SPEEDING UP NEURAL NETWORK VERIFICATION VIA AUTOMATED ALGORITHM CONFIGURATION

Matthias König[1], **Holger H. Hoos**[1,2], **Jan N. van Rijn**[1]
[1] Leiden University, Leiden, The Netherlands
[2] University of British Columbia, Vancouver, Canada
{h.m.t.konig, j.n.van.rijn}@liacs.leidenuniv.nl, hh@liacs.nl

ABSTRACT

Despite their great success in recent years, neural networks have found to be vulnerable to adversarial attacks. These attacks are often based on slight perturbations of given inputs that cause them to be misclassified. Several methods have been proposed to formally prove robustness of a given network against such attacks. However, these methods typically give rise to high computational demands, which severely limit their scalability. Recent state-of-the-art approaches state the verification task as a minimisation problem, which is formulated and solved as a mixed integer linear programming (MIP) problem. We extend this approach by applying automated algorithm configuration techniques to the solver and, more specifically, construct a portfolio of MIP solver configurations optimised for the neural network verification task. Thereby, we achieve a substantial reduction in CPU time by a factor of 4.7 when verifying a neural network classifier on the MNIST dataset and, at the same time, reduce the number of timeouts by a factor of 1.42 compared to the state of the art.

1 INTRODUCTION

It is now well known that neural networks are vulnerable to *adversarial attacks* (Szegedy et al., 2014), in which a given input is transformed in such a way that it is misclassified by the network. In the case of image recognition tasks, the required perturbation can be so small that it remains virtually undetectable to the human eye. Various methods have been proposed to establish robustness of neural networks against adversarial attacks. Some of these methods perform *heuristic* attacks (Goodfellow et al., 2014; Kurakin et al., 2016; Carlini & Wagner, 2017); however, these do not draw a full picture of the network's robustness to adversarial attacks, as one defense mechanism might still be circumvented by another, possibly new class of attacks. In light of this, approaches have been developed to more thoroughly verify neural networks (Scheibler et al., 2015; Bastani et al., 2016; Ehlers, 2017; Katz et al., 2017; Dvijotham et al., 2018; Gehr et al., 2018; Xiang et al., 2018; Bunel et al., 2018; Tjeng et al., 2019). These *formal* verification methods are able to assess the robustness of a given network in a principled fashion, which means that they yield provable guarantees on certain properties of input-output combinations. However, this class of network verification methods tends to be computationally expensive, making it difficult to verify networks with a large number of units and/or on a large number of input samples.

Recent work by Tjeng et al. (2019) addresses this challenge and presents a verification method that, for the first time, was able to evaluate the robustness of larger neural networks on the full MNIST dataset. In their study, Tjeng et al. (2019) formulate the verification task as a minimisation problem, which is then solved through *mixed integer linear programming* (MIP). More specifically, the optimisation task is to apply a perturbation to the original sample that maximises model error, while staying close to the initial example, *i.e.*, keeping the distance at a minimum. Speed-ups are achieved by means of tight formulations for non-linearities and a pre-solving algorithm that reduces the number of binary variables, *i.e.*, the number of unstable nodes. While their verifier works quite efficiently on some of the MNIST classifiers considered in their study, for others, it requires very substantial amounts of CPU time. For example, some classifiers can be verified within less than 5 seconds per sample on average, while others take at least 20 times as long. For those classifiers, verification tends to become very costly, not only in terms of wall-clock time, but especially in terms of overall CPU time. In addition, a sizeable fraction of instances could not be solved within a rather generous time limit of 1 200 seconds per sample.

We seek to overcome this limitation by applying *automated algorithm configuration* techniques to MIP-based neural network verification. Specifically, we automatically construct a *parallel portfolio* of MIP solver configurations optimised for solving neural network verification problems. Hereby, focus shifts from problem formulation to solving, as several studies have demonstrated that automatically configuring a MIP solver can lead to substantial decreases in computational cost (Hutter et al., 2009; 2010; 2011; Lopez-Ibanez & Stützle, 2014). To the best of our knowledge, ours is the first study to pursue this direction. In brief, the main contributions of our work are as follows: (*i*) A framework for automatically constructing a parallel portfolio of MIP solver configurations optimised for neural network verification; (*ii*) promising results when applying this framework to the verification of an MNIST classifier designed for robustness (Raghunathan et al., 2018), achieving a substantial improvement in CPU time by an average factor of 4.7 over the state of the art on a *solvable* subset of instances from the MNIST dataset. This subset excludes all samples that cannot be solved by any of the approaches we consider. Beyond that, we attain a reduction in timeouts by a factor of 1.42 and a 1.29-fold reduction in the upper bound on adversarial error, as we are able to certify a statistically significantly larger fraction of the dataset than the previous state of the art.

