_k[i]\}$. Then, the objective function term for $j$ becomes:

$$\begin{aligned} \min_{\hat{\mathbf{x}}_{B,k}[i]} \quad & \left(\boldsymbol{\rho}_k^T[i] - \boldsymbol{\rho}_{k+1}^T[i] W_{k+1}\right) \hat{\mathbf{x}}_{B,k}[i] - \boldsymbol{\rho}_{k+1}^T[i] \mathbf{b}_{k+1}[i] \\ \text{s.t} \quad & \hat{\mathbf{x}}_{B,k}[i] \in [\hat{\mathbf{l}}_k[i], \hat{\mathbf{u}}_k[i]]. \end{aligned}$$

Using equations (2.4) and (A.9), and pointing out that $\bar{\mathbf{a}}_k[i] = \underline{\mathbf{a}}_k[i] = 1$ for $j \in \mathcal{I}^+$ due to equation (A.8), the minimum of the subproblem above simplifies to: $-\bar{\boldsymbol{\mu}}_{k+1}^T[i] \mathbf{b}_{k+1}[i]$.

If $i \in \mathcal{I}^-$, we have $\text{Conv}(\sigma_k[i], \hat{\mathbf{l}}_k[i], \hat{\mathbf{u}}_k[i]) \equiv \{(\mathbf{x}_k[i], \hat{\mathbf{x}}_{B,k}[i]) | \mathbf{x}_k[i] = 0\}$. Replacing $\mathbf{x}_k[i] = 0$:

$$\begin{aligned} \min_{\hat{\mathbf{x}}_{B,k}[i]} \quad & \boldsymbol{\rho}_k^T[i] \hat{\mathbf{x}}_{B,k}[i] - \boldsymbol{\rho}_{k+1}^T[i] \mathbf{b}_{k+1}[i] \\ \text{s.t} \quad & \hat{\mathbf{x}}_{B,k}[i] \in [\hat{\mathbf{l}}_k[i], \hat{\mathbf{u}}_k[i]]. \end{aligned}$$

Using equations (2.4) and (A.9), and pointing out that $\bar{\mathbf{a}}_k[i] = \underline{\mathbf{a}}_k[i] = 0$ for $j \in \mathcal{I}^-$ due to equation (A.8), the minimum of the subproblem above simplifies again to: $-\bar{\boldsymbol{\mu}}_{k+1}^T[i] \mathbf{b}_{k+1}[i]$.

The case for $i \in \mathcal{I}$ is more complex. We apply the same strategy that led to equation (2.17), keeping all constant terms in the objective. This leads us to the following subproblem:

$$\underset{(\hat{\mathbf{x}}_{B,k}[i], \mathbf{x}_k[i]) \in \{(\hat{\mathbf{l}}_k[i], 0), (0,0), (\hat{\mathbf{u}}_k[i], \hat{\mathbf{u}}_k[i])\}}{\text{argmin}} \begin{bmatrix} \left(\boldsymbol{\rho}_k[i] \hat{\mathbf{x}}_{B,k}[i] - (\boldsymbol{\rho}_{k+1}^T W_{k+1})[i] \mathbf{x}_k[i]\right) + \\ -\boldsymbol{\rho}_{k+1}[i] \mathbf{b}_{k+1}[i] \end{bmatrix},$$

which amounts to:

$$\min \left\{ \boldsymbol{\rho}_k[i] \hat{\mathbf{l}}_k[i], \left(\boldsymbol{\rho}_k^T - \boldsymbol{\rho}_{k+1}^T W_{k+1}\right)[i] \hat{\mathbf{u}}_k[i], 0 \right\} - \boldsymbol{\rho}_{k+1}[i] \mathbf{b}_{k+1}[i].$$

Using equation (A.9), equation (2.4) and $\bar{\boldsymbol{\mu}}_n = -1$, we obtain:

$$\min \left\{ \begin{array}{l} \left(\bar{\mathbf{a}}_k \odot [\bar{\boldsymbol{\lambda}}_k]_+ + \underline{\mathbf{a}}_k \odot [\bar{\boldsymbol{\lambda}}_k]_-\right)[i]\hat{\mathbf{l}}_k[i], \\ \left(\left(\bar{\mathbf{a}}_k \odot [\bar{\boldsymbol{\lambda}}_k]_+ + \underline{\mathbf{a}}_k \odot [\bar{\boldsymbol{\lambda}}_k]_-\right) - \bar{\boldsymbol{\lambda}}_k\right)[i]\hat{\mathbf{u}}_k[i], \\ 0, \end{array} \right\} - \bar{\boldsymbol{\mu}}_{k+1}^T[i]\mathbf{b}_{k+1}[i]$$

$$\overset{\text{using (A.8)}}{=} \min \left\{ \begin{array}{l} -\bar{\mathbf{b}}_k[i][\bar{\boldsymbol{\lambda}}_k]_+[i] + \hat{\mathbf{l}}_k[i]\underline{\mathbf{a}}_k[i][\bar{\boldsymbol{\lambda}}_k]_-[i], \\ \left((\bar{\mathbf{a}}_k - 1) \odot [\bar{\boldsymbol{\lambda}}_k]_+ + (\underline{\mathbf{a}}_k - 1) \odot [\bar{\boldsymbol{\lambda}}_k]_-\right)[i]\hat{\mathbf{u}}_k[i], \\ 0, \end{array} \right\} - \bar{\boldsymbol{\mu}}_{k+1}^T[i]\mathbf{b}_{k+1}[i]$$

Let us make the following observations: (i) due to equation (A.8) and $\hat{\mathbf{u}}_k[i] > 0$ (as $i \in \mathcal{I}$), the second argument of the minimum is always non-negative, (ii) $-\bar{\mathbf{b}}_k[i][\bar{\boldsymbol{\lambda}}_k]_+[i] \leq 0$, (iii) $\hat{\mathbf{l}}_k[i]\underline{\mathbf{a}}_k[i][\bar{\boldsymbol{\lambda}}_k]_-[i] \geq 0$ due to $\hat{\mathbf{l}}_k[i] \leq 0$ (as $i \in \mathcal{I}$). Therefore, the minimum evaluates to 0 if $\bar{\boldsymbol{\lambda}}_k \leq 0$, to $-\bar{\mathbf{b}}_k[i][\bar{\boldsymbol{\lambda}}_k]_+[i]$ otherwise. Hence, for $k \in [\![1, n-1]\!]$:

$$\min\left\{\boldsymbol{\rho}_k[i]\hat{\mathbf{l}}_k[i], \left(\boldsymbol{\rho}_k^T - \boldsymbol{\rho}_{k+1}^T W_{k+1}\right)[i]\hat{\mathbf{u}}_k[i], 0\right\} \overset{\text{using (A.9)}}{\rightarrow} -\bar{\mathbf{b}}_k[i][\bar{\boldsymbol{\lambda}}_k]_+[i].$$

Plugging all the optimisation result into Equation (A.10), we obtain:

$$\min_{\mathbf{x}_0 \in \mathcal{C}} \left(-\bar{\boldsymbol{\mu}}_1^T W_1 \mathbf{x}_0\right) - \sum_{k=1}^{n} \bar{\boldsymbol{\mu}}_k^T \mathbf{b}_k - \sum_{k=1}^{n-1} [\bar{\boldsymbol{\lambda}}_k]_+^T \bar{\mathbf{b}}_k.$$

which is exactly equation (2.5), recalling that $\underline{\mathbf{b}}_k = 0$ from (A.8) and that we had set $\bar{\boldsymbol{\mu}}_n = -1$.

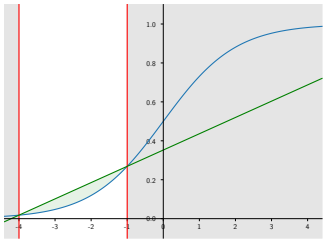
Having proved that (2.11) can be initialized with propagation-based methods, the result for dual (2.6) then trivially follows from theorem 1. In fact, equation (2.4) satisfies the conditions (namely, $\bar{\boldsymbol{\lambda}}_{n-1} = -W_n^T\mathbf{1}$ and $\bar{\boldsymbol{\lambda}}_{k-1} = W_k^T\bar{\boldsymbol{\mu}}_k$ for $k \in [\![2, n-1]\!]$) under which the two duals yield the same bounds. □

## A.3 Sigmoid Activation function

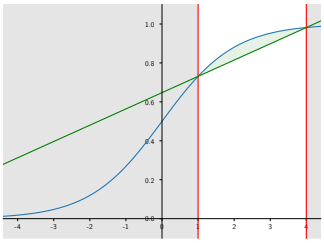
This section describes the computation highlighted in chapter 2 in the context where the activation function $\sigma(x)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{A.11}$$

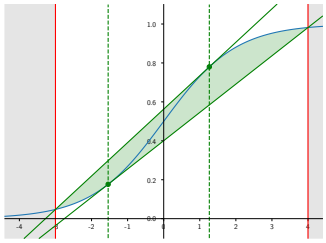
A similar methodology to the one described in this section could be used to adapt the method to work with other activation function such as hyperbolic tangent.



**(a)** The upper bound is in Case 1, while the lower bound is in the special case of Case 2 with only one piece.



**(b)** The upper bound is in the special case of Case 2 with only one piece, while the lower bound is in Case 1.



**(c)** Both upper and lower bounds are in Case 2 with the two pieces visible.

**Figure A.1:** Convex hull of the Sigmoid activation function for different input bound configurations.

We will start with a reminder about some properties of the sigmoid activation function. It takes values between 0 and 1, with $\sigma(0) = 0.5$. We can easily compute its derivatives:

$$\begin{aligned} \sigma'(x) &= \sigma(x) \times (1 - \sigma(x)) \\ \sigma''(x) &= \sigma(x) \times (1 - \sigma(x)) \times (1 - 2\sigma(x)) \end{aligned} \tag{A.12}$$

If we limit the domain of study to the negative inputs ($[-\infty, 0]$), then the function $x \mapsto \sigma(x)$ is a convex function, as can be seen by looking at the sign of the second derivative over that domain. Similarly, if we limit the domain of study to the positive inputs ($[0, \infty]$), the function is concave.

### A.3.1 Convex hull computation

In the context of ReLU, the convex hull of the activation function is given by equation (2.8), as introduced by Ehlers (2017). We will now derive it for sigmoid functions. Upper and lower bounds will be dealt in the same way, so our description will only focus on how to obtain the concave upper bound, limiting the activation function convex hull by above. The computation to derive the convex lower bound is equivalent.

Depending on the range of inputs over which the convex hull is taken, the form of the concave upper bound will change. We distinguish two cases.

**Case 1:** $\sigma'\left(\hat{\mathbf{u}}_k\right) \geq \frac{\sigma(\mathbf{u}_k)-\sigma(\mathbf{l}_k)}{\mathbf{u}_k-\mathbf{l}_k}$**.** We will prove that in this case, the upper bound will be the line passing through the points $(\hat{\mathbf{l}}_k, \sigma\left(\hat{\mathbf{l}}_k\right))$ and $(\hat{\mathbf{u}}_k, \sigma\left(\hat{\mathbf{l}}_k\right))$. The equation of it is given by:

$$\phi_{\mathbf{u}_k,\mathbf{l}_k}(\mathbf{x}) = \frac{\sigma\left(\mathbf{u}_k\right) - \sigma\left(\mathbf{l}_k\right)}{\mathbf{u}_k - \mathbf{l}_k}\left(\mathbf{x} - \hat{\mathbf{l}}_k\right) + \sigma\left(\hat{\mathbf{l}}_k\right) \tag{A.13}$$

Consider the function $d(\mathbf{x}) = \phi_{\mathbf{u}_k,\mathbf{l}_k}(\mathbf{x}) - \sigma\left(\mathbf{x}\right)$. To show that $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ is a valid upper bound, we need to prove that $\forall \mathbf{x} \in \left[\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k\right],\ d(\mathbf{x}) \geq 0$. We know that $d(\hat{\mathbf{l}}_k) = 0$ and $d(\hat{\mathbf{u}}_k) = 0$, and that $d$ is a continuous function. Its derivative is given by:

$$d'(\mathbf{x}) = \frac{\sigma\left(\mathbf{u}_k\right) - \sigma\left(\mathbf{l}_k\right)}{\mathbf{u}_k - \mathbf{l}_k} - \sigma\left(\mathbf{x}\right)\left(1 - \sigma\left(\mathbf{x}\right)\right). \tag{A.14}$$

To find the roots of $d'$, we can solve $d'(\mathbf{x}) = 0$ for the value of $\sigma\left(\mathbf{x}\right)$ and then use the logit function to recover the value of $\mathbf{x}$. In that case, this is only a second order polynomial, so it can admit at most two roots.

We know that $\lim_{\mathbf{x}\to\infty} d'(\mathbf{x}) \geq 0$, and our hypothesis tells us that $d'(\hat{\mathbf{u}}_k) \leq 0$. This means that at least one of the root lies necessarily beyond $\hat{\mathbf{u}}_k$ and therefore, the derivative of $d$ change signs at most once on the $\left[\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k\right]$ interval. If it never changes sign, it is monotonous. Given that the values taken at both extreme points are the same, $d$ being monotonous would imply that $d$ is constant, which is impossible. We therefore conclude that this means that the derivative change its sign exactly once on the interval, and is therefore unimodal. As we know that $d'(\hat{\mathbf{u}}_k) \leq 0$, this indicates that $d$ is first increasing and then decreasing. As both extreme points have a value of zero, this means that $\forall \mathbf{x} \in \left[\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k\right],\ d(x) \geq 0$.

From this result, we deduce that $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ is a valid upper bound of $\sigma$. As a linear function, it is concave by definition. Given that it constitutes the line between two points on the curve, all of its points necessarily belong to the convex hull. Therefore, it is not possible for a concave upper bound of the activation function to have lower values. This means that $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ defines the upper bounding part of the convex hull of the activation function.

**Case 2:** $\sigma'(\hat{\mathbf{u}}_k) \leq \frac{\sigma(\mathbf{u}_k)-\sigma(\mathbf{l}_k)}{\mathbf{u}_k-\mathbf{l}_k}$**.** In this case, we will have to decompose the upper bound into two parts, defined as follows:

$$\phi_{\mathbf{u}_k,\mathbf{l}_k}(\mathbf{x}) \begin{cases} \dfrac{\sigma(\mathbf{t}_k)-\sigma(\mathbf{l}_k)}{\mathbf{t}_k-\mathbf{l}_k}\left(\mathbf{x}-\hat{\mathbf{l}}_k\right)+\sigma\left(\hat{\mathbf{l}}_k\right) & \text{if } \mathbf{x} \in \left[\hat{\mathbf{l}}_k, \mathbf{t}_k\right] \quad \text{(A.15a)} \\ \sigma(\mathbf{x}) & \text{if } \mathbf{x} \in [\mathbf{t}_k, \hat{\mathbf{u}}_k], \quad \text{(A.15b)} \end{cases}$$

where $\mathbf{t}_k$ is defined as the point such that $\sigma'(\mathbf{t}_k) = \frac{\sigma(\mathbf{t}_k)-\sigma(\mathbf{l}_k)}{\mathbf{t}_k-\mathbf{l}_k}$ and $\mathbf{t}_k > 0$. The value of $\mathbf{t}_k$ can be computed by solving the equation $\sigma(\mathbf{t}_k)(1-\sigma(\mathbf{t}_k)) = \frac{\sigma(\mathbf{t}_k)-\sigma(\mathbf{l}_k)}{\mathbf{t}_k-\mathbf{l}_k}$, which can be done using the Newton-Raphson method or a binary search. Note that this needs to be done only when defining the problem, and not at every iteration of the solver. In addition, the value of $\mathbf{t}_k$ is dependant only on $\hat{\mathbf{l}}_k$ so it's possible to start by building a table of the results at the desired accuracy and cache them.

Evaluating both pieces of the function of equation(A.15) at $\mathbf{t}_k$ show that $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ is continuous. Both pieces are concave (for $\mathbf{x} \geq \mathbf{t}_k \geq 0$, $\sigma$ is concave) and they share a supergradient (the linear function of slope $\frac{\sigma(\mathbf{t}_k)-\sigma(\mathbf{l}_k)}{\mathbf{t}_k-\mathbf{l}_k}$) in $\mathbf{t}_k$, so $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ is a concave function. The proof we did for **Case 1** can be duplicated to show that the linear component is the best concave upper bound that can be achieved over the interval $\left[\hat{\mathbf{l}}_k, \mathbf{t}_k\right]$. On the interval $[\mathbf{t}_k, \hat{\mathbf{u}}_k]$, $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ is equal to the activation function, so it is also an upper bound which can't be improved upon. Therefore, $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ is the upper bounding part of the convex hull of the activation function.

Note that a special case of this happens when $\hat{\mathbf{l}}_k \geq \mathbf{t}_k$. In which case, $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ consists of only equation (A.15b).

All cases are illustrated in Figure A.1. Case 1 is shown in A.1(a), where the upper bound contains only the linear upper bound. Case 2 with both segments is visible in FigureA.1(c), with the cutoff points $tb_k$ highlighted by a green dot, and the special case with $\hat{\mathbf{l}}_k \geq \mathbf{t}_k$ is demonstrated in Figure A.1(b).

### A.3.2 Solving the $\mathcal{P}_k$ subproblems over sigmoid activation

As a reminder, the problem that needs to be solved is the following (where $\mathbf{c}_{k+1} = -\boldsymbol{\rho}_{k+1},\ \mathbf{c}_k = \boldsymbol{\rho}_k$):

$$\begin{aligned}[\hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}] = \underset{\hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}}{\operatorname{argmin}} \quad & \mathbf{c}_k^T \hat{\mathbf{x}}_{B,k} + \mathbf{c}_{k+1}^T \hat{\mathbf{x}}_{A,k+1} \\ \text{s.t} \quad & \hat{\mathbf{l}}_k \leq \hat{\mathbf{x}}_{B,k} \leq \hat{\mathbf{u}}_k \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}) \in \operatorname{Conv}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) \\ & \hat{\mathbf{x}}_{A,k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1},\end{aligned} \tag{A.16}$$

where `cvx_hull` is defined either by equations (A.13) or (A.15). In this case, we will still be able to compute a closed form solution.

We will denote $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ and $\psi_{\mathbf{u}_k,\mathbf{l}_k}$ the upper and lower bound functions defining the convex hull of the sigmoid function, which can be obtained as described in the previous subsection. If we use the last equality constraint of Problem (A.16) to replace the $\hat{\mathbf{x}}_{A,k+1}$ term in the objective function, we obtain $\mathbf{c}_k^T \hat{\mathbf{x}}_{B,k} + \mathbf{c}_{k+1}^T W_{k+1}\mathbf{x}_k$. Depending on the sign of $\mathbf{c}_{k+1}^T W_{k+1}$, $\mathbf{x}_k$ will either take the value $\phi_{\mathbf{u}_k,\mathbf{l}_k}$ or $\psi_{\mathbf{u}_k,\mathbf{l}_k}$, resulting in the following problem:

$$\begin{aligned}\min_{\hat{\mathbf{x}}_{B,k}} \quad & \mathbf{c}_k^T \hat{\mathbf{x}}_{B,k} + \left[\mathbf{c}_{k+1}^T W_{k+1}\right]_- \phi_{\mathbf{u}_k,\mathbf{l}_k}(\hat{\mathbf{x}}_{B,k}) + \left[\mathbf{c}_{k+1}^T W_{k+1}\right]_+ \psi_{\mathbf{u}_k,\mathbf{l}_k}(\hat{\mathbf{x}}_{B,k}) \\ \text{s.t} \quad & \hat{\mathbf{l}}_k \leq \hat{\mathbf{x}}_{B,k} \leq \hat{\mathbf{u}}_k.\end{aligned} \tag{A.17}$$

To solve this problem, several observations can be made: First of all, at that point, the optimisation decomposes completely over the components of $\hat{\mathbf{x}}_{B,k}$ so all problems can be solved independently from each other. The second observation is that to solve this problem, we can decompose the minimisation over the whole range $\left[\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k\right]$ into a set of minimisation over the separate pieces, and then returning the minimum corresponding to the piece producing the lowest value.

The minimisation over the pieces can be of two forms. Both bounds can be linear (such as between the green dotted lines in Figure A.1(c)), in which case the problem is easy to solve by looking at the signs of the coefficient of the objective function. The other option is that one of the bound will be equal to the activation function

(such as in Figures A.1(a) or A.1(b), or in the outer sections of Figure A.1(c)), leaving us with a problem of the form:

$$\begin{aligned} \min_{\hat{\mathbf{x}}_{B,k}} \quad & \mathbf{c}_{\text{lin}}^T \hat{\mathbf{x}}_{B,k} + \mathbf{c}_\sigma^T \sigma\left(\hat{\mathbf{x}}_{B,k}\right) \\ & \hat{\mathbf{l}} \leq \hat{\mathbf{x}}_{B,k} \leq \hat{\mathbf{u}}, \end{aligned} \tag{A.18}$$

where $\hat{\mathbf{l}}$, $\hat{\mathbf{u}}$, $\mathbf{c}_{\text{lin}}$ and $\mathbf{c}_\sigma$ will depend on what part of the problem we are trying to solve.