2 Methods

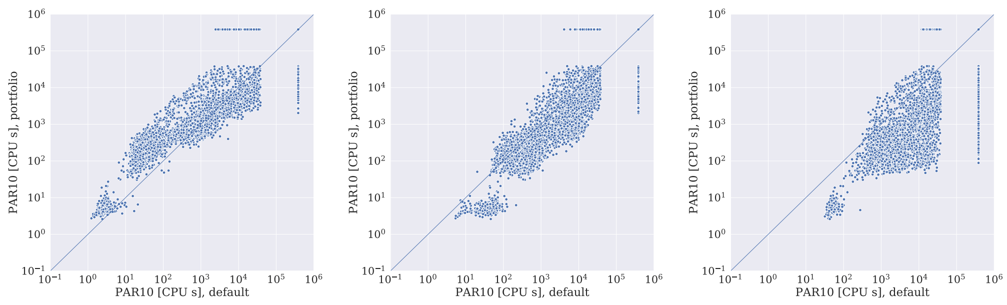
In order to reduce complexity, Tjeng et al. (2019) mainly focused on the problem formulation, but left the solver and its numerous parameters untouched. More specifically, they employed the commercial MIP solver Gurobi with default settings. This decision, along with their problem formulation, forms the starting point for our work.

Automated Algorithm Configuration. In general, the algorithm configuration problem can be described as follows: Given an algorithm A (also referred to as the *target algorithm*) with parameter configuration space Θ, a set of problem instances Π and a cost metric $c : \Theta \times \Pi \rightarrow \mathbb{R}$, find a configuration $\theta^* \in \Theta$ that minimises cost c across the instances in Π. In this study, we use SMAC (Hutter et al., 2011), a widely known, freely available, state-of-the-art configurator based on sequential model-based optimisation or Bayesian optimisation. The main idea of SMAC is to construct and iteratively update a statistical model of target algorithm performance (specifically: a random forest regressor; Breiman, 2001) to guide the search for good configurations.

Automatic Portfolio Construction. For the configuration procedure to work effectively, the problem instances of interest have to be sufficiently similar, such that a configuration that performs well on a subset of them also performs well on others. If a given instance set does not satisfy this homogeneity assumption, it is likely that automated configuration results in performance improvements on some instances, while performance on others might suffer, making it difficult to achieve overall performance improvements. When applying SMAC to our instance set, we observed precisely this behaviour. This problem can be addressed through *automatic portfolio construction* (Xu et al., 2010; Kadioglu et al., 2011; Malitsky et al., 2012; Lindauer et al., 2015). The general concept behind automatic portfolio construction techniques is to create a set of algorithm configurations that are chosen such that they complement each others strengths and weaknesses. This portfolio should then be able to exploit per-instance variation much more effectively than a single algorithm configuration, which is designed to achieve high performance overall, but may perform badly on certain types or subsets of instances. More specifically, we employ Hydra (Xu et al., 2010) to construct a parallel portfolio of different configurations of the Gurobi MIP solver. Leveraging standard multi-core CPU architectures, we run all configurations in this portfolio in parallel until one of them returns a solution or until an overall limit on CPU time is exceeded. We note that, in principle, automated algorithm selection (see, *e.g.*, Kotthoff, 2016) could be used to determine from this portfolio the configuration likely to solve any given instance most efficiently, though this requires substantial amounts of training data and creates uncertainty from sub-optimal choices made by the machine learning technique at the heart of such selection approaches.