This is a convex problem so the value will be reached either at the extremum of the considered domain ($\hat{\mathbf{l}}$ or $\hat{\mathbf{u}}$), or it will be reached at the points where the derivative of the objective functions cancels. This corresponds to the roots of $\mathbf{c}_{\text{lin}} + \mathbf{c}_\sigma^T \sigma\left(\hat{\mathbf{x}}_{B,k}\right)\left(1 - \sigma\left(\hat{\mathbf{x}}_{B,k}\right)\right)$. Provided that $1 + \frac{4\mathbf{c}_{\text{lin}}}{\mathbf{c}_\sigma} \geq 0$, the possible roots will be given by $\sigma^{-1}\left(\frac{1 \pm \sqrt{1 + \frac{4\mathbf{c}_{\text{lin}}}{\mathbf{c}_\sigma}}}{2}\right)$, with $\sigma^{-1}$ being the logit function, the inverse of the sigmoid function $\left(\sigma^{-1}(\mathbf{x}) = \log\left(\frac{\mathbf{x}}{1-\mathbf{x}}\right)\right)$. To solve problem (A.18), we evaluate its objective function at the extreme points ($\hat{\mathbf{l}}$ and $\hat{\mathbf{u}}$) and at those roots if they are in the feasible domain, and return the point corresponding to the minimum score achieved. With this method, we can solve the $\mathcal{P}_k$ subproblems even when the activation function is a sigmoid.

## A.4 Momentum for the Proximal solver

Supergradient methods for both our dual (2.11) and for (2.6) by Dvijotham et al. (2018b) can easily rely on acceleration techniques such as Adam (Kingma and Ba, 2015) to speed-up convergence. Therefore, inspired by its presence in Adam, and by the work on accelerating proximal methods (Lin et al., 2017; Salzo and Villa, 2012), we apply momentum on the proximal updates.

By closely looking at equation (2.18), we can draw a similarity between dual updates in the proximal algorithm and supergradient ascent. The difference is that the former operation takes place after a closed-form minimization of the linear inner problem in $\hat{\mathbf{x}}_A$ and $\hat{\mathbf{x}}_B$, whereas the latter after some steps of an optimization algorithm that solves the quadratic form of the Augmented Lagrangian (2.19). Let us denote the (approximate) argmin of the Augmented Lagrangian at the $t$-th iteration of the method of multipliers by $\hat{\mathbf{x}}^{t,\dagger}$. Thanks to the aforementioned

similarity, we can keep an exponential average of the gradient-like terms with parameter $\mu \in [0, 1]$ and adopt momentum for the dual updates, yielding:

$$\begin{aligned} \boldsymbol{\pi}_k^{t+1} &= \mu \boldsymbol{\pi}_k^t + \frac{\hat{\mathbf{x}}_{B,k}^{t,\dagger} - \hat{\mathbf{x}}_{A,k}^{t,\dagger}}{\eta_k} \\ \boldsymbol{\rho}_k^{t+1} &= \boldsymbol{\rho}_k^t + \boldsymbol{\pi}_k^{t+1} \end{aligned} \tag{A.19}$$

Under the proximal interpretation of the method of multipliers, the Augmented Lagrangian can be derived by adding a (quadratic) proximal term on $\boldsymbol{\rho}$ in the standard Lagrangian, and plugging the closed-form solution of the resulting quadratic problem, which corresponds to equation (2.18), into the objective (Bertsekas and Tsitsiklis, 1989). In other words, we make the following modification to the standard Lagrangian:

$$\hat{x}_{A,n} + \sum_{k=1}^{n-1} \left(\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}\right)^T \boldsymbol{\rho}_k^{t+1} \quad \rightarrow \quad \hat{x}_{A,n} + \sum_{k=1}^{n-1} \left(\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}\right)^T \boldsymbol{\rho}_k^{t+1} - \frac{\eta_k}{2} \|\boldsymbol{\rho}_k^{t+1} - \boldsymbol{\rho}_k^t\|^2,$$

and then plug in equation (2.18), which results from setting the dual gradient of the quadratic function above to 0. If, rather than equation (2.18), we plug in equation (A.19), incorporating momentum, the primal problem of the modified method of multipliers becomes:

$$\begin{aligned} &\left[\mathbf{x}^t, \hat{\mathbf{x}}^t\right] = \underset{\mathbf{x}, \hat{\mathbf{x}}}{\operatorname{argmin}} \mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}^t\right) \text{ s.t. } \mathcal{P}_0(\mathbf{x}_0, \hat{\mathbf{x}}_{A,1}); \ \mathcal{P}_k(\mathbf{x}_k, \hat{\mathbf{x}}_{B,k}, \hat{\mathbf{x}}_{A,k+1}) \quad k \in [\![1, n-1]\!] \\ &\text{where: } \mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}^t\right) = \begin{bmatrix} \hat{x}_{A,n} + \sum_{k=1}^{n-1} \left(\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}\right)^T \boldsymbol{\rho}_k^t + \sum_{k=1}^{n-1} \frac{1}{2\eta_k} \|\hat{\mathbf{x}}_{B,k} - \hat{\mathbf{x}}_{A,k}\|^2 \\ - \sum_{k=1}^{n-1} \quad \frac{\eta_k}{2} \|\mu \boldsymbol{\pi}_k\|^2. \end{bmatrix} \end{aligned} \tag{A.20}$$

We point out that the Augmented Lagrangian's gradients $\nabla_{\hat{\mathbf{x}}_{B,k}} \mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}\right)$ and $\nabla_{\hat{\mathbf{x}}_{A,k}} \mathcal{L}\left(\hat{\mathbf{x}}, \boldsymbol{\rho}\right)$ remain unvaried with respect to the momentum-less solver. Therefore, in practice, the only algorithmic change for the solver lies in the dual update formula (A.19).

## A.5 Supplementary Incomplete Verification Experiments

We now complement the incomplete verification results presented in the chapter 2. Figures A.2 and A.3 show experiments (see section 2.7) for a network trained using standard stochastic gradient descent and cross entropy, with no robustness objective. We employ $\epsilon_{\text{verif}} = 1.125/255$, which is smaller than commonly employed verification radii due to the network's adversary-agnostic training. Most observations done in chapter 2 for the adversarially trained network (Figures 2.2 and 2.3) still hold true: in particular, the advantage of the Lagrangian Decomposition based dual compared to the dual by Dvijotham et al. (2018b) is even more marked than in Figure 2.3. In fact, for a subset of the properties, DSG+ returns rather loose bounds even after 1040 iterations. The Proximal returns better bounds than the Supergradient in this case as well, with Proximal displaying a much larger support

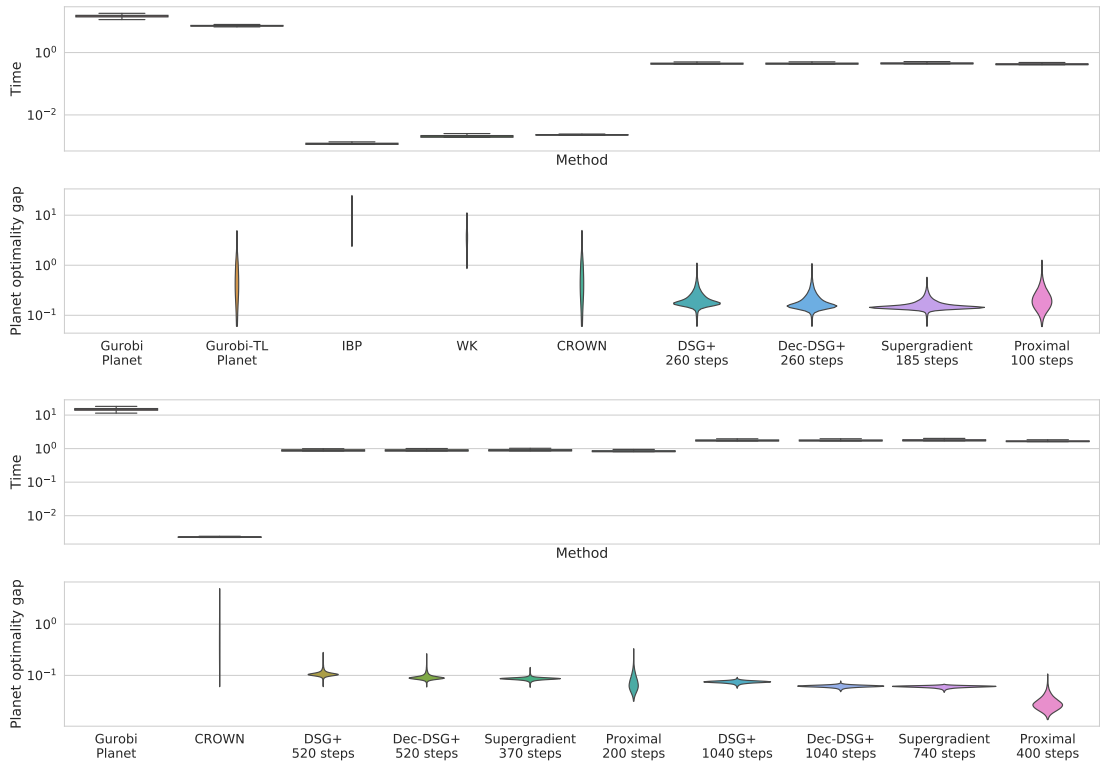



**Figure A.2:** Comparison of the distribution of runtime and gap to optimality on an SGD-trained network. In both cases, lower is better. The width at a given value represents the proportion of problems for which this is the result. Gurobi Planet always returns the optimal solution to problem (2.7), at a large computational cost.

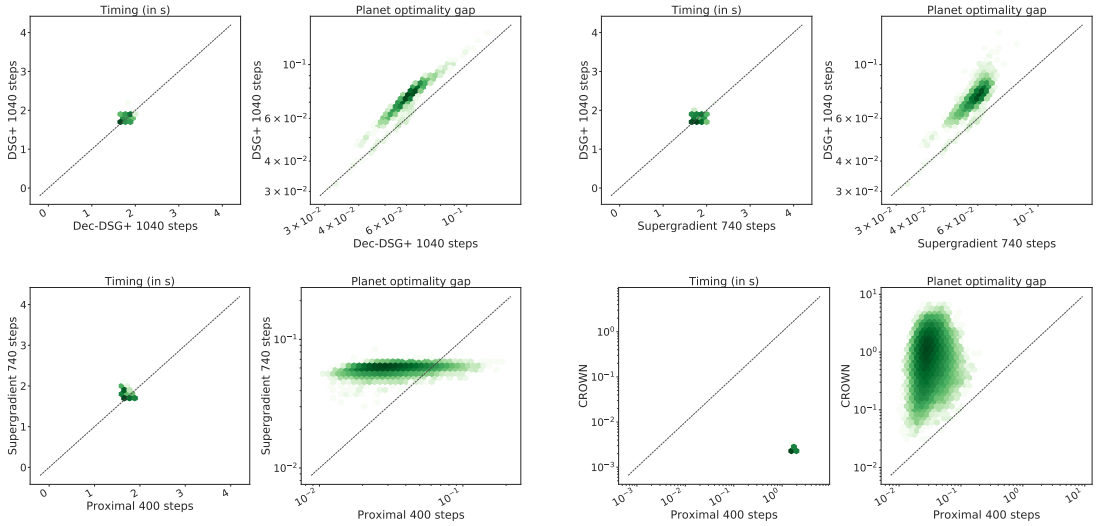



**Figure A.3:** Pointwise comparison for a subset of the methods on the data presented in Figure A.2. Each datapoint corresponds to a CIFAR image, darker colour shades mean higher point density in a logarithmic scale. The dotted line corresponds to the equality and in both graphs, lower is better along both axes.

of the optimality gap distribution.

# B

# Scaling the Convex Barrier with Sparse Dual Algorithms

## Contents


B.1 Limitations of Previous Dual Approaches . . . . . **162**
B.2 Dual Initialisation . . . . . **163**
B.2.1 Equivalence to Planet . . . . . 163
B.2.2 Big-M Dual . . . . . 164
B.2.3 Big-M solver . . . . . 165
B.3 Implementation Details for Saddle Point . . . . . **166**
B.3.1 Constraint Price Caps . . . . . 166
B.3.2 Primal Initialization . . . . . 167
B.4 Dual Derivations . . . . . **168**
B.5 Intermediate Bounds . . . . . **169**
B.6 Pre-activation Bounds in $\mathcal{A}_k$ . . . . . **170**
B.6.1 Motivating Example . . . . . 171
B.6.2 Derivation of $\mathcal{A}_k$ . . . . . 174
B.7 Masked Forward and Backward Passes . . . . . **177**
B.7.1 Convolution as matrix-matrix multiplication . . . . . 178
B.7.2 Masked convolution as matrix-matrix multiplication . . . 178
B.8 Experimental Appendix . . . . . **179**
B.8.1 Adversarially-Trained CIFAR-10 Incomplete Verification . 179
B.8.2 Sensitivity of Active Set to selection criterion and frequency 180
B.8.3 MNIST Incomplete Verification . . . . . 183

## B.1 Limitations of Previous Dual Approaches

In this section, we show that previous dual derivations (Bunel et al., 2020a; Dvijotham et al., 2018b) violate Fact 1. Therefore, they are not efficiently applicable to problem (3.3), motivating our own derivation in section 3.3.

We start from the approach by Dvijotham et al. (2018b), which relies on relaxing equality constraints (3.1b), (3.1c) from the original non-convex problem (3.1). Dvijotham et al. (2018b) prove that this relaxation corresponds to solving convex problem (3.2), which is equivalent to the Planet relaxation (Ehlers, 2017), to which the original proof refers. As we would like to solve tighter problem (3.3), the derivation is not directly applicable. Relying on intuition from convex analysis applied to duality gaps (Lemaréchal, 2001), we conjecture that relaxing the composition (3.1c) $\circ$ (3.1b) might tighten the primal problem equivalent to the relaxation, obtaining the following dual:

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

Unfortunately dual (B.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. (2018b), which had an efficient closed-form for the inner problems.

The derivation by Bunel et al. (2020a), instead, operates by substituting (3.1c) with its convex hull and solving its Lagrangian Decomposition dual. The Decomposition dual for the convex hull of (3.1c) $\circ$ (3.1b) (i.e., $\mathcal{A}_k$) takes the following form:

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

where $\mathcal{A}_{\text{dec},k}$ corresponds to $\mathcal{A}_k$ with the following substitutions: $\mathbf{x}_k \rightarrow \mathbf{x}_{B,k}$, and $\hat{\mathbf{x}}_k \rightarrow W_n \mathbf{x}_{A,n-1} + \mathbf{b}_n$. 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.

## B.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 $\boldsymbol{\alpha}^0, \beta^0_{\mathcal{M}}$ using interval bound propagation (Gowal et al., 2018a)
  **for** $t \in [\![1, T-1]\!]$ **do**
    $\mathbf{x}^{*,t}, \mathbf{z}^{*,t} \in \operatorname{argmin}_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^t, \beta^t_{\mathcal{M}})$ using (B.5)-(B.6)
    $\boldsymbol{\alpha}^{t+1}, \beta^{t+1}_{\mathcal{M}} \leftarrow [\boldsymbol{\alpha}^t, \beta^t] + H[\nabla_{\boldsymbol{\alpha}} d_{\mathcal{M}}(\boldsymbol{\alpha}, \beta), \nabla_{\beta_{\mathcal{M}}} d_{\mathcal{M}}(\boldsymbol{\alpha}, \beta)]$ ▷ supergradient step, using (B.7)
    $\boldsymbol{\alpha}^{t+1}, \beta^{t+1}_{\mathcal{M}} \leftarrow \max(\boldsymbol{\alpha}^{t+1}, 0), \max(\beta^{t+1}_{\mathcal{M}}, 0)$ ▷ dual projection
  **return** $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \beta^T_{\mathcal{M}})$

As shown in section 3.3, the Active Set solver reduces to a dual solver for the Big-M relaxation (3.2) if the active set $\mathcal{B}$ is kept empty throughout execution. We employ this Big-M solver as dual initialization for both Active Set (§3.4) and Saddle Point (§3.5). We demonstrate experimentally in §3.8 that, when used as a stand-alone solver, our Big-M solver is competitive with previous dual algorithms for problem (3.2).

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 3.4, $\boldsymbol{\beta}_{\mathcal{M}} := \boldsymbol{\beta}_{\emptyset}$. We now describe the equivalence between the Big-M and Planet relaxations, before presenting the solver in section B.2.3 and the dual it operates on in section B.2.2.

### B.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 (3.2) is equivalent to the Planet relaxation by Ehlers (2017). Then, due to strong duality, our Big-M solver (section B.2.2) and the solvers by Bunel et al. (2020a); Dvijotham et al. (2018b) will all converge to the bounds from the solution of problem (3.2). In fact, the Decomposition-based

method (Bunel et al., 2020a) directly operates on the Planet relaxation, while Dvijotham et al. (2018b) prove that their dual is equivalent to doing so.

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

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

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 (3.2), 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 B.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.

### B.2.2 Big-M Dual

As evident from problem (3.3), $\mathcal{A}_k \subseteq \mathcal{M}_k$. 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 (3.5). The Big-M dual is a specific instance of the Active

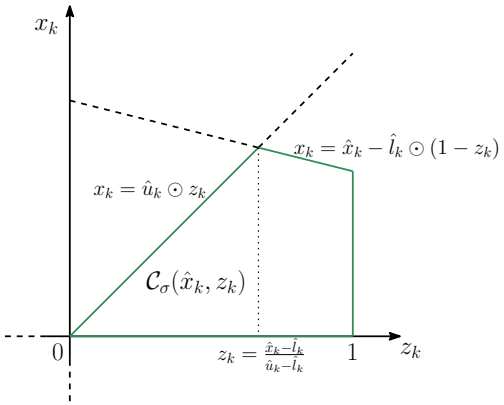



**Figure B.1:** $\mathcal{M}_k$ plotted on the $(\mathbf{z}_k, \mathbf{x}_k)$ plane, when $\hat{\mathbf{l}}_k \leq 0$ and $\hat{\mathbf{u}}_k \geq 0$.

Set dual (3.7) where $\mathcal{B} = \emptyset$, and it takes the following form:

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

### B.2.3 Big-M solver

We initialise dual variables from interval propagation bounds (Gowal et al., 2018a): 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 3.4 on a generic active set $\mathcal{B}$. All the computations in the solver follow from keeping $\mathcal{B} = \emptyset$ in §3.4. We explicitly report them here for the reader's convenience.

Let us define the following shorthand for the primal coefficients:

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

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:

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

For $k=0$, instead (assuming, as §3.4 that this can be done efficiently):

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

The supergradient over the Big-M dual variables $\boldsymbol{\alpha},\boldsymbol{\beta}_{k,0},\boldsymbol{\beta}_{k,1}$ is computed exactly as in §3.4 and is again a subset of the supergradient of the full dual problem (3.5). We report it for completeness. For each $k\in[\![0,n-1]\!]$:

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

## B.3 Implementation Details for Saddle Point

In this section, we present details of the Saddle Point solver that were omitted from chapter 3. We start with the choice of price caps $\boldsymbol{\mu}_k$ on the Lagrangian multipliers (§B.3.1), and conclude with a description of our primal initialisation procedure (§B.3.2).

### B.3.1 Constraint Price Caps

Price caps $\boldsymbol{\mu}$ containing the optimal dual solution to problem (3.5) can be found via binary search (see §3.5.1). However, such a strategy might require running Saddle Point to convergence on several instances of problem (3.13). Therefore, in practice, we employ a heuristic to set constraint caps to a reasonable approximation.

Given duals $(\boldsymbol{\alpha}^0, \boldsymbol{\beta}^0)$ from the dual initialization procedure (algorithm 3 or algorithm 5, depending on the computational budget), for each $k \in [\![1, n-1]\!]$ we set $\boldsymbol{\mu}_k$ as follows:

$$\begin{aligned} \boldsymbol{\mu}_{\alpha,k} &= \begin{cases} \boldsymbol{\alpha}_k^0 & \text{if } \boldsymbol{\alpha}_k^0 > 0 \\ c_{\alpha,k} & \text{otherwise} \end{cases} \\ \boldsymbol{\mu}_{\beta,k} &= \begin{cases} \sum_{I_k \in 2^{W_k}} \boldsymbol{\beta}_k^0 & \text{if } \sum_{I_k \in 2^{W_k}} \boldsymbol{\beta}_k^0 > 0 \\ c_{\beta,k} & \text{otherwise,} \end{cases} \end{aligned} \tag{B.8}$$

where $c_{\alpha,k}$ and $c_{\beta,k}$ are small positive constants. In other words, we cap the (sums over) dual variables to their values at initialization if these are non-zero, and we allow dual variables to turn positive otherwise. While a larger feasible region (for instance, setting $\boldsymbol{\mu}_{\alpha,k} = \max_i \boldsymbol{\alpha}_k[i] \odot \mathbf{1}$, $\boldsymbol{\mu}_{\beta,k} = \max_i \sum_{I_k \in 2^{W_k}} \boldsymbol{\beta}_k^0[i] \odot \mathbf{1}$) might yield tighter bounds at convergence, we found assignment (B.8) to be more effective on the iteration budgets employed in §3.8.2,§3.8.3.

### B.3.2 Primal Initialization

If we invert the minimization and maximization, our saddle-point problem (3.13) can be seen as a non-smooth minimization problem (the minimax theorem holds):

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

$$\begin{array}{lll} \text{s.t.} & \mathbf{x}_0 \in \mathcal{C}, & \\ & (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] & k \in [\![1, n-1]\!]\,, \\ & \boldsymbol{\alpha}_k \in [\mathbf{0}, \boldsymbol{\mu}_{k,\alpha}] & \\ & \left(\boldsymbol{\beta}_k \geq \mathbf{0},\ \sum_{I_k \in \mathcal{E}_k \cup \{0,1\}} \boldsymbol{\beta}_{k,I_k} \leq \boldsymbol{\mu}_{k,\beta}\right) & k \in [\![1, n-1]\!]\,. \end{array} \tag{B.9}$$

The "primal view"[1] of problem (B.9) admits an efficient subgradient method, which we employ as primal initialization for the Saddle Point solver.

**Technical details** The inner maximizers $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, which are required to obtain a valid subgradient, will be given in closed-form by the dual conditional gradients from equations (3.14), (3.15) (the objective is bilinear). Then, using the dual functions defined in (3.6), the subgradient over the linearly-many primal variables can be computed as $\boldsymbol{f}_k(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ for $\mathbf{x}_k$ and $\boldsymbol{g}_k(\boldsymbol{\beta}^*)$ for $\mathbf{z}_k$. After each subgradient step, the primals are projected to the feasible space: this is trivial for $\mathbf{x}_k$ and $\mathbf{z}_k$, which are box constrained, and can be efficiently performed for $\mathcal{C}$ in the common cases of $\ell_\infty$ or $\ell_2$ balls.

**Simplified initialization** Due to the additional cost associated to masked forward-backward passes (see appendix B.7), the primal initialization procedure can be simplified by restricting the dual variables to the ones associated to the Big-M relaxation (appendix B.2). This can be done by substituting $\mathcal{E}_k \leftarrow \emptyset$ and $\boldsymbol{\beta}_k \leftarrow \boldsymbol{\beta}_{\emptyset,k}$

[1] due to the restricted dual domain, problem (B.9) does not correspond to the original primal problem in (3.3).

(see notation in §3.4.1) in problem (B.9). A subgradient method for the resulting problem can then be easily adapted from the description above.

## B.4 Dual Derivations

We now prove that the dual of:

$$\begin{aligned}
&\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \ \hat{x}_n \ \text{ s.t.} \\
&\quad \mathbf{x}_0 \in \mathcal{C} \\
&\quad \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} \qquad\qquad k \in [\![0, n-1]\!]\,, \\
&\quad (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in \mathcal{M}_k \\
&\quad \mathbf{x}_k \leq \left.\begin{pmatrix} (W_k \odot I_k)\,\mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k \ + \\ -\left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1-\mathbf{z}_k) \ + \\ +\left(W_k \odot (1-I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \end{pmatrix} \ \forall I_k \in \mathcal{E}_k \right\} := \mathcal{A}_k \quad k \in [\![1, n-1]\!]\,.
\end{aligned} \tag{3.3}$$

corresponds to as:

$$\begin{aligned}
&\max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\geq 0} d(\boldsymbol{\alpha},\boldsymbol{\beta}) \qquad \text{where:} \qquad d(\boldsymbol{\alpha},\boldsymbol{\beta}) := \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}), \\
&\qquad \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}) = \left[\begin{array}{l} \sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k - \sum_{k=0}^{n-1} \boldsymbol{f}_k(\boldsymbol{\alpha},\boldsymbol{\beta})^T \mathbf{x}_k - \sum_{k=1}^{n-1} \boldsymbol{g}_k(\boldsymbol{\beta})^T \mathbf{z}_k \\ +\sum_{k=1}^{n-1} \left(\sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \square \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k)\right) \end{array}\right], \\
&\text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad k \in [\![1, n-1]\!],
\end{aligned} \tag{3.5}$$

with:

$$\begin{aligned}
\boldsymbol{f}_k(\boldsymbol{\alpha},\boldsymbol{\beta}) &= \ \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \textstyle\sum_{I_k} \boldsymbol{\beta}_{k,I_k} + \sum_{I_{k+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}}, \\
\boldsymbol{g}_k(\boldsymbol{\beta}) &= \left[\begin{array}{l} \sum_{I_k \in \mathcal{E}_k} \left(W_k \odot (1-I_k) \odot \check{U}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k \\ +\sum_{I_k \in \mathcal{E}_k} \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} + \sum_{I_k \in \mathcal{E}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k. \end{array}\right.
\end{aligned} \tag{3.6}$$

The Active Set (equation (3.7)) and Big-M duals (equation (B.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 $\hat{\mathbf{x}}_k = W_k \mathbf{x}_{k-1} + \mathbf{b}_k$):

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

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:

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

which corresponds to the form shown in problem (3.5).

## B.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 (3.1) over subsets of the

network (Bunel et al., 2020a). For $\hat{\mathbf{l}}_i$, this means solving the following problem (separately, for each entry $\hat{\mathbf{l}}_i[j]$):

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

As (B.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 (3.3), 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., 2018a).

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

## B.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 §B.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 (§B.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:

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

While pre-activation bounds regularly appear as lower and upper bounds to $\hat{\mathbf{x}}_k$, they do not appear in any other constraint. Indeed, the difference with respect to

$\mathcal{A}_k$ as defined in equation (3.3) exclusively lies in the treatment of pre-activation bounds within the exponential family. Set $\mathcal{A}_k$ explicitly employs generic $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ in the constraint set via $\mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k$, and $\mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k)$ from $\mathcal{M}_k$. On the other hand, $\mathcal{A}'_k$ implicitly sets $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ to the value dictated by interval propagation bounds (Gowal et al., 2018a) 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:

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

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 B.10), the optimisation for $M_k^-, M_k^+$ involves only the layer before the current. Therefore, the constraints in (B.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.

### B.6.1 Motivating Example

Figure B.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 (3.1))

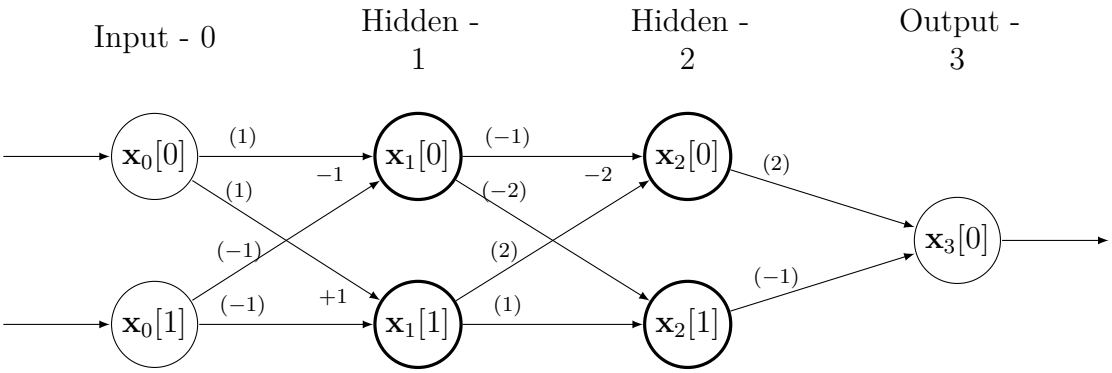



**Figure B.2:** Example network architecture in which $\mathcal{M}_k \subset \mathcal{A}'_k$, with pre-activation bounds computed with $\mathcal{C}_k = \mathcal{M}_k$. For the bold nodes (the two hidden layers) a ReLU activation follows the linear function. The numbers between parentheses indicate multiplicative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

optimization problem for the lower bound on the output node $\mathbf{x}_3$.

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

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

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

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

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

Let us compute pre-activation bounds with $\mathcal{C}_k = \mathcal{M}_k$ (see problem (B.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:

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

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

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

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

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

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

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:

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

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

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

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 (B.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 (B.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 (B.15) for $I_2[1,\cdot] = [1\ 1]$ and $I_2[1,\cdot] = [0\ 0]$, yielding a weaker lower bound for the network output.

### B.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 (B.11)) to obtain our formulation $\mathcal{A}_k$ (as defined in equation (3.3)). 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., 2019), which is defined as follows:

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

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 (B.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. (2019) 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 B.5) are available, we can naturally add them to (B.17) as follows:

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

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

**Proposition 9.** *Sets $\mathcal{S}_k$ from equation* (B.18) *and $\mathcal{A}_k$ from problem* (3.3) *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., 2019). Going along the lines of the original proof, we start from (B.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. (2019), 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:

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

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

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

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

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

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

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

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

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

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

where only inequalities (B.19g) to (B.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. (2019) for the others. The combination of these new inequalities yields trivial constraints. For instance:

$$\text{(B.19i)} + \text{(B.19g)} \implies \hat{\mathbf{l}}_k[i] \leq \boldsymbol{w}_{i,k}^T \mathbf{x}_{k-1} + \boldsymbol{b}_k[i] = \hat{\mathbf{x}}_k[i] \tag{B.20}$$

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 (B.19a) + (B.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 (B.19a). In particular, combining inequalities (B.19g) to (B.19j) with inequalities (B.19d) to (B.19f) generates constraints that are (after algebraic manipulations) superfluous with respect to those in (B.21). We are now ready to show the system resulting from the elimination:

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

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

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

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

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

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

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

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

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

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

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

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

By noting that no auxiliary variable is left in (B.21i) and in (B.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., 2019), leading to $\mathcal{A}_{k,i}$. Repeating the proof for each neuron $i$ at layer $k$, we get $\mathcal{A}_k = \text{Proj}_{\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{z}_k}(\mathcal{S}_k)$.

□

## B.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 B.2), both the Active Set (§3.4) and Saddle Point (§3.5) solvers require a specialized operator that we call “masked" forward/backward pass. We now provide the details to our implementation.

As a reminder from §3.6, masked forward and backward passes respectively take the following forms (writing convolutional operators via their fully connected equivalents):

$$(W_k \odot I_k)\, \mathbf{a}_k, \quad (W_k \odot I_k)^T\, \mathbf{a}_{k+1},$$

where $\mathbf{a}_k \in \mathbb{R}^{n_k}, \mathbf{a}_{k+1} \in \mathbb{R}^{n_{k+1}}$. They are needed when dealing with the exponential family of constraints from the relaxation by Anderson et al. (2020). Masked operators are straightforward to implement for fully connected layers (via element-wise products). We instead need to be more careful when handling convolutional layers. Standard convolution relies on re-applying the same weights (kernel) to many

different parts of the image. A masked pass, instead, implies that the convolutional kernel is dynamically changing while it is being slided through 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. Our implementation relies on an alternative view of convolutional layers, which we outline next.

### B.7.1 Convolution as matrix-matrix multiplication

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:

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

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:

$$W_R = [w]_{\mathbb{R}^{c \times k_1 \times k_2}, \mathbb{R}^{c \times k_1 k_2}}, \quad \mathbf{y}_R = [\mathbf{y}]_{\mathbb{R}^{c \times o_2 \times o_3}, \mathbb{R}^{c \times o_2 o_3}}$$

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:

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

### B.7.2 Masked convolution as matrix-matrix multiplication

We need to mask the convolution with a different scalar for each input-output pair. Therefore, we employ a mask $I \in \mathbb{R}^{c \times k_1 k_2 \times o_2 o_3}$, whose additional dimension with respect to $W_R$ is associated to the output space of the convolution. Assuming

vectors are broadcast to the correct output shape[2] , we can write the masked forward and backward passes by adapting equation (B.23) as follows:

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

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 (§3.8.2) it ranges from 16 to 64 depending on the layer.

## B.8 Experimental Appendix

We conclude the appendix by presenting supplementary experiments with respect to the presentation in chapter 3.

### B.8.1 Adversarially-Trained CIFAR-10 Incomplete Verification

In addition to the SGD-trained network in §3.8.2, 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 $\epsilon_{train} = 2/255$. Each adversarial sample for the training was obtained using 50 steps of projected gradient descent. For this network, we upper bound the vulnerability to perturbations with $\epsilon_{ver} = 2.7/255$. Hyper-parameters are kept to the values tuned on the SGD-trained network from section 3.8.2.

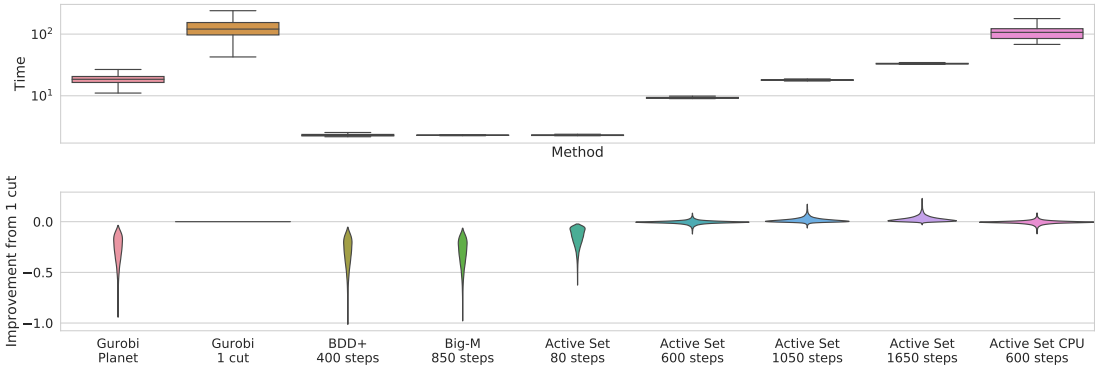
Figures B.3, B.4, B.5 confirm most of the observations carried out for the SGD-trained network in §3.8.2, with fewer variability around the bounds returned by Gurobi cut. Big-M is competitive with BDD+, and switching to Active Set

[2]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.

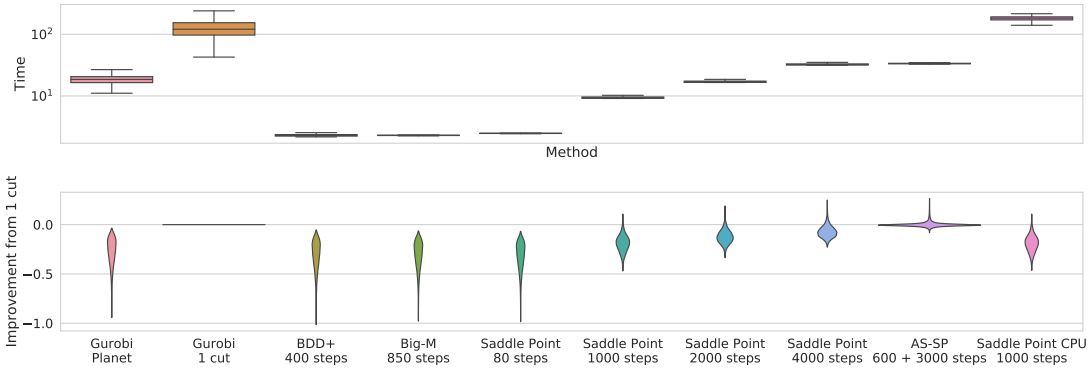
after 500 iterations results in much better bounds in the same time. Increasing the computational budget for Active Set still results in better bounds than Gurobi cut in a fraction of its running time, even though the performance gap is on average smaller than on the SGD-trained network. As in section 3.8.2 the gap between Saddle Point and Active Set, though larger here on average, decreases with the computational budget, and is further reduced when initializing with a few Active Set iterations.

### B.8.2 Sensitivity of Active Set to selection criterion and frequency

In section 3.4.2, 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 (3.4) invoked at the primal

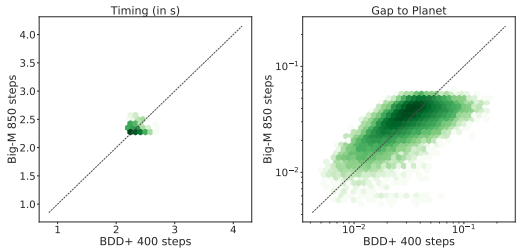



**(a)** Speed-accuracy trade-offs of Active Set for different iteration ranges and devices.

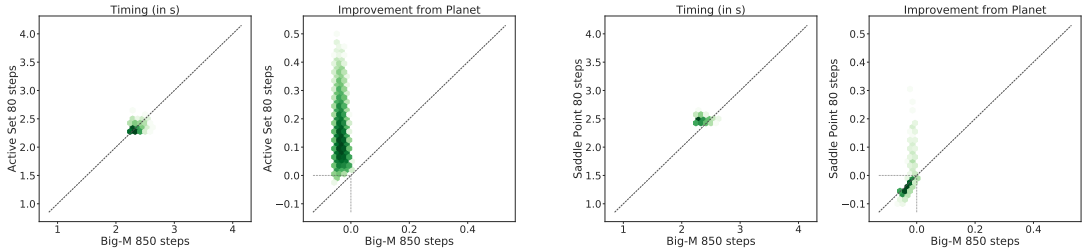



**(b)** Speed-accuracy trade-offs of Saddle Point for different iteration ranges and devices.

**Figure B.3:** Upper bounds to the adversarial vulnerability for the network adversarially trained with the method by Madry et al. (2018), from Bunel et al. (2020a). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better, 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 (right). For the latter, lower is better.




**(b)** Comparison of runtime (Timing) and difference with the Gurobi Planet bounds (Improvement from Planet). For the latter, higher is better.

**Figure B.4:** Pointwise comparison for a subset of the methods on the data presented in figure 3.1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

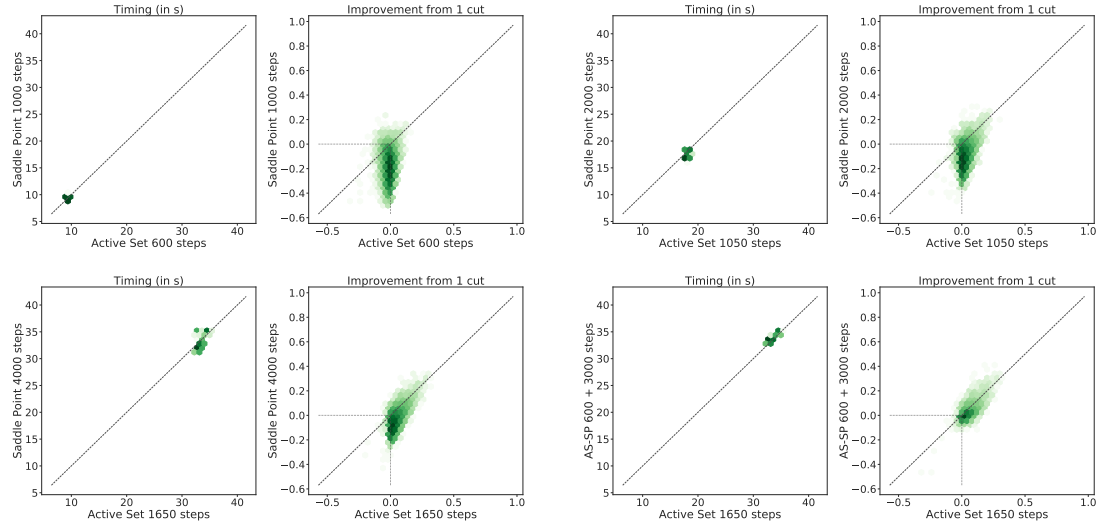



**Figure B.5:** Pointwise comparison between our proposed solvers on the data presented in figure B.3. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

minimiser of $\mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$, at a fixed frequency $\omega$. 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 B.6 shows that a good selection criterion is key to the

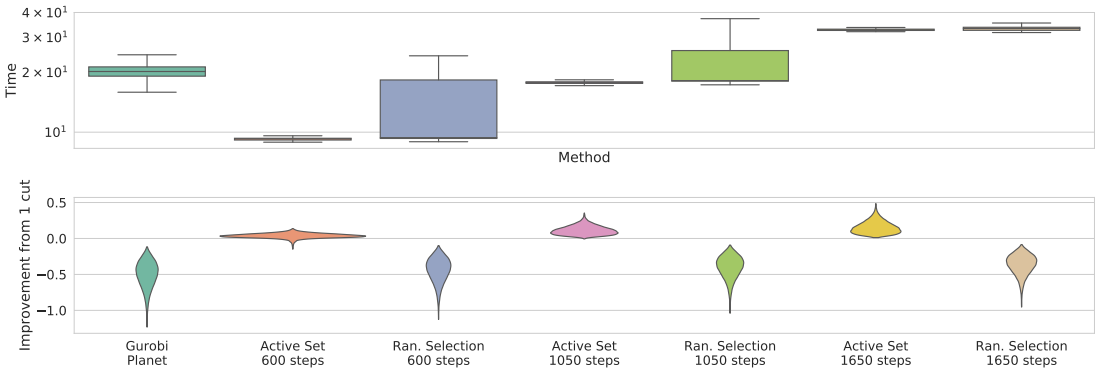



**Figure B.6:** Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the SGD-trained network from Bunel et al. (2020a). Sensitivity of Active Set to selection criterion (see §3.4.2).