3 Experimental Setup

In this section, we provide details on the configuration experiments presented in this study. More specifically, we describe the the MIP solver we configured, our benchmark set and details of the



(a) Cores = 1 (b) Cores = 4 (c) Cores = 32

Figure 1: Evaluation of our parallel portfolio approach on the MNIST dataset (n=10 000). Each dot represents a problem instance and the penalised running time for that instance achieved by the baseline approach (x-axis) *vs* our portfolio (y-axis). The baselines we considered are (a) the default solver running on 1 core, (b) the default solver running on 4 cores and (c) the default solver running on all available, *i.e.*, 32 cores, as in the work of Tjeng et al. (2019). Our parallel portfolio, using 4 cores, achieved substantially fewer timeouts than any of the baselines and lower CPU times (in terms of $\mathrm{PAR10}$ scores). Points grouped at the top and right border represent instances for which the solver reached the time limit, and are measured according to their penalised running time values.

configuration objective.

Benchmark Set. Our benchmark set is comprised of verification problem instances for the 10 000 samples of the MNIST dataset, using the network weights of the robust classifier $\mathrm{SDP_d} - \mathrm{MLP_A}$ from Raghunathan et al. (2018). Among the classifiers considered in the work of Tjeng et al. (2019), we regard this one as the most difficult to verify, since it shows the highest average solve time and optimality gaps for many examples.

MIP Solver. Following Tjeng et al. (2019), we used the Gurobi[1] MIP solver with a free academic license. This allows us to directly measure the impact of the configuration procedure on the performance of the verification method. Using the online documentation on Gurobi's parameters, we selected 62 performance-relevant parameters for configuration. To control and limit the computational resources given to the solver, we fixed the number of CPU cores to the minimum value of 1. The time limit for the solver was fixed at 9 600 seconds.

Configuration Objective. The objective of our configuration experiments is to minimise mean CPU time over all instances from the benchmark set. Generally, if the cost metric is running time, configurators typically optimise penalised average running time (PAR) as the metric of interest, which penalises unsuccessful runs by counting runs exceeding the cutoff time k as $p \times k$. In line with common practice in the algorithm configuration literature, we use $p = 10$ and refer to the cost metric as $\mathrm{PAR10}$.

Configuration Procedure. The parameters for the configuration procedure were set as follows. Hydra performed four independent runs of SMAC, where each run had a total time budget of 24 hours. After every run, Hydra adds one configuration to the portfolio, i.e., the configuration that yields the largest gain on overall portfolio performance. Thus, the final output is a portfolio of 4 solver configurations, which we ran in parallel on a given problem instance during evaluation.

4 Results

We report empirical results for our new approach and each baseline in the form of (*i*) the fraction of timeouts; and (*ii*) bounds on adversarial error (the fraction of the dataset for which a valid adversarial example can be found), complement to adversarial accuracy (the fraction of the dataset known to be robust); (*iii*) CPU time (*i.e.*, $\mathrm{PAR10}$ scores) on solvable instances, i.e., instances that were solved by our portfolio or any of the baselines within the given cutoff time. Aggregated performance numbers are presented in Table 1, whereas Figure 1 visualises penalised running time

[1]`https://www.gurobi.com`

Table 1: Timeouts, adversarial error and PAR10 scores for different solver configurations on the MNIST dataset. Note that all approaches were given the same budget in terms of CPU time (the number of cores times the cutoff time). Using our portfolio, we achieved better performance than the state-of-the-art (SOTA) method of Tjeng et al. (2019) as well as the default configuration of Gurobi using different numbers of cores. In concrete terms, we statistically significantly reduced the number of timeouts by factor 1.42 and certified 1.1 times as many examples, while decreasing the overall computational burden by a factor of 4.7 on solvable instances, *i.e.*, instances that could be solved by either the portfolio or any of the baseline approaches (*n*=8 646). Boldfaced values indicate statistically significant improvements according to a binomial test with $\alpha = 0.05$.

Configuration	**Cores**	**Cutoff**	**Timeouts**	**Adversarial Error**		**PAR10**
		[Seconds]		Lower Bound	Upper Bound	[CPU s]
Default [SOTA]	32	1 200	21.29%	14.37%	30.67%	39 772
Default	4	9 600	17.74%	14.40%	27.49%	22 065
Default	1	38 400	17.66%	14.36%	27.58%	20 117
Portfolio [Ours]	4	9 600	**14.96%**	14.43%	**23.86%**	**8 478**

of our portfolio approach against the baselines on an instance level.