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 §3.4.2. In addition, we investigate the sensitivity of Active Set (AS) to variable addition frequency $\omega$. In order to do so, we cap the maximum number of cuts at 7 for all runs, and vary $\omega$ while keeping the time budget fixed (we test on three different time budgets). Figure B.7 compares the results for $\omega = 450$ (Active Set), which were presented in §3.8.2, 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 $\omega$ across all the three budgets, with the difference in obtained bounds decreasing with the number of iterations. Moreover,

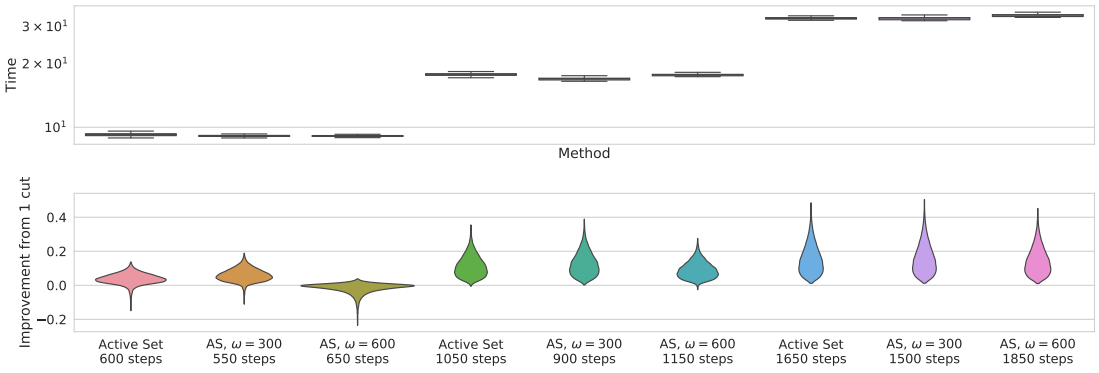
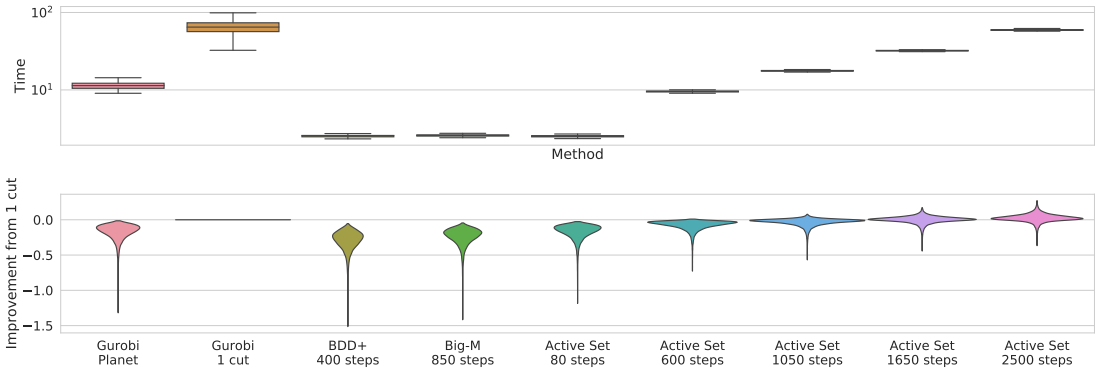
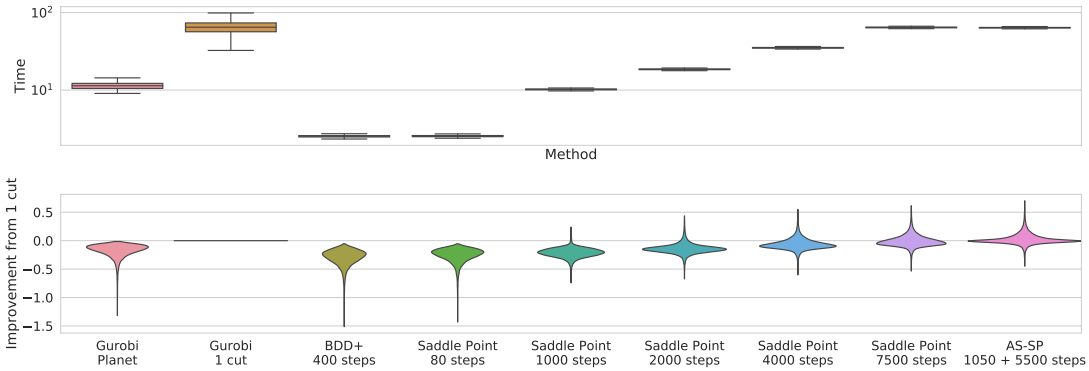



**Figure B.7:** Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the SGD-trained network from Bunel et al. (2020a). Sensitivity of Active Set to variable addition frequency $\omega$, with the selection criterion presented in §3.4.2.




**(a)** Speed-accuracy trade-offs of Active Set for different iteration ranges and devices.




**(b)** Speed-accuracy trade-offs of Saddle Point for different iteration ranges and devices.

**Figure B.8:** Upper bounds to the adversarial vulnerability for the MNIST network trained with the verified training algorithm by Wong and Kolter (2018), from Lu and Kumar (2020). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better.

early cut addition tends to yield better bounds in the same time, suggesting that our selection criterion is effective before subproblem convergence.

### B.8.3 MNIST Incomplete Verification

We conclude the experimental appendix by presenting incomplete verification results (the experimental setup mirrors the one employed in section 3.8.2 and appendix B.8.1) 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 3.1 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 (§3.8.3),

and differently from the incomplete verification experiments in section 3.8.2 and appendix B.8.1, the network was trained with the verified training method by Wong and Kolter (2018). We compute the vulnerability to $\epsilon_{ver} = 0.15$ on the first 820 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 B.8, B.9 show that our solvers for problem (3.3) (Active Set and Saddle Point) yield 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 Gurobi 1 cut (for Active Set, 2500 as opposed to 600 in Figures 3.1, B.3). Finally, the smaller average gap between the bounds of Gurobi Planet and Gurobi 1 cut (especially with respect to Figure 3.1) suggests that the relaxation by Anderson et al. (2020) is less effective on this MNIST benchmark.

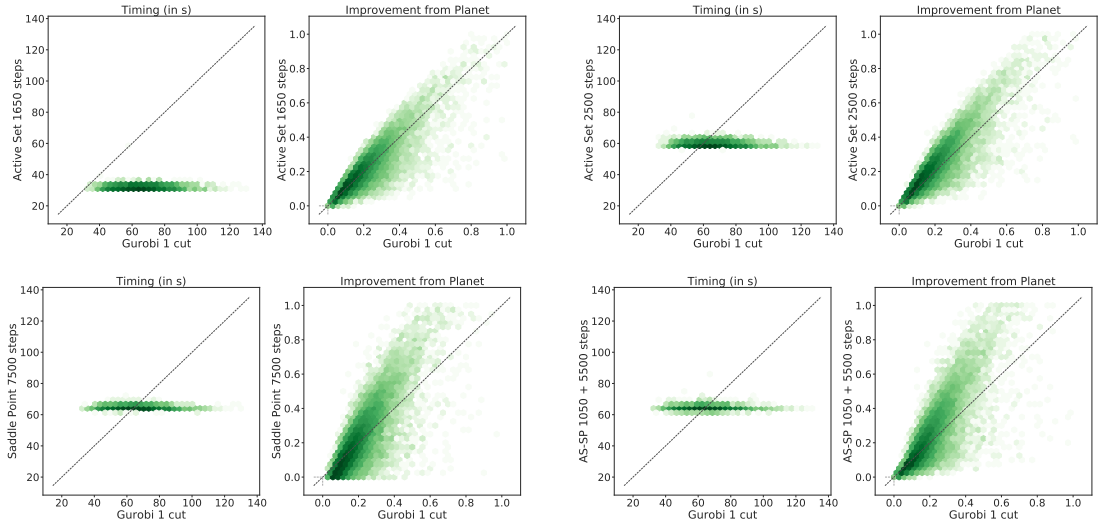



**Figure B.9:** Pointwise comparison for a subset of the methods on the data presented in Figure B.8. Comparison of runtime (left) and improvement from the Gurobi Planet bounds. For the latter, higher is better. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

# C

# IBP Regularization for Verified Adversarial Robustness via Branch-and-Bound

## Contents


C.1 Comparison between IBP-R and COLT . . . . . **186**
C.2 Complete Verification Problem . . . . . **187**
C.3 Intermediate Bounds . . . . . **187**
C.4 Interval Bound Propagation . . . . . **188**
C.5 UPB Branching: Beta-CROWN Dual and Planet relaxation . . **189**
C.5.1 Planet Relaxation . . . . . 189
C.5.2 Beta-CROWN Dual . . . . . 190
C.5.3 UPB Branching . . . . . 191
C.6 Experimental Details . . . . . **191**
C.6.1 Experimental Setting, Hyper-parameters . . . . . 192
C.6.2 Jax Porting of COLT . . . . . 193
C.7 Supplementary Branching Experiments . . . . . **194**

## C.1 Comparison between IBP-R and COLT

In this section, we provide a detailed comparison between COLT by Balunovic and Vechev (2020) (§4.2.3) and IBP-R (§4.3) in the context of training for adversarial robustness. Equation (C.1) pictorially highlights the differences between the two algorithms, which are summarized in table C.1.

| **Training Detail** | **COLT** | **IBP-R** |
|---|---|---|
| Stages | Proceeds in a stage-wise fashion: at each stage, a subset of the network parameters (all the $\boldsymbol{\theta}$ that are not in $\boldsymbol{\theta}^j$) is frozen, and the remaining parameters are optimized over. The number of training stages is $O(n)$. | All network parameters are optimized at once. |
| PGD domains | The attacks are carried out in the space of the $j$-th activations ($\mathbf{x}_j$). The frozen subset of the network is replaced by a zonotope-based convex outer-approximation, employed to define $\mathbb{C}_j(\mathbf{x})$ and $\mathbb{C}_{j-1}(\mathbf{x})$, the domains of the PGD attacks. At each stage, the zonotope domains are computed for train-time perturbations larger than those to be verified: $\epsilon_{\text{train},j} > \epsilon_{\text{ver}}$. In practice, the CIFAR-10 experiments from Balunovic and Vechev (2020) use $1.05 \leq \epsilon_{\text{train},j} \leq 1.512$. For the first stage: $\mathbb{C}_0(\mathbf{x}) := \mathcal{C}_+(\mathbf{x})$, $\mathbb{C}_{-1}(\mathbf{x}) := \mathbf{x}$. | The attacks are performed in the network input space, over a superset of the perturbations employed at verification. Our notation for $\mathcal{C}_+(\mathbf{x})$ corresponds to setting $\epsilon_{\text{train}} := \alpha\epsilon_{\text{ver}}$, with $1.6 \leq \alpha \leq 2.1$ in our experiments (after the mixing phase). Note that the first stage of COLT displays the same attack structure, with smaller employed $\alpha$ values. |
| Intermediate bounds | Computed via an approximation of the zonotope relaxation based on Cauchy random projections (Li et al., 2007). These bounds, computed for perturbation radius $\epsilon_{\text{train},j}$, are used both for the regularization term and to define the zonotope outer-approximations (which depend on intermediate bounds). | Computed via IBP, for perturbation radius $\epsilon_{\text{train}}$. Used for the regularization term. |
| Bounds regularization | Minimizes the area of the zonotope relaxation of ambiguous ReLUs for a single later per stage. This term produces a non-null gradient only for the parameters of the $(j+1)$-th layer. | Minimizes the area of the Planet relaxation (in practice, one half of the area of the zonotope relaxation) of ambiguous ReLUs for all layers at once (including the output space $\hat{x}_n$) . |
| Hyper-parameters | Each stage is potentially associated to a different regularization coefficient $\rho_j$, to a different train-time perturbation radius $\epsilon_{\text{train},j}$, and to a different learning rate $\eta_j$. In practice, Balunovic and Vechev (2020) tune the values for the first stage ($\rho_0$, $\epsilon_{\text{train},0}$, $\eta_0$), and then respectively decay $\rho_0$ and $\epsilon_{\text{train},0}$, and increase $\rho_0$, by a fixed and tunable quantity at each stage. Finally, Balunovic and Vechev (2020) set $\rho_{n-2} = 0$ in all cases. | $\rho$, $\alpha$, and $\eta$ are not altered throughout training. |
| Mixing phase | At each stage, $\kappa$ is linearly increased from 0 to 1. | $\kappa$ is linearly increased from 0 to 1, $\epsilon_{\text{train}}$ is linearly increased from 0 to $\alpha\epsilon_{\text{ver}}$. |

**Table C.1:** Main differences between COLT (§4.2.3) and IBP-R (§4.3).

COLT:

$$\left[\min_{\boldsymbol{\theta}^j} \mathbb{E}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \begin{cases} \kappa \left[\begin{array}{l} \max_{\mathbf{x}'\in\mathbb{C}_j(\mathbf{x})} \mathcal{L}(f^j(\boldsymbol{\theta}^j,\mathbf{x}'),\mathbf{y}) \\ +\rho_j \left[-\hat{\mathbf{l}}_{j+1}(\boldsymbol{\theta}^j)\right]_+^T [\hat{\mathbf{u}}_{j+1}(\boldsymbol{\theta}^j)]_+ \end{array}\right] \\ +(1-\kappa) \max\limits_{\mathbf{x}'\in\mathbb{C}_{j-1}(\mathbf{x})} \mathcal{L}(f^{j-1}(\boldsymbol{\theta}^{j-1},\mathbf{x}'),\mathbf{y}) \\ +\lambda \left\|\boldsymbol{\theta}^j\right\|_1, \end{cases}\right] \quad \forall j \in [\![0, n-2]\!]$$

IBP-R:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \begin{cases} \kappa \left[\begin{array}{l} \max_{\mathbf{x}\in\mathcal{C}_+(\mathbf{x})} \mathcal{L}(f(\boldsymbol{\theta},\mathbf{x}),\mathbf{y}) + \\ \frac{\rho}{2} \sum_{j=1}^n \left[-\hat{\mathbf{l}}_j(\boldsymbol{\theta})\right]_+^T [\hat{\mathbf{u}}_j(\boldsymbol{\theta})]_+ \end{array}\right] \\ + (1-\kappa)\mathcal{L}(f(\boldsymbol{\theta},\mathbf{x}),\mathbf{y}) + \lambda \left\|\boldsymbol{\theta}\right\|_1. \end{cases} \tag{C.1}$$

## C.2 Complete Verification Problem

Provided the network is in canonical form (Bunel et al., 2020b), complete verification amounts to finding sign of the minimum of the following problem, of which problem (4.2) is a convex outer-approximation:

$$\begin{array}{lll} \min\limits_{\mathbf{x},\hat{\mathbf{x}}} \ \hat{x}_n \quad \text{s.t.} & \mathbf{x}_0 \in \mathcal{C}(\mathbf{x}), & \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} & k \in [\![0, n-1]\!], \\ & \mathbf{x}_k = \sigma(\hat{\mathbf{x}}_k) & k \in [\![1, n-1]\!]. \end{array} \tag{C.2}$$

## C.3 Intermediate Bounds

As seen in §4.2.1, the network relaxations employed for incomplete verification (and, hence, complete verification via branch-and-bound) are defined in terms of bounds on the network pre-activations (intermediate bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$). Intermediate bounds are computed by solving instances of problem (4.2) over subsets of the network. For instance, for $\hat{\mathbf{l}}_i[j]$, the lower bound on $\hat{\mathbf{x}}_i[j]$:

$$\begin{array}{lll} \min\limits_{\mathbf{x},\hat{\mathbf{x}}} \ \hat{\mathbf{x}}_i[j] & \mathbf{x}_0 \in \mathcal{C}(\mathbf{x}), & \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} & k \in [\![0, i-1]\!], \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in \mathrm{Rel}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) & k \in [\![1, i-1]\!], \\ & \hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] & k \in [\![1, i-1]\!], \end{array} \tag{C.3}$$

where intermediate bounds until the $(i-1)$-th layer are needed. Overall, problem (C.3) needs to be solved twice per neuron: once for the lower bound, once for the upper bound, by flipping the sign of the last layer's weights. Therefore, intermediate bounds are often computed by relying on looser relaxations than for the output bounding (that is, solving for $\min_{\mathbf{x},\hat{\mathbf{x}}} \hat{x}_n$). See §4.4.1 for details on how we compute intermediate bounds within branch-and-bound.

## C.4 Interval Bound Propagation

Interval bound propagation (Gowal et al., 2018b; Mirman et al., 2018), a simple application of interval arithmetic (Sunaga, 1958; Hickey et al., 2001) to neural networks, implies solving a version of problem (4.2) where $\text{Rel}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ is the hyper-rectangle depicted in Figure C.2. We will use $\mathcal{B}$ to denote the corresponding feasible region. Furthermore, let us write $[\mathbf{x}]_- := \min(\mathbf{x}, \mathbf{0})$ and

$$\hat{\mathbf{l}}_n := \min_{\mathbf{x},\hat{\mathbf{x}}} \hat{x}_n \text{ s.t. } (\mathbf{x}, \hat{\mathbf{x}}) \in \mathcal{B}.$$

Due to the simplicity of the relaxation, the problem enjoys the following closed form solution:

$$\hat{\mathbf{l}}_n := \min_{\mathbf{x},\hat{\mathbf{x}}} \hat{x}_n = [W_n]_+ [\hat{\mathbf{l}}_{n-1}]_+ + [W_n]_- [\hat{\mathbf{u}}_{n-1}]_+ + \mathbf{b}_n.$$

Upper bounds can be computed by replacing $\hat{\mathbf{l}}_{n-1}$ with $\hat{\mathbf{u}}_{n-1}$, and viceversa. Note that the bounds at layer $n$ only depend on the intermediate bounds at layer $(n-1)$. Therefore, output bounds and all intermediate bounds can be computed at once by forward-propagating the bounds from the first layers, at the total cost of four network evaluations. Let $d$ be the input dimensionality: $\mathbf{x}_0 \in \mathbb{R}^d$. IBP is significantly less expensive than relaxations based on linear bounds (Wong and Kolter, 2018; Zhang et al., 2018), which incur a cost equivalent to $O(d)$ network evaluations for the same computation (Xu et al., 2020).

## C.5 UPB Branching: Beta-CROWN Dual and Planet relaxation

Our UPB (see §4.4.2) branching strategy makes use of variables from the Beta-CROWN (Wang et al., 2021a) dual objective. We will now describe the objective as well as its relationship to the Planet relaxation (see §4.2.1).

### C.5.1 Planet Relaxation

Let us denote by $\mathrm{Conv}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ the element-wise convex hull of the activation function, as a function of intermediate bounds. For ReLUs, this corresponds to the Planet relaxation, which is depicted in Figure 2.1 for the ambiguous case. By replacing $\mathrm{Rel}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ with $\mathrm{Conv}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k)$ in problem (4.2), we obtain:

$$\begin{aligned} \min_{\mathbf{x},\hat{\mathbf{x}}} \quad \hat{x}_n \qquad \text{s.t.} \qquad & \mathbf{x}_0 \in \mathcal{C}(\mathbf{x}), \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!], \\ & (\mathbf{x}_k, \hat{\mathbf{x}}_k) \in \mathrm{Conv}(\sigma, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k) && k \in [\![1, n-1]\!], \\ & \hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] && k \in [\![1, n-1]\!]. \end{aligned} \tag{C.4}$$

Problem (C.4) can be alternatively written as:

$$\begin{aligned} \max_{\boldsymbol{\alpha}\in[\mathbf{0},\mathbf{1}]} \min_{\mathbf{x},\hat{\mathbf{x}}} \quad & \hat{x}_n \\ \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\ & \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!], \\ & \underline{\mathbf{a}}_k(\boldsymbol{\alpha}_k) \odot \hat{\mathbf{x}}_k + \underline{\mathbf{b}}_k \le \mathbf{x}_k \le \bar{\mathbf{a}}_k \odot \hat{\mathbf{x}}_k + \bar{\mathbf{b}}_k && k \in [\![1, n-1]\!], \\ & \hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] && k \in [\![1, n-1]\!], \end{aligned} \tag{C.5}$$

where the coefficients are defined as follows:

$$\begin{aligned} \underline{\mathbf{a}}_k(\boldsymbol{\alpha}_k) = \boldsymbol{\alpha}_k, \quad \bar{\mathbf{a}}_k = \frac{\hat{\mathbf{u}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k}, \quad \bar{\mathbf{b}}_k = \frac{-\hat{\mathbf{l}}_k \odot \hat{\mathbf{u}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k} && \text{if } \hat{\mathbf{l}}_k \le 0 \text{ and } \hat{\mathbf{u}}_k \ge 0, \\ \bar{\mathbf{a}}_k = \underline{\mathbf{a}}_k(\boldsymbol{\alpha}_k) = 0 && \text{if } \hat{\mathbf{u}}_k \le 0, \\ \bar{\mathbf{a}}_k = \underline{\mathbf{a}}_k = 1 && \text{if } \hat{\mathbf{l}}_k \ge 0, \\ \underline{\mathbf{b}}_k = 0 && \text{in all cases.} \end{aligned} \tag{C.6}$$

For ambiguous ReLUs, $\bar{\mathbf{a}}_k \odot \hat{\mathbf{x}}_k + \bar{\mathbf{b}}_k$, represents the upper bounding constraint of Figure 2.1. The bias of such constraint, $\bar{\mathbf{b}}_k = \frac{-\hat{\mathbf{l}}_k \odot \hat{\mathbf{u}}_k}{\hat{\mathbf{u}}_k - \hat{\mathbf{l}}_k}$, gives the name to our branching strategy (Upper Planet Bias).

### C.5.2 Beta-CROWN Dual

Within branch-and-bound for ReLU networks, the constraints of the form $\hat{\mathbf{x}}_k \in [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k]$ can be usually omitted from (C.5), except when they capture the additional constraints associated to the domain splits (split constraints). For simplicity, we will enforce split constraints on all stable neurons, regardless of whether stability comes from actual split constraints or held before splitting. In this context, using (Salman et al., 2019b, equations (8), (9), (38)), the Lagrangian relaxation of problem (C.5) can be written as follows:

$$\begin{aligned} \max_{\boldsymbol{\alpha}\in[\mathbf{0},\mathbf{1}],\boldsymbol{\mu},\boldsymbol{\lambda},\beta} \min_{\mathbf{x},\hat{\mathbf{x}}} & \left[ \begin{array}{l} W_n\mathbf{x}_{n-1} + b_n + \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T \left(\hat{\mathbf{x}}_k - W_k\mathbf{x}_{k-1} - \mathbf{b}_k\right) \\ + \sum_{k=1}^{n-1} [\boldsymbol{\lambda}_k]_-^T \left(\mathbf{x}_k - (\underline{\mathbf{a}}_k(\boldsymbol{\alpha}_k) \odot \hat{\mathbf{x}}_k + \underline{\mathbf{b}}_k)\right) \\ + \sum_{k=1}^{n-1} [\boldsymbol{\lambda}_k]_+^T \left(\mathbf{x}_k - (\bar{\mathbf{a}}_k \odot \hat{\mathbf{x}}_k + \bar{\mathbf{b}}_k)\right) \\ + \sum_{k=1}^{n-1} \boldsymbol{\beta}_k^T \mathbf{x}_k \mathbb{1}_{\hat{\mathbf{u}}_k \le 0} - \sum_{k=1}^{n-1} \boldsymbol{\beta}_k^T \mathbf{x}_k \mathbb{1}_{\hat{\mathbf{l}}_k \ge 0} \end{array} \right. \\ \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}. \end{aligned} \tag{C.7}$$

By enforcing the coefficient of the unconstrained $\mathbf{x}$ and $\hat{\mathbf{x}}$ terms to be null, we obtain the following:

$$\begin{aligned} d_P = \max_{\boldsymbol{\alpha}\in[\mathbf{0},\mathbf{1}],\ \beta\ge 0} & \left\{ \min_{\mathbf{x}_0\in\mathcal{C}} \left(-\bar{\boldsymbol{\mu}}_1^T W_1 \mathbf{x}_0\right) + b_n - \sum_{k=1}^{n-1} \left( [\bar{\boldsymbol{\lambda}}_k]_-^T \underline{\mathbf{b}}_k + [\bar{\boldsymbol{\lambda}}_k]_+^T \bar{\mathbf{b}}_k + \bar{\boldsymbol{\mu}}_k^T \mathbf{b}_k \right) \right\} \\ \text{s.t.} \quad & \bar{\boldsymbol{\lambda}}_{n-1} = -W_n^T, \\ & \bar{\boldsymbol{\mu}}_k = \bar{\mathbf{a}}_k \odot [\bar{\boldsymbol{\lambda}}_k]_+ + \boldsymbol{\alpha}_k \odot [\bar{\boldsymbol{\lambda}}_k]_- + \mathbf{s}_k \odot \boldsymbol{\beta}_k \qquad k \in [\![1, n-1]\!], \\ & \bar{\boldsymbol{\lambda}}_{k-1} = W_k^T \bar{\boldsymbol{\mu}}_k \qquad\qquad\qquad\qquad k \in [\![2, n-1]\!], \\ \text{where:} \quad & \mathbf{s}_k = \mathbb{1}_{\hat{\mathbf{l}}_k \ge 0} - \mathbb{1}_{\hat{\mathbf{u}}_k \le 0}. \end{aligned} \tag{C.8}$$

Problem (C.8) corresponds to the $\beta$-CROWN objective, as it can be easily seen by comparing it with (Wang et al., 2021a, equation (20)) and pointing out that, in their formulation, the input domain represents $\ell_\infty$ norm perturbations of radius $\epsilon$. The dual variables $\boldsymbol{\beta}$, which give the name to the algorithm, are necessary only

| Network Specifications | | Network Architecture |
|---|---|---|
| Perturbation radius: | $\epsilon_{\text{ver}} = 2/255$ | Conv2d(3, 32, 3, stride=1, padding=1) |
| Activation: | ReLU | Conv2d(32,32,4, stride=2, padding=1) |
| Total activations: | 49402 | Conv2d(32,128,4, stride=2, padding=1) |
| Total parameters: | 2133736 | linear layer of 250 hidden units |
| | | linear layer of 10 hidden units |
| Perturbation radius: | $\epsilon_{\text{ver}} = 8/255$ | Conv2d(3, 32, 5, stride=2, padding=2) |
| Activation: | ReLU | Conv2d(32,128,4, stride=2, padding=1) |
| Total activations: | 16643 | linear layer of 250 hidden units |
| Total parameters: | 2118856 | linear layer of 10 hidden units |

**Figure C.1:** Specifications of the employed network architectures for the experiments of §4.6.

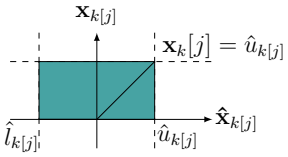



**Figure C.2:** Depiction of the IBP hyper-rectangle.

for the neurons whose domains have been split within branch-and-bound (Wang et al., 2021a).

### C.5.3 UPB Branching

The key observation behind our UPB branching strategy is that the $[\bar{\boldsymbol{\lambda}}_k]_+^T \bar{\mathbf{b}}_k$ term is present only for ambiguous neurons (see the coefficients in equation (C.6), note that $[\bar{\boldsymbol{\lambda}}_k]_-^T \underline{\mathbf{b}}_k = 0$). We can hence heuristically employ it as a proxy for the improvement that a split constraint will have on $d_P$ from (C.8). Replacing $\underline{\mathbf{b}}_k$ with its definition in $[\bar{\boldsymbol{\lambda}}_k]_+^T \bar{\mathbf{b}}_k$ yields the branching scores $\mathbf{s}_{\text{UPB},k}$ in equation (4.6). Given the values for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ obtained in the last bounding step within branch-and-bound, $[\bar{\boldsymbol{\lambda}}_k]_+$, and hence the scores, can be computed using their definition in problem (C.8), which has a cost equivalent to a single gradient backpropagation through the network. The UPB branching strategy then proceeds by enforcing split constraints on the neuron associated to the largest $\mathbf{s}_{\text{UPB},k}$ score throughout the network.

## C.6 Experimental Details

We now present experimental details that were omitted from §4.6. In particular, we describe the computational setup, network architectures, employed hyper-parameters, and details concerning the Jax porting of the COLT algorithm (Balunovic and Vechev, 2020).

### C.6.1 Experimental Setting, Hyper-parameters

All the experiments were run on a single GPU, either an Nvidia Titan Xp, or an Nvidia Titan V. The timing experiments for verified training were all run on an Nvidia Titan V GPU, on a machine with a 20-core Intel i9-7900X CPU. The branching experiments were instead consistently run on an Nvidia Titan XP GPU, on a machine with a 12-core Intel i7-6850K CPU.

**Training hyper-parameters** COLT was run with the hyper-parameters provided by the authors (Balunovic and Vechev, 2020), whereas the hyper-parameters for IBP-R are listed in Figure C.3. Note that the learning rate is annealed, at each epoch, only after the mixing phase of training. Please refer to equation (4.5) for the meaning of the various hyper-parameters. We did not tune the parameters of the PGD attacks, which were set as for COLT: we report them for convenience.

**Verification hyper-parameters** We now complement section §4.4.1 with omitted details concerning the configuration of the OVAL branch-and-bound framework (Bunel et al., 2020a,b; De Palma et al., 2021c). These details apply to both the training and the branching experiments. The entire verification procedure is run on a single GPU. For the UPB and SR branching strategies, the branching and bounding steps are performed in parallel for batches of 600 and 1200 subproblems, respectively. For the FSB branching strategy, these numbers were reduced to 500 and 1000 subproblems, respectively, in order to prevent PyTorch out-of-memory errors. $\beta$-CROWN is run with a dynamically adjusted number of iterations (see (De Palma et al., 2021c, section 5.1.2)) of the Adam optimizer (Kingma and Ba, 2015), with a learning rate of 0.1, decayed by 0.98 at each iteration. Similarly, $\alpha$-CROWN for the intermediate bounds is run with Adam for 5 iterations, a learning rate of 1, decayed by 0.98 at each iteration. Early termination is triggered when an exponential moving average of the expected branching improvement, computed on the subproblem with the smallest lower bound within the current sub-problem batch, suggests that the decision threshold will be crossed after the timeout. The

| Hyperparameter | Value |
|---|---|
| IBP-R with and without masking | |
| Optimizer | SGD |
| Learning rate | $10^{-2}$ |
| Learning rate exponential decay | 0.95 |
| Batch size | 100 |
| Total training steps | 800 |
| Mixing steps | 600 |
| PGD attack steps | 8 |
| PGD attack step size | 0.25 |
| $\alpha$ | 2.1 |
| $\lambda$ | $2 \times 10^5$ |
| $\frac{\varrho}{2}$ | $10^4$ |

**(a)** $\epsilon_{\text{ver}} = 2/255$.

| Hyperparameter | Value |
|---|---|
| IBP-R with and without masking | |
| Optimizer | SGD |
| Learning rate | $10^{-2}$ |
| Learning rate exponential decay | 0.95 |
| Batch size | 150 |
| Total training steps | 800 |
| Mixing steps | 600 |
| PGD attack steps | 8 |
| PGD attack step size | 0.25 |
| $\lambda$ | $10^5$ |
| IBP-R | |
| $\alpha$ | 1.7 |
| $\frac{\varrho}{2}$ | $5 \times 10^3$ |
| IBP-R with Masking | |
| $\alpha$ | 1.6 |
| $\frac{\varrho}{2}$ | $10^2$ |

**(b)** $\epsilon_{\text{ver}} = 8/255$.

**Figure C.3:** IBP-R hyper-parameters for the experiment of table 4.1.

time to deplete the sub-problem queue (estimated via the runtime per bounding batch) is also added to the estimated time to termination.

**Network architectures** Figure C.1 reports the details of the employed network architectures for both the training and the branching experiments.

### C.6.2 Jax Porting of COLT

In order to ensure a fair timing comparison with our IBP-R, we ported COLT, whose original implementation is in PyTorch (Paszke et al., 2019), to Jax (Bradbury et al., 2018). The porting resulted in speed-up factors of around 5.7 and 4.4 for the $\epsilon_{\text{ver}} = 2/255$ and $\epsilon_{\text{ver}} = 8/255$ experiments, respectively: see table 4.1. The standard accuracy results were similar to the original implementation (see table 4.1), testifying the validity of the porting. The verified accuracy of our experiments is larger than the one reported in the literature: this is likely due to the different, and arguably more effective, verification procedure that we employed. We conclude this subsection by reporting the main functional difference of our Jax porting with respect to the original implementation. COLT's Cauchy random projections-approximated zonotope intermediate bounds require extensive use of median values computed over arrays. The median values are employed as estimators for the $\ell_1$ norm (Li et al., 2007) of the zonotope propagation matrices. Unfortunately, these

operations do not scale very well in Jax (Bradbury et al., 2020). Therefore, we rely on a different $\ell_1$ estimator, based on the geometric mean (Li et al., 2007), and we clip the employed Cauchy samples to ensure numerical stability during training.

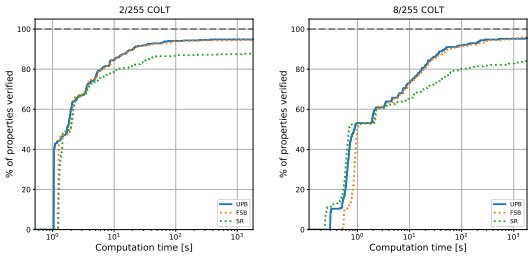
## C.7 Supplementary Branching Experiments

We now complement the branching results in §4.6.2 by repeating the same complete verification experiment on two COLT-trained networks from table 4.1. Figure C.4(a) confirms the results from §4.6.2, albeit with reduced margins between UPB and FSB. UPB yields small ($< 6\%$) improvements on the average verification times with respect to FSB. In addition, UPB increases the average number of visited subproblems on the $\epsilon_{\text{ver}} = 8/255$ network, and reduces it for $\epsilon_{\text{ver}} = 2/255$, where it also decreases the number of visited subproblems and timed-out properties. As for the IBP-R-trained networks, UPB yields branching decisions competitive with those of FSB while incurring smaller overheads. Finally, as seen in §4.6.2, SR is significantly slower than both UPB and FSB.

| | $\epsilon_{\text{ver}} = 2/255$ COLT | | | $\epsilon_{\text{ver}} = 8/255$ COLT | | |
|---|---|---|---|---|---|---|
| Method | time [s] | subproblems* | %Timeout | time [s] | subproblems* | %Timeout |
| UPB | **98.87** | **257.17** | **5.16** | **100.19** | 2924.80 | 4.46 |
| FSB | 104.41 | 379.77 | 5.41 | 101.00 | **2840.52** | **4.09** |
| SR | 229.77 | 5894.53 | 12.29 | 318.21 | 20 916.78 | 15.99 |

*computed on the properties that did not time out for neither UPB nor FSB. The inclusion of timed-out results in the average leads to an overestimation of the number of subproblems for the less expensive branching strategy.

**(a)** Comparison of average runtime, average number of solved subproblems and the percentage of timed out properties. The best performing method is highlighted in bold.




**(b)** Cactus plots: percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

**Figure C.4:** Complete verification performance of different branching strategies, on two COLT-trained CIFAR-10 networks from §4.6.1.

# D

# In Defense of the Unitary Scalarization for Deep Multi-Task Learning

## Contents


D.1 Societal Impact . . . . . **196**
D.2 Supplement to the Overview of Multi-Task Optimizers . . . . . **196**
D.2.1 MGDA . . . . . 196
D.2.2 IMTL . . . . . 198
D.2.3 PCGrad . . . . . 200
D.2.4 GradDrop . . . . . 202
D.3 Experimental Setting, Reproducibility . . . . . **204**
D.3.1 Supervised Learning . . . . . 204
D.3.2 Reinforcement Learning . . . . . 205
D.3.3 Software Acknowledgments and Licenses . . . . . 207
D.4 Supplementary Supervised Learning Experiments . . . . . **208**
D.4.1 Addendum . . . . . 208
D.4.2 Unregularized Experiments . . . . . 208
D.4.3 Sign-Agnostic GradDrop . . . . . 211
D.5 Supplementary Reinforcement Learning Experiments . . . . . **211**
D.5.1 Addendum . . . . . 212
D.5.2 Ablation studies . . . . . 213
D.5.3 Sensitivity to Reward Normalization . . . . . 213

## D.1 Societal Impact

Due to the object of its study, our work does not have a direct societal impact. However, as any machine learning paper, it can potentially negatively effect the society through automation and loss of jobs. While it is hard to anticipate any particular risk, as any technology, if not regulated properly, it might lead to growing social and economic inequality.

On the positive side, our work might have a positive environmental impact since it advocates for simpler and more economical methods which will reduce energy consumption in data centers. Finally, simpler methods are usually easier to understand, which is beneficial in terms of explainability, an important factor for real-life applications.

## D.2 Supplement to the Overview of Multi-Task Optimizers

This section presents the proofs and the technical results omitted from section 5.5, along with a description of the use of per-task gradients with respect to the last shared activation for encoder-decoder architectures (usually less expensive than per-task gradients with respect to shared parameters).

### D.2.1 MGDA

**Proposition 5.** *The* MGDA SMTO *(Sener and Koltun, 2018) converges to a superset of the convergence points of unitary scalarization. More specifically, it converges to any point* $\boldsymbol{\theta}^*_{\|}$ *such that:* $\mathbf{0} \in$ *Conv*$(\{\nabla_{\boldsymbol{\theta}^*_{\|}} \mathcal{L}_i \mid i \in \mathcal{T}\})$.

*Proof.* As shown by Désidéri (2012), equation (5.3) is a simplex-constrained norm-minimization problem. In other words, the argument of the minimum is the projection of $\mathbf{0}$ onto the feasible set. Therefore:

$$\mathbf{g} = \mathbf{0} \iff \mathbf{0} \in \text{Conv}(\{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \mid i \in \mathcal{T}\}).$$

It then suffices to point out that $\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i = \mathbf{0} \iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|} \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i = \mathbf{0} \Rightarrow \mathbf{0} \in \text{Conv}(\{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \mid i \in \mathcal{T}\})$ to conclude the proof. □

Due to the cost of computing per-task gradients, Sener and Koltun (2018) propose MGDA-UB, which replaces the gradients wrt the parameters $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$ with the gradients wrt the shared activation $\nabla_{\mathbf{z}}\mathcal{L}_i$ in the computation of the coefficients of $\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$. This yields an upper bound on the objective of equation (5.3), thus restricting the set of points the algorithm convergences to. Rather than directly relying on $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$, $\mathbf{g}$ can then be obtained by computing the gradient of $\sum_{i\in\mathcal{T}} \alpha_i \mathcal{L}_i$ via reverse-mode differentiation, hence saving memory and compute.

**Corollary 3.** *The MGDA-UB* SMTO *by Sener and Koltun (2018) converges to any point such that:* $\mathbf{0} \in \mathit{Conv}(\{\nabla_{\mathbf{z}}\mathcal{L}_i \mid i \in \mathcal{T}\})$. *Furthermore, if* $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}$ *is non-singular, it converges to a superset of the convergence points of the unitary scalarization.*

*Proof.* The first part of the proof proceeds as the proof of proposition 5, noting that the MGDA-UB update is associated to the following problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\|\mathbf{g}\|_2^2 \\ \text{s.t.} \quad & \sum_i \alpha_i \nabla_{\mathbf{z}}\mathcal{L}_i = -\mathbf{g}, \quad \sum_{i\in\mathcal{T}} \alpha_i = 1, \\ & \alpha_i \geq 0 \qquad \forall\, i \in \mathcal{T}. \end{aligned}$$

In order to show that a stationary point of the unitary scalarization satisfies $\mathbf{0} \in \mathrm{Conv}(\{\nabla_{\mathbf{z}^*}\mathcal{L}_i \mid i \in \mathcal{T}\})$, we will assume $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}$ is non-singular, as done by Sener and Koltun (2018, theorem 1). Then, relying on the chain rule, the result follows from:

$$\begin{aligned} \sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0} &\iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0} \\ &\iff \sum_{i\in\mathcal{T}} \frac{\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}}{|\mathcal{T}|}\nabla_{\mathbf{z}}\mathcal{L}_i = \mathbf{0} \\ &\iff \left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}\right)^{-1} \frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}} \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|}\nabla_{\mathbf{z}}\mathcal{L}_i = \mathbf{0} \\ &\iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|}\nabla_{\mathbf{z}}\mathcal{L}_i = \mathbf{0} \\ &\Rightarrow \mathbf{0} \in \mathrm{Conv}(\{\nabla_{\mathbf{z}}\mathcal{L}_i \mid i \in \mathcal{T}\}) \end{aligned}$$

□

### D.2.2 IMTL

**Proposition 6.** *IMTL by Liu et al. (2021c) updates* $\boldsymbol{\theta}_{\|}$ *by taking a step in the* steepest descent direction whose *cosine similarity with per-task gradients is the same across tasks.*

*Proof.* First, equation (5.4) solves the linear system in $\boldsymbol{\alpha} := [\alpha_1, \dots, \alpha_m]$ given by:

$$\begin{aligned}
&\mathbf{g}^T \left( \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_1}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_1 \right\|} - \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\|} \right) = \mathbf{0} \qquad \forall\, i \in \mathcal{T} \setminus \{1\}, \\
&\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i, \quad \sum_{i \in \mathcal{T}} \alpha_i = 1,
\end{aligned}$$

which corresponds to finding a point of $\mathcal{A}' := \text{Aff}(\left\{ \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i |\ i \in \mathcal{T} \right\})$ which is orthogonal to $\mathcal{A} := \text{Aff}\left( \left\{ \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\|} |\ i \in \mathcal{T} \right\} \right)$. To see this, it suffices to point out that any point orthogonal to $\mathcal{A}$ is also orthogonal to the vector subspace spanned by differences of vectors belonging to $\mathcal{A}$. As this subspace has $m-1$ dimensions, any vector orthogonal to $\left( \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_1}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_1 \right\|} - \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\|} \right)$ for each $i \in \mathcal{T} \setminus \{1\}$ is orthogonal to the entire subspace.