Baselines. Firstly, we evaluated our portfolio approach against Gurobi as used by Tjeng et al. (2019), using all 32 cores per CPU available on our compute cluster, with the cutoff time set to $1\,200 \times 32 = 38\,400$ CPU seconds (*i.e.*, $1\,200$ seconds wall-clock time on a CPU without any additional load). In addition, since our parallel portfolio used 1 core for each of its 4 component configurations, we gathered additional baseline results from running the default configuration of Gurobi on the same number of cores and with the same cutoff as our portfolio, *i.e.*, $9\,600 \times 4 =$ $38\,400$ CPU seconds. Lastly, to maximise the number of instances processed in parallel, we considered Gurobi in its default configuration limited to a single CPU core, with cutoff time of 38 400 seconds. In short, we compared our approach against baselines with a variable number of cores, and a constant budget in terms of CPU time.

Portfolio* vs *Default (32 cores). As seen in Table 1, our portfolio was able to certify a statistically significantly larger fraction of samples , while reducing CPU time by a factor of 4.7 on the solvable instances (8 478 *vs* 39 772 CPU seconds). Besides that, the portfolio strongly outperformed this baseline in terms of timeouts (14.96% *vs* 21.29%).

Portfolio* vs *Default (4 cores). The default configuration of Gurobi running on 4 cores was clearly outperformed by our portfolio in terms of CPU time (8 478 *vs* 22 065 CPU seconds). Furthermore, the portfolio was able to reduce the number of timeouts (14.96% *vs* 17.74%), while improving on the upper bound (23.86% *vs* 27.49%). In other words, the portfolio certified more samples, but consumed fewer computational resources to do so, although it was provided with the same number of cores and overall time budget.

Portfolio* vs *Default (1 core). Lastly, we compared the portfolio against the default configuration of Gurobi running on a single core. Here, our portfolio showed improved performance in terms of PAR10 (8 478 *vs* 20 117 CPU seconds) as well as the fraction of timeouts (14.96% *vs* 17.66%) and the upper bound (23.86% *vs* 27.58%), which clearly demonstrates the strength of the portfolio approach.

5 Conclusion

In this study, we have, for the first time, demonstrated the effectiveness of automated algorithm configuration in the context of MIP-based neural network verification. More specifically, we constructed a parallel portfolio of solver configurations with complementary strengths and evaluated it against a recent, state-of-the-art MIP-based verification method. Specifically, we verified a neural network classifier from the literature on the MNIST dataset, achieving a 4.7-fold reduction in CPU time as well as 1.42 times fewer timeouts than the strongest baseline, while certifying a statistically significantly larger number of samples. In future work, we plan to apply our approach to other MIP-based verification engines, network architectures and datasets.

REFERENCES

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring Neural Net Robustness with Constraints. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS 2016)*, pp. 2613–2621, 2016.

Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A Unified View of Piecewise Linear Neural Network Verification. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, pp. 4790–4799, 2018.

Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (IEEE S&P 2017)*, pp. 39–57, 2017.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A Dual Approach to Scalable Verification of Deep Networks. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*, volume 1, pp. 550–559, 2018.

Ruediger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA 2017)*, pp. 269–286, 2017.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proceedings of the 39th IEEE Symposium on Security and Privacy (IEEE S&P 2018)*, pp. 3–18, 2018.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572*, 2014.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Automated Configuration of Mixed Integer Programming Solvers. In *Proceedings of the 7th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming (CPAIOR 2010)*, pp. 186–202, 2010.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION 5)*, pp. 507–523, 2011.

Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm Selection and Scheduling. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP 2011)*, pp. 454–469, 2011.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification(CAV 2017)*, pp. 97–117, 2017.

Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*, pp. 149–190. Springer, 2016.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Marius Lindauer, Holger H Hoos, Frank Hutter, and Torsten Schaub. AutoFolio: An Automatically Configured Algorithm Selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

Manuel Lopez-Ibanez and Thomas Stützle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582, 2014.

Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Parallel SAT Solver Selection and Scheduling. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012)*, pp. 512–526. Springer, 2012.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified Defenses against Adversarial Examples. *arXiv preprint arXiv:1801.09344*, 2018.

Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker. Towards Verification of Artificial Neural Networks. In *Proceedings of the 18th Workshop on Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV 2015)*, pp. 30–40, 2015.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2014.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*, 2019.

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output Reachable Set Estimation and Verification for Multilayer Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, 2018.

Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.