Second, consider the problem of finding a point in $\mathcal{A}$ that is orthogonal to the linear subspace spanned by differences of vectors in $\mathcal{A}$. In other words, we seek the projection of $\mathbf{0}$ onto $\mathcal{A}$. Recalling the definition of $\mathcal{A}$, we can write:

$$\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2} \left\| \mathbf{g}' \right\|_2^2 \\
\text{s.t.} \quad & \sum_i \alpha_i \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\|} = -\mathbf{g}', \quad \sum_i \alpha_i = 1.
\end{aligned} \tag{D.1}$$

The solution of equation (D.1) is always collinear to the solution of equation (5.4). In fact, if a vector $\mathbf{g} \in \mathcal{A}'$ is orthogonal to the affine subspace $\mathcal{A}$ (or to the linear subspace spanned by differences of its members), then

$$\gamma \mathbf{g} = \left( -\gamma \sum_i \left( \alpha_i \left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\| \right) \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\|} \right)$$

is orthogonal to $\mathcal{A}$ as well, and $\gamma = \frac{1}{\sum_i \left( \alpha_i \left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\| \right)} \implies \gamma \mathbf{g} \in \mathcal{A}$.

Finally, equation (D.1) differs from equation (5.3) in two aspects: $\boldsymbol{\alpha}$ is not constrained to be non-negative (hence the convex hull is replaced by the affine hull), and the task vectors are normalized. Therefore, equation (D.1) is the dual of:

$$\begin{aligned} \min_{\mathbf{g},\epsilon} \quad & \epsilon + \frac{1}{2}\|\mathbf{g}\|_2^2 \\ \text{s.t.} \quad & \frac{\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i^T}{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i\right\|}\mathbf{g} = \epsilon \qquad \forall\, i \in \{1,\dots,m\}\,. \end{aligned} \tag{D.2}$$

The proposition then follows by comparing equation (D.2) with equation (5.2), and recalling that IMTL-L only adds a scaling factor to the chosen update direction. □

**Corollary 1.** IMTL *by Liu et al. (2021c) converges to a superset of the Pareto-stationary points for* $\boldsymbol{\theta}_\parallel$ *(and hence of the convergence points of the unitary scalarization). More specifically, it converges to any point* $\boldsymbol{\theta}_\parallel^*$ *such that:*

$$\mathbf{0} \in \mathit{Aff}\left(\left\{\nabla_{\boldsymbol{\theta}_\parallel^*}\mathcal{L}_i / \left\|\nabla_{\boldsymbol{\theta}_\parallel^*}\mathcal{L}_i\right\| \mid i \in \mathcal{T}\right\}\right).$$

*Proof.* Inspecting equation (D.2), which yields a collinear point to the IMTL update, reveals that IMTL might converge to non Pareto-stationary points: due to the restrictive equality constraints, the minimizer of equation (D.2) might be $\mathbf{0}$ even if a descent direction exists. Furthermore, its dual, equation (D.1), implies that:

$$\begin{aligned} \mathbf{g} = \mathbf{0} &\iff \mathbf{0} \in \operatorname{Aff}\left(\left\{\frac{\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i\right\|} \mid i \in \mathcal{T}\right\}\right) \\ &\iff \mathbf{0} \in \operatorname{Aff}\left(\left\{\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i \mid i \in \mathcal{T}\right\}\right), \end{aligned}$$

which, noting that $\operatorname{Conv}(\mathcal{A}) \subseteq \operatorname{Aff}(\mathcal{A})$ for any $\mathcal{A}$, concludes the proof. □

Similarly to MGDA-UB, Liu et al. (2021c) advocate using $\nabla_{\mathbf{z}}\mathcal{L}_i$ in place of $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i$ while solving equation (5.4), typically reducing the cost of computing the coefficients of $\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i$.

**Corollary 4.** *When employing the approximation of problem (5.4) that relies on* $\nabla_{\mathbf{z}}\mathcal{L}_i$*, IMTL by Liu et al. (2021c) converges to* $\mathbf{0} \in \mathit{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right)$*. If* $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_\parallel}$ *is non-singular, this is a superset of of the convergence points of the unitary scalarization.*

*Proof.* Following the proof of proposition 6, the following problem yields a collinear point to the $\nabla_{\mathbf{z}}\mathcal{L}_i$-approximate IMTL update:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\left\|\mathbf{g}'\right\|_2^2 \\ \text{s.t.} \quad & \sum_i \alpha_i \frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} = -\mathbf{g}', \quad \sum_i \alpha_i = 1. \end{aligned}$$

Therefore:

$$\mathbf{g} = \mathbf{0} \iff \mathbf{0} \in \operatorname{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right).$$

Finally, assuming $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}$ is non-singular, we can replicate the procedure in the proof of corollary 3 to get:

$$\begin{aligned} &\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0} \iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|}\nabla_{\mathbf{z}}\mathcal{L}_i = \mathbf{0} \\ &\iff \sum_{i\in\mathcal{T}} \frac{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|}{|\mathcal{T}|}\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} = \mathbf{0} \\ &\iff \left(\frac{|\mathcal{T}|}{\sum_{i\in\mathcal{T}}\left(\|\nabla_{\mathbf{z}}\mathcal{L}_i\|\right)}\right)\sum_{i\in\mathcal{T}} \frac{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|}{|\mathcal{T}|}\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} = \mathbf{0} \\ &\Rightarrow \mathbf{0} \in \operatorname{Conv}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right) \\ &\Rightarrow \mathbf{0} \in \operatorname{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right), \end{aligned}$$

which shows that $\operatorname{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right)$ contains the convergence points of the unitary scalarization. □

### D.2.3 PCGrad

**Proposition 7.** *PCGrad is equivalent to a dynamic, and possibly stochastic, loss rescaling for $\boldsymbol{\theta}_{\|}$. At each iteration, per-task gradients are rescaled as follows:*

$$\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i \leftarrow \left(1 + \textstyle\sum_{j\in\mathcal{T}\setminus\{i\}} d_{ji}\right)\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i, \; d_{ji} \in \left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|}\right].$$

*Furthermore, if $|\mathcal{T}| > 2$, $d_{ji}$ is a random variable, and the above range contains its support.*

*Proof.* We start by pointing out that:

$$\begin{aligned}\left[\frac{-\mathbf{g}_i^T\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|^2}\right]_+ &= \left[\frac{-\mathbf{g}_i^T\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|}\right]_+ \frac{1}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|} \\ &= \left[-\cos(\mathbf{g}_i, \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j)\,\|\mathbf{g}_i\|\right]_+ \frac{1}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|} \\ &\in \left[0, \frac{\|\mathbf{g}_i\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|}\right].\end{aligned}$$

As $\mathbf{g}_i$ is obtained by iterative projections of $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$ onto the normals of $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\ \forall j \in \mathcal{T}\setminus\{i\}$, and the norm of a vector can only decrease or remain unvaried after projections, we can write the coefficient of each $\mathbf{g}_i$ update as:

$$d_{ij} := \left[\frac{-\mathbf{g}_i^T\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|^2}\right]_+ \in \left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|}\right],\ \ \forall i \neq j.$$

Furthermore, if $|\mathcal{T}| > 2$ the contraction factor $\frac{\|\mathbf{g}_i\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|}$ for the norm of $g_i$ depends on the ordering of the projections, which is stochastic by design (Yu et al., 2020). Therefore, $d_{ij}$ a random variable whose support is contained in $\left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right\|}\right]$. Finally, exploiting the definition of $d_{ij}$, we can re-write equation (5.5) as:

$$\begin{aligned}-\mathbf{g} &= \sum_{i\in\mathcal{T}}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i + \sum_{i\in\mathcal{T}}\sum_{j\in\mathcal{T}\setminus\{i\}} d_{ij}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j = \sum_{i\in\mathcal{T}}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i + \sum_{j\in\mathcal{T}}\sum_{i\in\mathcal{T}\setminus\{j\}} d_{ji}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i \\ &= \sum_{j\in\mathcal{T}}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j + \sum_{j\in\mathcal{T}}\sum_{i\in\mathcal{T}\setminus\{j\}} d_{ji}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \sum_{j\in\mathcal{T}}\left(\sum_{i\in\mathcal{T}\setminus\{j\}} d_{ji}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i + \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right).\end{aligned}$$

Introducing (and then removing, using their definition) dummy variables $d_{jj} = 1$:

$$\begin{aligned}-\mathbf{g} &= \sum_{j\in\mathcal{T}}\left(\sum_{i\in\mathcal{T}\setminus\{j\}} d_{ji}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i + d_{jj}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right) = \sum_{j\in\mathcal{T}}\left(\sum_{i\in\mathcal{T}} d_{ji}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right) = \sum_{i\in\mathcal{T}}\left(\sum_{j\in\mathcal{T}} d_{ji}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right) \\ &= \sum_{i\in\mathcal{T}}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\left(\sum_{j\in\mathcal{T}} d_{ji}\right) = \sum_{i\in\mathcal{T}}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\left(1 + \sum_{j\in\mathcal{T}\setminus\{i\}} d_{ji}\right),\end{aligned}$$

from which the result trivially follows. □

**Corollary 2.** *If $|\mathcal{T}| = 2$, PCGrad will stop at any point where $\cos(\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1, \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2) = -1$. Furthermore, if $\mathcal{L}_1$ and $\mathcal{L}_2$ are differentiable, and $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{MT}$ is L-Lipschitz with $L > 0$, PCGrad with step size $t < \frac{1}{L}$ converges to a superset of the convergence points of the unitary scalarization.*

*Proof.* Let us start from the first statement, which does not require any assumption on the loss landscape. From proposition 7, we get:

$$\begin{aligned}
-\mathbf{g} &= \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1\left(1+d_{21}\right)+\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2\left(1+d_{12}\right) \\
&= \left(1+\left[\frac{-\cos(\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1,\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2)\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1\right\|}\right]_+\right)\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1 \\
&+ \left(1+\left[\frac{-\cos(\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1,\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2)\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2\right\|}\right]_+\right)\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2,
\end{aligned}$$

which shows that, in case of conflicting gradient directions, gradient norms are rebalanced proportionally to the angle between them. For $\cos(\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1,\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2)=-1$, the above evaluates to:

$$-\mathbf{g}=\left(\frac{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1\right\|+\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1\right\|}\right)\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1+\left(\frac{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1\right\|+\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2\right\|}\right)\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2.$$

The first part of the result then follows by pointing out that, if $\cos(\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1,\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2)=-1$, then $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_1=-\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_2$, and hence $\mathbf{g}=\mathbf{0}$. We remark that a similar proof appears in (Yu et al., 2020, theorem 1 and proposition 1). However, our derivation relaxes the author's assumptions on $\mathcal{L}^{\mathrm{MT}}$ and is therefore applicable to the training of neural networks.

Finally, given the assumptions on differentiability and smoothness, we need to prove that PCGrad converges to the stationary points of the unitary scalarization: this directly follows from (Yu et al., 2020, proposition 1). □

### D.2.4 GradDrop

**Proposition 10.** *Let us assume, as often demonstrated in the single-task case (Ma et al., 2018; Allen-Zhu et al., 2019), that the multi-task network has the capacity to interpolate the data on all tasks at once:* $\min_{\boldsymbol{\theta}}\mathcal{L}^{MT}=\sum_{i\in\mathcal{T}}\min_{\boldsymbol{\theta}}\mathcal{L}_i$*, and that its training by gradient descent attains such global minimum. Then, if* $\inf_{\boldsymbol{\theta}}\mathcal{L}_i>-\infty\ \forall\ i\in\mathcal{T}$*, unitary scalarization converges to a joint minimum.*

*Proof.* It suffices to point out that if $\mathcal{L}^{\mathrm{MT}}(\boldsymbol{\theta}^*)=\sum_{i\in\mathcal{T}}\min_{\boldsymbol{\theta}}\mathcal{L}_i$, then the globally optimal loss is attained for all tasks. In other words $\mathcal{L}_i(\boldsymbol{\theta}^*)=\min_{\boldsymbol{\theta}}\mathcal{L}_i\ \forall i\in\mathcal{T}$,

and hence $\nabla_{\boldsymbol{\theta}^*}\mathcal{L}_i = \mathbf{0}\ \forall\ i \in \mathcal{T}$ (joint minimum). Furthermore, running gradient descent on $\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathrm{MT}}$ corresponds to the unitary scalarization (§5.3), which concludes the proof. □

**Proposition 8.** *Let* $\mathcal{L}^{RGD}(\boldsymbol{\theta}_{\|}) := \sum_{i\in\mathcal{T}} u_i\mathcal{L}_i(\boldsymbol{\theta}_{\|})$*, where* $u_i \sim$ *Bernoulli*$(p)$ $\forall i \in \mathcal{T}$ *and* $p \in (0, 1]$*. The gradient* $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{RGD}$ *is always zero if and only if* $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0}\ \forall i \in \mathcal{T}$*. In other words, the result from (Chen et al., 2020, proposition 1) can be obtained without any information on the sign of per-task gradients.*

Proposition 8 can be proved by adapting the proof from Chen et al. (2020, proposition 1): it suffices to replace $f(\mathcal{P})$ with the Bernoulli parameter $p$, which is non-negative by definition. In our opinion, this seriously undermines the conflicting gradient hypothesis that motivated GradDrop. For the reader's convenience, we now provide a straightforward and self-contained proof.

*Proof.* Let us start from the statement on $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{\mathrm{RGD}}$. If $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0}\ \forall i \in \mathcal{T}$, then $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{\mathrm{RGD}} = \mathbf{0}$ with probability one. On the other hand, if $\exists j : \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j \neq \mathbf{0}$, then:

$$\begin{aligned}\mathbb{P}\left[\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{\mathrm{RGD}} \neq \mathbf{0}\right] &\geq \mathbb{P}\left[\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{\mathrm{RGD}} = \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j\right] \\ &= p(1-p)^{m-1} > 0,\end{aligned}$$

where the first inequality comes from the fact that $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{\mathrm{RGD}} = \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_j$ is only one of the many instances of a non-null $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{\mathrm{RGD}}$. □

Let $\mathrm{sign}(\mathbf{x})$ stand for the element-wise sign operator applied on $\mathbf{x}$. On encoder-decoder architectures, similarly to MGDA and IMTL (see appendices D.2.1 and D.2.2), the authors do not apply GradDrop on $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$, but rather on a the usually less expensive $\nabla_{\mathbf{z}}\mathcal{L}_i$. In more detail, they compute the GradDrop sign purity scores $\mathbf{p}$ from equation (5.6) on $\sum_{i=1}^{n}\left(\mathrm{sign}(\mathbf{z}) \odot \nabla_{\mathbf{z}}\mathcal{L}_i\right)[i] \in \mathbb{R}^r$, and then apply equation (5.6) on the $\nabla_{\mathbf{z}}\mathcal{L}_i$ gradients, yielding a vector $\mathbf{g}_z \in \mathbb{R}^{n\times r}$. Then, relying on reverse-mode differentiation, the update direction in the space of the parameters $\boldsymbol{\theta}_{\|}$ is obtained via a Jacobian-vector product: $\mathbf{g} = -\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}\right)^T \mathbf{g}_z$. Such a computation replaces the similar $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}^{\mathrm{MT}} = \left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}\right)^T \nabla_{\mathbf{z}}\mathcal{L}^{\mathrm{MT}}$ from the unitary scalarization.

## D.3 Experimental Setting, Reproducibility

We now present details concerning the experimental settings from §5.4, including details on the employed open-source software, hardware specifications, and hyper-parameters.

### D.3.1 Supervised Learning

All the experiments were run under Ubuntu 18.04 LTS, on a single GPU per run (using two 8-GPU machines in total). Timing experiments were all run on Nvidia GeForce GTX 1080 Ti GPUs, with an Intel Xeon E5-2650 CPU. The remaining experiments were run on either Nvidia GeForce RTX 2080 Ti GPUs or Nvidia GeForce GTX 1080 Ti GPUs, respectively using an Intel Xeon Gold 6230 CPU or an Intel Xeon E5-2650 CPU.

#### D.3.1.1 MultiMNIST

For consistency with the experimental setup of Sener and Koltun (2018), we employ a modified encoder-decoder version of the LeNet architecture (LeCun et al., 1998). Specifically, the last layer is omitted from the encoder, and two fully-connected layers are employed as task-specific predictive heads. The cross-entropy loss is used for both tasks. All methods are trained for 100 epochs using Adam (Kingma and Ba, 2015) in the stochastic gradient setting, with an initial learning rate of $\eta = 10^{-2}$ (tuned in $\eta \in \{10^{-3}, 10^{-2}, 10^{-1}\}$ and yielding the best validation results for all considered algorithms), exponentially decayed by 0.95 after each epoch, and a mini-batch size of 256.

#### D.3.1.2 CelebA

As commonly done in previous work (Sener and Koltun, 2018; Yu et al., 2020; Liu et al., 2021c), we employ an encoder-decoder architecture where the encoder is a ResNet-18 (He et al., 2016) (without the final layer) with batch normalization layers (Ioffe and Szegedy, 2015), and the per-task decoders are linear classifiers. The cross-entropy loss is used for all tasks. The training is performed from scratch for

50 epochs using Adam, with a mini-batch size of 128 and a per-epoch exponential decay factor of 0.95. As common on this network-dataset combination (Lin et al., 2022; Chen et al., 2020), the initial learning rate is $\eta = 10^{-3}$ for all methods except for MGDA and IMTL, for which $\eta = 5 \times 10^{-4}$ yielded a better validation performance. As done by the respective authors, for PCGrad, RLW and GradDrop we use the same learning rate as the unitary scalarization (Yu et al., 2020; Lin et al., 2022; Chen et al., 2020).

#### D.3.1.3 Cityscapes

Consistently with recent work (Lin et al., 2022), we rely on a dilated ResNet-50 architecture pre-trained on ImageNet (Yu et al., 2017) for the encoder, and on the Atrous Spatial Pyramid Pooling (Chen et al., 2018), which internally uses batch normalization, as decoders. While more powerful encoders might lead to better performance on Cityscapes, like the SegNet (Badrinarayanan et al., 2017) used in (Javaloy and Valera, 2022; Liu et al., 2021a; Navon et al., 2022), we aim to provide a fair comparison of MTL optimizers, rather than maximize overall task performance. Cross-entropy loss is employed on the semantic segmentation task, whereas the $\ell_1$ loss is used for the depth estimation. The training is performed by using Adam with a mini-batch size of 32 for 100 epochs, with an initial step size $\eta = 5 \times 10^{-4}$ resulting in the best validation performance for all algorithms, exponentially decayed by 0.95 at each epoch.

### D.3.2 Reinforcement Learning

Similarly to the supervised learning experiments, we ran all the experiments under Ubuntu 18.04 LTS using one GPU per run (using six 8-GPU machines in total). Timing experiments were all run using NVIDIA GeForce RTX 2080 Ti GPUs, with an Intel Xeon Gold 6230 CPU. The main bulk of the remaining experiments was run on Nvidia GeForce RTX 2080 Ti GPUs with either Intel Xeon Gold 6230 or Intel Xeon Silver 4216. We utilised NVIDIA GeForce RTX 3080 GPUs with Intel Xeon Gold 6230 CPUs for a small fraction of experiments.

**Table D.1:** Hyperparameters of the RL experiments. Hyperparameters different from Sodhani et al. (2021) are in bold.

| **Hyperparameter** | **Value** |
|---|---|
| All methods | |
| – training steps | 2,000,000 |
| – batch size | 1280 |
| – **Replay buffer size** | **4,000,000** |
| – actor learning rate | 0.0003 |
| – critic learning rate | 0.0003 |
| – entropy $\alpha$ learning rate | 0.0003 |
| – shared entropy $\alpha$ | True |
| – runs | 10 |
| – discounting $\gamma$ | 0.99 |
| Unit. Scal. | |
| – **actor $l_2$ coeff.** | **0.0003** |
| PCGrad | |
| – **actor $l_2$ coeff.** | **0.0001** |
| RLW Norm. | |
| – normal mean | 0 |
| – normal std | 1 |
| – actor $l_2$ coeff. | 0.0003 |
| RLW Diri. | |
| – $\alpha$ | 1 |
| – actor $l_2$ coeff. | 0.0003 |
| GradDrop | |
| – k | 1 |
| – p | 0.5 |
| – actor $l_2$ coeff. | 0.0001 |
| MGDA | |
| – gradient normalization | $L_2$ |
| – actor $l_2$ coeff. | 0.0 |
| IMTL | |
| – actor learning rate | 0.00003 |
| – critic learning rate | 0.00003 |
| – entropy $\alpha$ learning rate | 0.00003 |
| – actor $l_2$ coeff. | 0.0 |

We use Sodhani et al. (2021) for most of the hyperparameters and list them in Table D.1. We use bold font where we use a hyperparameter different from Sodhani et al. (2021). Similarly to Sodhani et al. (2021), we use the v1 version of Metaworld for our experiments[1]. Sodhani et al. (2021) use a shared entropy loss weight $\alpha$ for PCGrad and separate $\alpha$ for unitary scalarization[2]. In our experiments, use shared $\alpha$ for all of the methods for fairness. Since it is a single number (rather than a vector), we used unitary scalarization to update $\alpha$ for all SMTOs apart from PCGrad which was already implemented in (Sodhani et al., 2021).

We use the same network architecture as in Sodhani et al. (2021), i.e. a three-layered feedforward fully-connected network with 400 hidden units per layer for both, the actor and the critic. The actor is shared across all tasks as well as the critic. To normalize rewards, we keep track of first and second moments in the buffer and normalise the rewards by their standard deviation: $r'_i = {}^{r_i}/_{\hat{\sigma}_i}$, where $\hat{\sigma}_i$ is the sample standard deviation of the rewards for environment $i$. Sodhani et al. (2021) average the gradient for unitary scalarisation and PCGrad, whereas our SMTO implementations sum the gradients, i.e. effectively using larger learning rates (apart from MGDA that assures that all the aggregation weights sum to 1). We tried reducing the learning

[1] https://github.com/rlworkgroup/metaworld.git@af8417bfc82a3e249b4b02156518d775f29eb289

[2] https://mtrl.readthedocs.io/en/latest/pages/tutorials/baseline.html

rate for SMTOs that sum (RLW Norm., RLW Diri., and GradDrop) both for MT10 and MT50, but it worked worse for these methods and we kept the default learning rate for them as well. We had to use a smaller learning rate for IMTL, because with the default one it crashed at the beginning of training due to numerical overflow. Smaller learning rate did not prevent it from crashing, but this happened much later.

We tried $10^6$, $2 \times 10^6$, and $4 \times 10^6$ for the replay buffer size with the last being superior in terms of stability. Additionally, for $l_2$ actor regularization, we tried 0.0001 and 0.0003 with the latter being slightly superior for the baseline. We tried the same options for other SMTOs, and picked the best option for each of the method. For MGDA, no regularisation works best, most likely due to a strong regularization effect of the method itself, which is mirrored by our supervised learning results. PCGrad and Graddrop work best with the regularization coefficient of 0.0001. Both RLW variants use the same coefficient as the baseline (0.0003).

For MT50, we took the best MT10 hyperparameters, and we believe one could obtain even better results for unitary scalarisation since it is much faster to tune compared to other SMTOs (e.g. 15 hours for unitary scalarisation vs 9 days for PCGrad).

### D.3.3 Software Acknowledgments and Licenses

Our codebase is built upon several prior works: (Sener and Koltun, 2018), Liu et al. (2019), (Lin et al., 2022) and (Sodhani et al., 2021): all of them were released under a MIT license. We also acknowledge Tseng (2020), upon which we built some of our code. Multi-MNIST is based on MNIST dataset that is released under Creative Commons Attribution-Share Alike 3.0 license. The code for generating Multi-MNIST dataset was taken from Sener and Koltun (2018) released under MIT license. CelebA dataset has a custom license allowing non-commercial research purposes. More details can be found on the project website: `http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`. Cityscapes also has a custom license allowing non-commercial research purposes. The full text of the

license can be found on the project website:`https://www.cityscapes-dataset.com/license/`. Metaworld, used for RL experiments is released under MIT license.

## D.4 Supplementary Supervised Learning Experiments

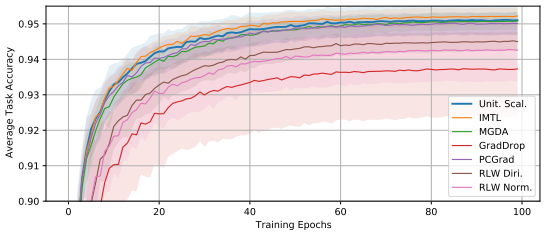
This section presents supervised learning results omitted from §5.4.1. In particular, we show additional plots for the experiments of §5.4.1, then present an analysis of the regularising effect of SMTOs in the absence of single-task regularization (§D.4.2), and conclude with an ablation study on GradDrop's dependency on the sign of per-task gradients (§D.4.3).

### D.4.1 Addendum

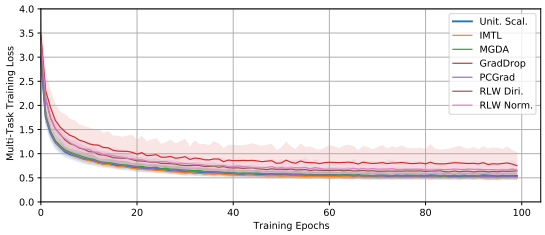
This section complements the plots presented in §5.4.1. In particular, we show the test and runtime results in table form, along with the behavior of the validation metrics and of the training loss over the training epochs. Plots for Multi-MNIST, CelebA, and Cityscapes are reported in Figures D.1, D.2 and D.10, respectively. The behavior of the CelebA training loss demonstrates heavier regularization (compare with the unregularized plot in Figure D.3(a)). Except IMTL and MGDA, for which the tuned values of the weight decay prevent overfitting, the other optimizers display very similar validation and training curves, and start overfitting around epoch 30. Considering that most SMTOs required less regularization (see §5.4.1.2), the results are consistent with our interpretation of SMTOs as regularizers in §5.5. The Cityscapes plots display a certain instability across training epochs, as demonstrated by the various peaks and valleys in the metrics. Nevertheless, in spite of a factor 10 difference in scale, both training losses are similarly decreased by most optimizers.

### D.4.2 Unregularized Experiments

Figures 5.5, D.3(a) and D.3(b) respectively report the average task validation accuracy, the multi-task training loss, and the multi-task validation loss at each training epoch. The regularizing effect of SMTOs compared to unitary scalarization is shown by: (i) the delay of the onset of overfitting on the validation data




**(a)** Mean (and 95% CI) average task validation accuracy per training epoch.

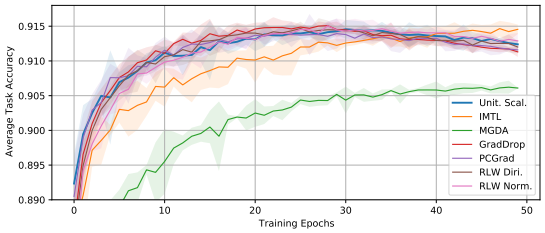



**(b)** Mean (and 95% CI) training multi-task loss $\mathcal{L}^{\text{MT}}$ per epoch.

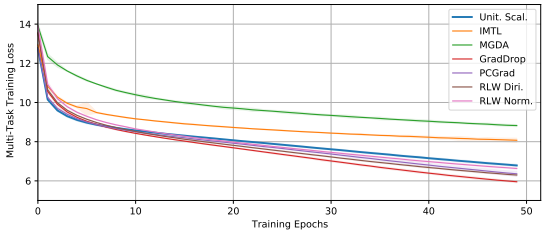
| MTO | Average Task Accuracy | Epoch Runtime [s] |
|---|---|---|
| Unit. Scal. | 9.476e-01 ± 4.368e-03 | [3.510e+00, 3.617e+00] |
| IMTL | 9.487e-01 ± 2.533e-03 | [3.695e+00, 3.996e+00] |
| MGDA | 9.478e-01 ± 1.977e-03 | [3.491e+00, 3.617e+00] |
| GradDrop | 9.347e-01 ± 1.282e-02 | [3.508e+00, 3.589e+00] |
| PCGrad | 9.479e-01 ± 3.578e-03 | [3.807e+00, 3.928e+00] |
| RLW Diri. | 9.430e-01 ± 2.973e-03 | [3.790e+00, 4.005e+00] |
| RLW Norm. | 9.399e-01 ± 8.929e-03 | [3.894e+00, 4.225e+00] |

**(c)** Mean and 95% CI of the avg. task test accuracy across runs, and interquartile range for the training time per epoch.

**Figure D.1:** Additional figures for the comparison of various SMTOs with the unitary scalarization on the MultiMNIST dataset (Sener and Koltun, 2018).




**(a)** Mean (and 95% CI) average task validation accuracy per training epoch.




**(b)** Mean (and 95% CI) training multi-task loss $\mathcal{L}^{\text{MT}}$ per epoch.

| MTO | Average Task Accuracy | Epoch Runtime [s] |
|---|---|---|
| Unit. Scal. | 9.090e-01 ± 7.568e-04 | [2.869e+02, 2.878e+02] |
| IMTL | 9.093e-01 ± 7.631e-04 | [3.600e+02, 3.621e+02] |
| MGDA | 9.022e-01 ± 9.687e-04 | [6.859e+02, 7.194e+02] |
| GradDrop | 9.098e-01 ± 3.383e-04 | [3.001e+02, 3.008e+02] |
| PCGrad | 9.093e-01 ± 1.108e-03 | [1.015e+04, 1.016e+04] |
| RLW Diri. | 9.099e-01 ± 7.845e-04 | [3.040e+02, 3.054e+02] |
| RLW Norm. | 9.095e-01 ± 1.012e-03 | [3.028e+02, 3.037e+02] |

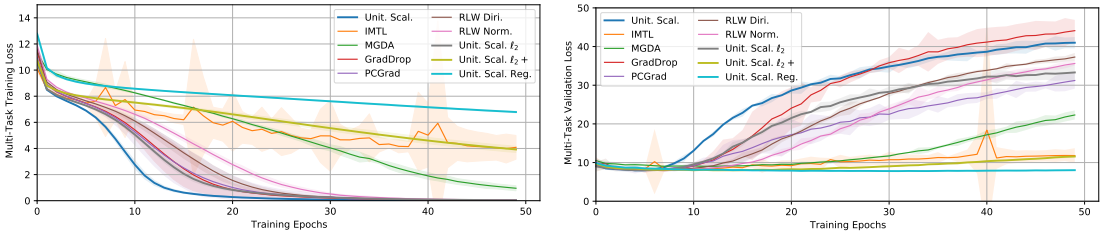
**(c)** Mean and 95% CI of the avg. task test accuracy across runs, and interquartile range for the training time per epoch.

**Figure D.2:** Additional figures for the comparison of various SMTOs with the unitary scalarization on the CelebA (Liu et al., 2015) dataset.

in figure 5.5, (ii) the reduction of the convergence rate on the training loss in figure D.3(a) (compare with figure D.2(b)), and (iii) the fact that validation and training losses remain positively correlated for larger numbers of epochs. In fact, the behavior of both the training and validation loss for the SMTOs closely parallels that of $\ell_2$-regularized unitary scalarization, with differing degrees of regularization. We further note that unregularized IMTL displays a certain instability (compare

with the regularized version in figure D.2(a)).

The addition of dropout layers further reduces overfitting, improves stability (reduced confidence intervals) and pushes the average validation curve upwards, motivating its use on all optimizers for the experiments of §5.4.1.2. Nevertheless, confidence intervals in Figure 5.5 still overlap due to the instability of the unregularized unitary scalarization. Figure D.5 provides a more detailed comparison over 20 repetitions, confirming that the combined use of dropout layers and $\ell_2$ regularization improves average performance and reduces the empirical variance for unitary scalarization. Furthermore, Figure D.4 shows that regularization improves the peak average validation performance for all algorithms, demonstrating the need of tuning $\lambda$ also for SMTOs. We conclude by pointing out that even without regularization, when carefully tuned, the maximal performance over epochs of unitary scalarization is comparable to SMTOs in Figure 5.5.




**(a)** Mean and 95% CI (3 runs) multi-task training loss per epoch.

**(b)** Mean and 95% CI (3 runs) multi-task validation loss per training epoch.

**Figure D.3:** Additional figures for the unregularized comparison of various SMTOs with the unitary scalarization on CelebA. SMTOs provide varying degrees of regularization.

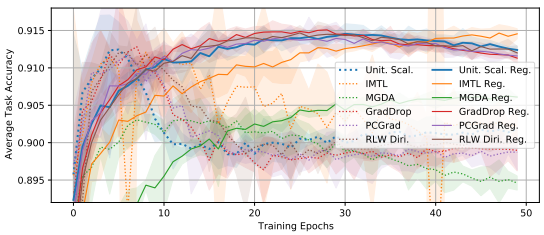



**Figure D.4:** Effect of regularization (dropout layers and weight decay) on the average task validation accuracy for all considered optimizers on the CelebA dataset: regularization improves the average performance of all algorithms.

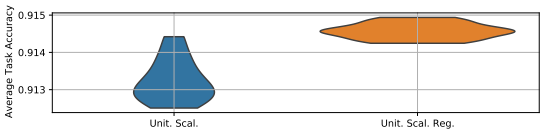



**Figure D.5:** Effect of regularization (dropout layers and weight decay) on unitary scalarization on the CelebA dataset: violin plots (20 runs) for the best avg. task validation accuracy over epochs. The width at a given value represents the proportion of runs yielding that result. Regularization improves the average performance while decreasing its variability.

### D.4.3 Sign-Agnostic GradDrop

We will now present an ablation study on GradDrop, investigating the effect of the sign of per-task gradients on the SMTO's performance. Specifically, we compare the performance of GradDrop with a sign-agnostic version of its stochastic gradient masking (which we refer to as "Sign-Agnostic GradDrop"), whose update direction is defined as follows:

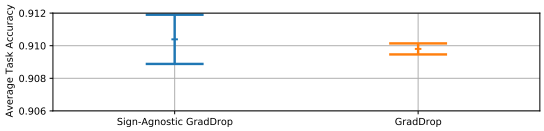
$$\mathbf{g} = -\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}\right)^{T} \left(\sum_{i \in \mathcal{T}} \mathbf{u}_i \odot \nabla_{\mathbf{z}} \mathcal{L}_i\right),$$

where $\mathbf{u}_i, \nabla_{\mathbf{z}} \mathcal{L}_i \in \mathbb{R}^{n \times r}$ and, for all $i \in \mathcal{T}$, $\mathbf{u}_i$ is i.i.d. according to $\mathbf{u}_i[j,k] \sim \text{Bernoulli}(p)$ $\forall j \in \{1,\dots,n\}, k \in \{1,\dots,r\}$. Differently from a similar study carried out by Chen et al. (2020), we tuned the hyper-parameter of the sign-agnostic masking in the following range: $p \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$.

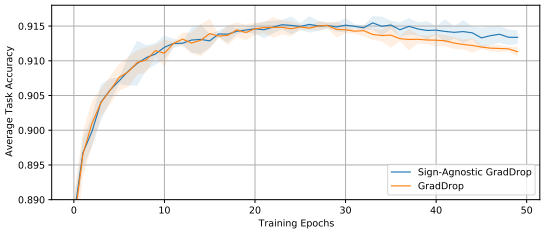
The experimental setup complies with the one described in appendix D.3.1. Figure D.6, plotting test and validation results for the CelebA dataset (Liu et al., 2015), shows that the performance of Sign-Agnostic GradDrop closely matches the original algorithm. Therefore, sign conflicts across per-task gradients do not seem to play a significant role in GradDrop's performance.

## D.5 Supplementary Reinforcement Learning Experiments

This section presents additional results pertaining to the RL experiments in §5.4.2.

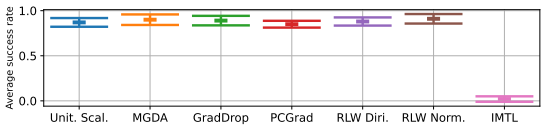



**(a)** Mean and 95% CI (3 runs) avg. task test accuracy.

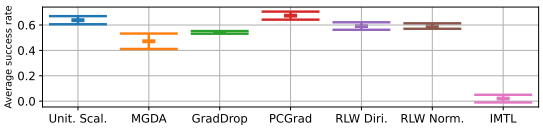



**(b)** Mean and 95% CI (3 runs) avg. task validation accuracy per training epoch.

**Figure D.6:** Comparison of GradDrop (Chen et al., 2020) with sign-agnostic masking of the shared-representation gradients on the CelebA dataset (Liu et al., 2015). No statistically relevant difference between the two methods can be observed for the majority of the epochs.




**(a)** MT10 (10 runs per method).

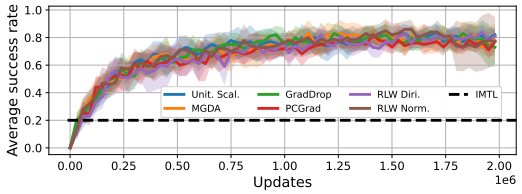



**(b)** MT50 (10 runs per method).

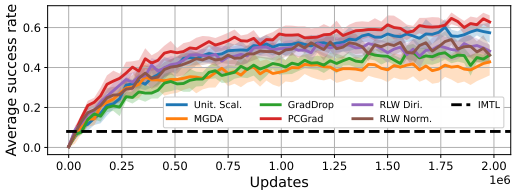
**Figure D.7:** Mean and 95% CI for the best avg. success rate on Metaworld. None of the SMTOs significantly outperforms unitary scalarization.

### D.5.1 Addendum

Figure D.7 re-plots Figure 5.4(a) and 5.4(b) with the omitted IMTL results, while Figure D.8 shows the learning curves omitted from §5.4.2. As pointed out in §5.4.2, none of the IMTL runs successfully terminated due to numerical instability. Indeed, Liu et al. (2021c) show that, in supervised settings, coefficients do not fluctuate much across epochs (Liu et al., 2021c, Figure 4, appendix B) and never become negative. By contrast, up to 50% of the scaling coefficients $\alpha$ are negative in our experiments, thus reversing subtask gradient directions. MGDA, which constrains the weights, is more stable and is comparable to unitary scalarization. In order to avoid incomplete curves and unfair calculations of the mean, Figure D.8 plots the highest value ever achieved by *any* seed as a dashed horizontal line. The IMTL results in Figure D.7, instead, report the best average success rate of each seed until its termination.

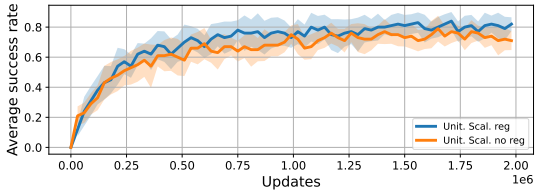



**(a)** MT10 (10 points per method).

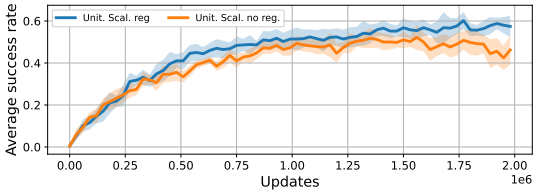



**(b)** MT50 (10 points per method).

**Figure D.8:** Mean and 95% CI for the avg. success rate on Metaworld. None of the SMTOs significantly outperforms unitary scalarization.




**(a)** MT10 average performance (10 runs) and 95% CI.




**(b)** MT50 average performance (5 runs) and 95% CI.

**Figure D.9:** For both MT10 and MT50, actor $l_2$ regularization pushes the average higher for unitary scalarization.

### D.5.2 Ablation studies

Figure D.12 presents our ablations for MT10 experiments. Due to computational constraints, we ran ablations on the unitary scalarization and PCGrad since these are the two methods previously tested in the RL setting.

Figure D.9 shows ablation studies on the effect of regularization on MT10 and MT50. In spite of CI overlaps, actor $l_2$ regularization pushes the average higher on both benchmarks, motivating our use of regularization for the experiments in §5.4.2. Furthermore, the gap between the averages tends to widen with the number of updates on MT50, suggesting improved stabilization.

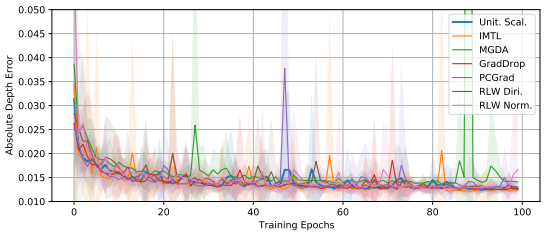
### D.5.3 Sensitivity to Reward Normalization

Figure D.11 shows that multitask agent performance is highly sensitive to the reward normalization moving average hyperparameter[3] motivating our buffer normalization in Section 5.4.2.

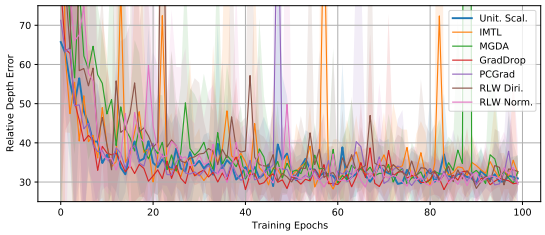
[3] https://github.com/facebookresearch/mtenv/blob/4a6d9d6fdfb321f1b51f890ef36b5161359e972d/mtenv/envs/metaworld/wrappers/normalized_env.py#L69

**(a)** Mean and 95% CI of the test metrics across runs, and interquartile range for the training time per epoch.

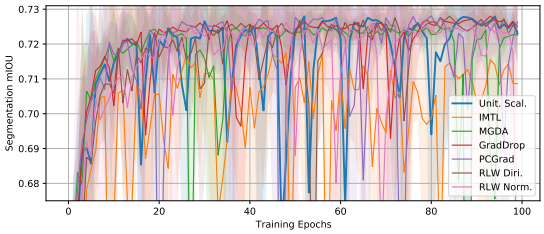
| MTO | Absolute Depth Error | Relative Depth Error | Segmentation Accuracy | Segmentation mIOU | Epoch Runtime [s] |
|---|---|---|---|---|---|
| Unit. Scal. | 1.301e-02 ± 2.342e-04 | 4.761e+01 ± 5.148e+00 | 9.196e-01 ± 2.913e-04 | 7.012e-01 ± 6.001e-04 | [3.228e+02, 3.241e+02] |
| IMTL | 1.281e-02 ± 7.521e-04 | 4.389e+01 ± 6.984e-01 | 9.164e-01 ± 2.828e-03 | 6.967e-01 ± 4.785e-03 | [7.329e+02, 7.373e+02] |
| MGDA | 1.418e-02 ± 2.331e-04 | 4.750e+01 ± 1.466e+01 | 9.189e-01 ± 2.636e-04 | 6.999e-01 ± 3.124e-03 | [7.251e+02, 7.269e+02] |
| GradDrop | 1.293e-02 ± 2.757e-04 | 4.674e+01 ± 7.709e+00 | 9.193e-01 ± 1.282e-03 | 7.024e-01 ± 3.628e-03 | [5.196e+02, 5.215e+02] |
| PCGrad | 1.294e-02 ± 2.284e-04 | 4.380e+01 ± 5.165e+00 | 9.198e-01 ± 9.119e-04 | 7.025e-01 ± 6.531e-04 | [4.202e+02, 4.212e+02] |
| RLW Diri. | 1.305e-02 ± 4.155e-04 | 4.810e+01 ± 2.259e+00 | 9.199e-01 ± 1.247e-03 | 7.037e-01 ± 1.989e-03 | [3.161e+02, 3.164e+02] |
| RLW Norm. | 1.301e-02 ± 5.528e-04 | 4.630e+01 ± 2.751e+00 | 9.192e-01 ± 4.962e-04 | 7.006e-01 ± 4.580e-03 | [3.194e+02, 3.210e+02] |



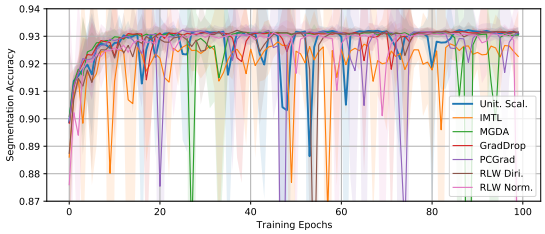

**(b)** Mean (and 95% CI) absolute depth validation error per training epoch.



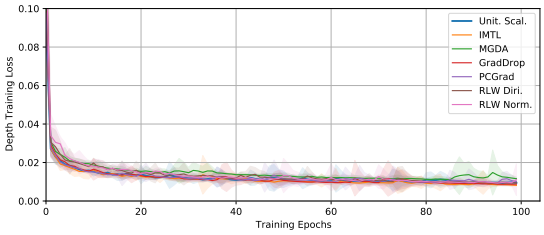

**(c)** Mean (and 95% CI) relative depth validation error per training epoch.



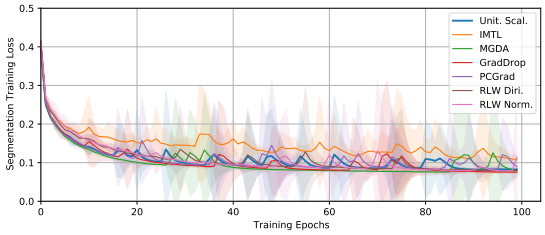

**(d)** Mean (and 95% CI) validation segmentation mIOU per training epoch.




**(e)** Mean (and 95% CI) validation segmentation accuracy per training epoch.




**(f)** Mean (and 95% CI) training depth loss per epoch.




**(g)** Mean (and 95% CI) training segmentation loss per epoch.

**Figure D.10:** Additional figures for the comparison of SMTOs with the unitary scalarization on the Cityscapes (Cordts et al., 2016) dataset.

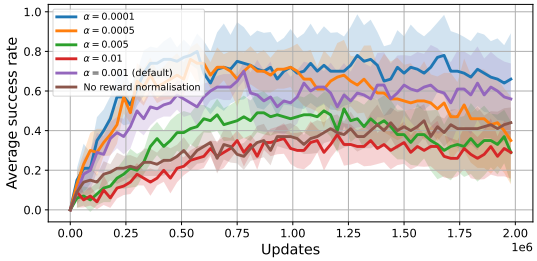



**Figure D.11:** The learning outcomes of a Multitask SAC agent vary considerably depending on the reward normalisation hyperparameter. Each of the curves represents and average of 10 runs with shaded 95% confidence interval.

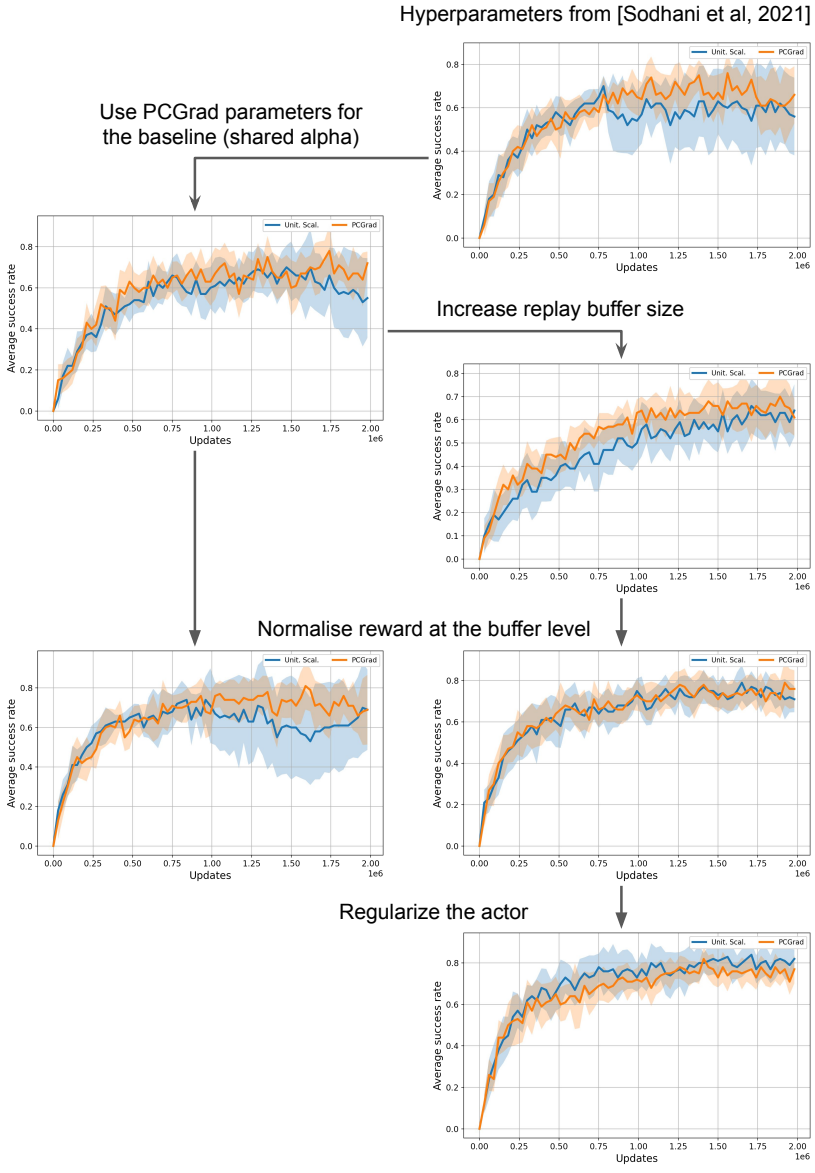



**Figure D.12:** Metaworld's MT10 ablation experiments.

# Bibliography


Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.

Tobias Achterberg and Roland Wunderling. *Mixed Integer Programming: Analyzing 12 Years of Progress*. Springer Berlin Heidelberg, 2013.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, 2019.

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

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

Plamen Angelov and Eduardo Soares. Towards explainable deep neural networks (xdnn). *Neural Networks*, 2020.

Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. *International Conference on Machine Learning*, 2018.

Francis Bach. Duality between subgradient and conditional gradient methods. *SIAM Journal on Optimization*, 2015.

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. Improved geometric path enumeration for verifying relu neural networks. In *Computer Aided Verification*. Springer International Publishing, 2020.

Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 2003.

Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. *International Conference on Learning Representations*, 2020.

Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989.

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.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX median scaling issues, 2020. URL `https://github.com/google/jax/issues/4379`.

Marc Brockschmidt. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning*, 2020.

Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. A unified view of piecewise linear neural network verification. *Neural Information Processing Systems*, 2018.

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.

Emmanuel J. Candès, Michael B. Wakin, and Stephen P. Boyd. Enhancing sparsity by reweighted $\ell$ 1 minimization. *Journal of Fourier Analysis and Applications*, 2008.

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. In *International Joint Conference on Artificial Intelligence*, 2021.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy*, 2017.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997a.

Rich Caruana. *Multitask learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1997b.

Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Neural Information Processing Systems*, 2000.

Jinghui Chen and Quanquan Gu. Padam: Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint*, 2018.

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision*, 2018.

Shijie Chen, Yu Zhang, and Qiang Yang. Multi-task learning in natural language processing: An overview. *arXiv preprint arXiv:2109.09138*, 2021.

Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. In *Neural Information Processing Systems*, 2020.

Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *International Conference on Machine Learning*, 2019.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *International Conference on Machine Learning*, 2008.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Conference on Computer Vision and Pattern Recognition*, 2016.

Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. *International Conference on Learning Representations*, 2021a.

Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with sparse dual algorithms. *arXiv preprint arXiv:2101.05844*, 2021b.

Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*, 2021c.

Alessandro De Palma, Rudy Bunel, Krishnamurthy Dvijotham, M. Pawan Kumar, and Robert Stanforth. IBP regularization for verified adversarial robustness via branch-and-bound. In *ICML 2022 Workshop on Formal Verification of Machine Learning*, 2022.

J. Désidéri. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus Mathematique*, 350:313–318, 2012.

Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Computing Surveys*, page 326–327, sep 1995.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

Mengnan Du, Fan Yang, Na Zou, and Xia Hu. Fairness in deep learning: A computational perspective. *IEEE Intelligent Systems*, 2020.

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arxiv:1805.10265*, 2018a.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *Uncertainty in Artificial Intelligence*, 2018b.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. *Uncertainty in Artificial Intelligence*, 2020.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *Automated Technology for Verification and Analysis*, 2017.

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 2022.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. *International Conference on Learning Representations*, 2022.

Jörg Fliege and Benar Fux Svaiter. Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 2000.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 1956.

Linda Geddes. DeepMind uncovers structure of 200m proteins in scientific leap forward. *The Guardian*, 2022. URL `https://www.theguardian.com/technology/2022/jul/28/deepmind-uncovers-structure-of-200m-proteins-in-scientific-leap-forward`.

Gauthier Gidel, Tony Jebara, and Simon Lacoste-Julien. Frank-Wolfe algorithms for saddle point problems. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

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, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *SECML NeurIPS*, 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. *Workshop on Security in Machine Learning, NeurIPS*, 2018b.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

Monique Guignard and Siwhan Kim. Lagrangean decomposition: A model yielding stronger lagrangean bounds. *Mathematical programming*, 1987.

Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *International Conference on Learning Representations*, 2018.

Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. 2020.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

Haoyu He, Tianhao Wei, Huan Zhang, Changliu Liu, and Cheng Tan. Characterizing neural network verification for systems with NN4SYSBENCH. In *ICML 2022 Workshop on Formal Verification of Machine Learning*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*, 2016.

P. Henriksen and A. Lomuscio. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21)*, 2021.

Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI20)*, 2020.

Tom Heskes. Empirical bayes for learning to learn. In *International Conference on Machine Learning*, 2000.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *AAAI Conference on Artificial Intelligence*, 2019.

Timothy Hickey, Qun Ju, and Maarten H Van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM (JACM)*, 2001.

Timothy M. Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J. Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, 2020.

Yujia Huang, Huan Zhang, Yuanyuan Shi, J Zico Kolter, and Anima Anandkumar. Training certifiably robust neural networks with efficient local lipschitz bounds. In *Neural Information Processing Systems*, 2021.

Matthew Hutson. Has artificial intelligence become alchemy? *Science*, 2018.

Sandro Iannaccone. DeepMind, l'IA di Alphabet, ha pubblicato la struttura 3D di (quasi) tutte le proteine esistenti: perché è importante. *la Repubblica*, 2022. URL `https://www.repubblica.it/tecnologia/2022/07/30/news/deepmind_lai_di_alphabet_ha_appena_pubblicato_la_struttura_di_quasi_tutte_le_proteine_esistenti-359685079/`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, 2017.

Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. *International Conference on Machine Learning*, 2013.

Adrián Javaloy and Isabel Valera. Rotograd: Gradient homogenization in multitask learning. In *International Conference on Learning Representations*, 2022.

R. G. Jeroslow. Representability in mixed integer programming, i: Characterization results. *Discrete Applied Mathematics*, 1987.

Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology*, 2017.

Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *IEEE/AIAA Digital Avionics Systems Conference*, 2016.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

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.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, 2019.

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations*, 2017.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? In *International Conference on Machine Learning*, 2018.

Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

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. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 2012.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *International Conference on Learning Representations*, 2017.

Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can q-learning with graph networks learn a generalizable branching heuristic for a SAT solver? In *Neural Information Processing Systems*, 2020.

Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. In *International Conference on Learning Representations*, 2021.

Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M. Pawan Kumar. In defense of the unitary scalarization for deep multi-task learning. In *Neural Information Processing Systems*, 2022.

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *International Conference on Machine Learning*, 2013.

David Larousserie. L'intelligence artificielle, nouveau moteur de la recherche scientifique. *Le Monde*, 2022. URL `https://www.lemonde.fr/sciences/article/2022/10/24/l-intelligence-artificielle-nouveau-moteur-de-la-recherche-scientifique_6147154_1650684.html`.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

Claude Lemaréchal. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer, 2001.

Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2020.

Ping Li, Trevor J. Hastie, and Kenneth W. Church. Nonlinear estimators and tail bounds for dimension reduction in l1 using cauchy random projections. Conference on Learning Theory, 2007.

Baijiong Lin, Feiyang Ye, and Yu Zhang. A closer look at loss weighting in multi-task learning. In *arXiv preprint arXiv:2111.10603*, 2022.

Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. Catalyst acceleration for first-order convex optimization: From theory to practice. *Journal of Machine Learning Research*, January 2017.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 2021a.

Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. Algorithms for verifying deep neural networks. *arXiv:1903.06758*, 2019.

Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 2021b.

Liyang Liu, Yi Li, Zhanghui Kuang, Jing-Hao Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. In *International Conference on Learning Representations*, 2021c.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *International Conference on Computer Vision*, 2015.

Jingyue Lu and M Pawan Kumar. Neural network branching for neural network verification. In *International Conference on Learning Representations*, 2020.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? A large-scale study. In *Neural Information Processing Systems*, 2018.

Zhaoyang Lyu, Minghao Guo, Tong Wu, Guodong Xu, Kehuan Zhang, and Dahua Lin. Towards evaluating and training verifiably robust neural networks. *Conference on Computer Vision and Pattern Recognition*, 2021.

Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *International Conference on Learning Representations*, 2018.

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.

Lalit Maganti. Verification of feed-forward relu neural networks. Master's thesis, Imperial College London, 2017.

Chengzhi Mao, Amogh Gupta, Vikram Nitin, Baishakhi Ray, Shuran Song, Junfeng Yang, and Carl Vondrick. Multitask learning strengthens adversarial robustness. In *European Conference on Computer Vision*, 2020.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. *International Conference on Machine Learning*, 2018.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Conference on Computer Vision and Pattern Recognition*, 2016.

David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19, 2016.

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Prima: General and precise neural network certification via scalable convex hull approximations. *Proceedings of the ACM on Programming Languages*, 2022.

Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Févry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 2020.

Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. In *International Conference on Machine Learning*, 2022.

Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and information conference*. Springer, 2019.

Emilio Parisotto, Lei Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2016.

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.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Neural Information Processing Systems*, 2019.

Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *Neural Information Processing Systems*, 2018.

Vicenç Rúbies Royo, Roberto Calandra, Dusan M. Stipanovic, and Claire J. Tomlin. Fast neural network verification via shadow prices. *arXiv preprint arXiv:1902.07247*, 2019.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2016.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. *Dynamic Routing between Capsules*. 2017.

Hadi Salman, Greg Yang, Jerry Li, Pengchuan Zhang, Huan Zhang, Ilya Razenshteyn, and Sébastien Bubeck. Provably robust deep learning via adversarially trained smoothed classifiers. In *Neural Information Processing Systems*, 2019a.

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*, 2019b.

Saverio Salzo and Silvia Villa. Inexact and accelerated proximal point algorithms. *Journal of Convex Analysis*, 19:1167–1192, 01 2012.

David Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. Winner's curse? on pace, progress, and empirical rigor. In *International Conference on Learning Representations, workshop track*, 2018.

Michael L. Seltzer and Jasha Droppo. Multi-task learning in deep neural networks for improved phoneme recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Neural Information Processing Systems*, 2018.

Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 2020.

François Serre, Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling polyhedral neural network verification on GPUs. In *Machine Learning and Systems (MLSys)*, 2021.

Hanif D Sherali and Warren P Adams. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero—one programming problems. *Discrete Applied Mathematics*, 1994.

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.

Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Fast certified robust training with short warmup. In *Neural Information Processing Systems*, 2021.

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, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. *International Conference on Learning Representations*, 2019c.

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. URL `https://github.com/eth-sri/eran`.

Gaurav Singh and John Shawe-Taylor. Faster convergence & generalization in DNNs. *arXiv preprint*, 2018.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In Marina Meila and Tong Zhang, editors, *International Conference on Machine Learning*, 2021.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

Teruo Sunaga. Theory of an interval algebra and its application to numerical analysis. *RAAG Memoirs*, 1958.

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.

Yee Whye Teh, Victor Bapst, Wojciech M. Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Neural Information Processing Systems*, 2017a.

Yee Whye Teh, Victor Bapst, Wojciech M. Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Neural Information Processing Systems*, 2017b.

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.

Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis for deep neural networks. *International Symposium on Formal Methods*, 2019.

Wei-Cheng Tseng. Weichengtseng/pytorch-pcgrad, 2020. URL `https://github.com/WeiChengTseng/Pytorch-PCGrad.git`.

Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. *International Conference on Machine Learning*, 2018.

Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. *Neural Information Processing Systems*, 2016.

Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Kush R. Varshney. *Trustworthy Machine Learning*. Independently Published, Chappaqua, NY, USA, 2022.

VNN-COMP. International verification of neural networks competition (VNN-COMP). *Verification of Neural Networks workshop at the International Conference on Computer-Aided Verification*, 2020. URL `https://sites.google.com/view/vnn20/vnncomp`.

Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018a.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Neural Information Processing Systems*, 2018b.

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. *Neural Information Processing Systems*, 2021a.

Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. In *International Conference on Learning Representations*, 2021b.

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.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *International Conference on Machine Learning*, 2018.

Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. Scaling provable adversarial defenses. *Neural Information Processing Systems*, 2018.

Bichen Wu, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

Kai Y Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. *International Conference on Learning Representations*, 2019.

Derrick Xin, Behrooz Ghorbani, Ankush Garg, Orhan Firat, and Justin Gilmer. Do current multi-task optimization methods in deep learning even help? In *Neural Information Processing Systems*, 2022.

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. In *Neural Information Processing Systems*, 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. In *International Conference on Learning Representations*, 2021.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence generative adversarial nets with policy gradient. *AAAI Conference on Artificial Intelligence*, 2017.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *3rd Annual Conference on Robot Learning*, 2019.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Neural Information Processing Systems*, 2020.

Bohang Zhang, Tianle Cai, Zhou Lu, Di He, and Liwei Wang. Towards certifying l-infinity robustness using neural networks with l-inf-dist neurons. In *International Conference on Machine Learning*, 2021.

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.

Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *International Conference on Learning Representations*, 2020.