Verifying Generalization in Deep Learning

Guy Amir(✉), Osher Maayan, Tom Zelazny, Guy Katz, and Michael Schapira

The Hebrew University of Jerusalem, Jerusalem, Israel
{guyam,osherm,tomz,guykatz,schapiram}@cs.huji.ac.il

Abstract. Deep neural networks (DNNs) are the workhorses of deep learning, which constitutes the state of the art in numerous application domains. However, DNN-based decision rules are notoriously prone to poor *generalization*, i.e., may prove inadequate on inputs not encountered during training. This limitation poses a significant obstacle to employing deep learning for mission-critical tasks, and also in real-world environments that exhibit high variability. We propose a novel, verification-driven methodology for identifying DNN-based decision rules that generalize well to new input domains. Our approach quantifies generalization to an input domain by the extent to which decisions reached by *independently trained* DNNs are in agreement for inputs in this domain. We show how, by harnessing the power of DNN verification, our approach can be efficiently and effectively realized. We evaluate our verification-based approach on three deep reinforcement learning (DRL) benchmarks, including a system for Internet congestion control. Our results establish the usefulness of our approach. More broadly, our work puts forth a novel objective for formal verification, with the potential for mitigating the risks associated with deploying DNN-based systems in the wild.

1 Introduction

Over the past decade, deep learning [35] has achieved state-of-the-art results in natural language processing, image recognition, game playing, computational biology, and many additional fields [4,18,21,45,50,84,85]. However, despite its impressive success, deep learning still suffers from severe drawbacks that limit its applicability in domains that involve mission-critical tasks or highly variable inputs.

One such crucial limitation is the notorious difficulty of deep neural networks (DNNs) to *generalize* to new input domains, i.e., their tendency to perform poorly on inputs that significantly differ from those encountered while training. During training, a DNN is presented with input data sampled from a specific distribution over some input domain ("*in-distribution*" inputs). The induced DNN-based rules may fail in generalizing to inputs not encountered during training due to (1) the DNN being invoked "out-of-distribution" (OOD), i.e., when there is a mismatch between the distribution over inputs in the training data and in

G. Amir and O. Maayan—Contributed equally.

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13965, pp. 438–455, 2023.
https://doi.org/10.1007/978-3-031-37703-7_21

the DNN's operational data; (2) some inputs not being sufficiently represented in the finite training data (e.g., various low-probability corner cases); and (3) "overfitting" the decision rule to the training data.

A notable example of the importance of establishing the generalizability of DNN-based decisions lies in recently proposed applications of deep reinforcement learning (DRL) [56] to real-world systems. Under DRL, an *agent*, realized as a DNN, is trained by repeatedly interacting with its environment to learn a decision-making *policy* that attains high performance with respect to a certain objective ("*reward*"). DRL has recently been applied to many real-world challenges [20,44,54,55,64–67,96,108]. In many application domains, the learned policy is expected to perform well across a daunting breadth of operational environments, whose diversity cannot possibly be captured in the training data. Further, the cost of erroneous decisions can be dire. Our discussion of DRL-based Internet congestion control (see Sect. 4.3) illustrates this point.

Here, we present a methodology for identifying DNN-based decision rules that generalize well to *all possible distributions* over an input domain of interest. Our approach hinges on the following key observation. DNN training in general, and DRL policy training in particular, incorporate multiple stochastic aspects, such as the initialization of the DNN's weights and the order in which inputs are observed during training. Consequently, even when DNNs with *the same* architecture are trained to perform an *identical* task on *the same* data, somewhat different decision rules will typically be learned. Paraphrasing Tolstoy's Anna Karenina [93], we argue that "successful decision rules are all alike; but every unsuccessful decision rule is unsuccessful in its own way". Differently put, when examining the decisions by several *independently trained* DNNs on a certain input, these are likely to agree only when their (similar) decisions yield high performance.

In light of the above, we propose the following heuristic for generating DNN-based decision rules that generalize well to *an entire* given domain of inputs: independently train multiple DNNs, and then seek a subset of these DNNs that are in strong agreement across *all* possible inputs in the considered input domain (implying, by our hypothesis, that these DNNs' learned decision rules generalize well to all probability distributions over this domain). Our evaluation demonstrates (see Sect. 4) that this methodology is extremely powerful and enables distilling from a collection of decision rules the few that indeed generalize better to inputs within this domain. Since our heuristic seeks DNNs whose decisions are in agreement for *each and every* input in a specific domain, the decision rules reached this way achieve robustly high generalization across different possible distributions over inputs in this domain.

Since our methodology involves contrasting the outputs of different DNNs over possibly *infinite* input domains, using formal verification is natural. To this end, we build on recent advances in formal verification of DNNs [2,12,14,16,27,60,78,86,102]. DNN verification literature has focused on establishing the local adversarial robustness of DNNs, i.e., seeking small input perturbations that result in misclassification by the DNN [31,36,61]. Our approach broadens the applicability of DNN verification by demonstrating, for the first time (to the best of our knowledge), how it can also be used to identify DNN-based decision rules that generalize well. More specifically, we show how, for a given input domain, a DNN verifier can be utilized to assign a score to a DNN reflecting its level of agreement with other DNNs across the entire input domain. This enables iteratively pruning the set of candidate DNNs, eventually keeping only those in strong agreement, which tend to generalize well.

To evaluate our methodology, we focus on three popular DRL benchmarks: (i) *Cartpole*, which involves controlling a cart while balancing a pendulum; (ii) *Mountain Car*, which involves controlling a car that needs to escape a valley; and (iii) *Aurora*, an Internet congestion controller.

Aurora is a particularly compelling example for our approach. While Aurora is intended to tame network congestion across a vast diversity of real-world Internet environments, Aurora is trained only on synthetically generated data. Thus, to deploy Aurora in the real world, it is critical to ensure that its policy is sound for numerous scenarios not captured by its training inputs.

Our evaluation results show that, in all three settings, our verification-driven approach is successful at ranking DNN-based DRL policies according to their ability to generalize well to out-of-distribution inputs. Our experiments also demonstrate that formal verification is superior to gradient-based methods and predictive uncertainty methods. These results showcase the potential of our approach. Our code and benchmarks are publicly available as an artifact accompanying this work [8].

The rest of the paper is organized as follows. Section 2 contains background on DNNs, DRLs, and DNN verification. In Sect. 3 we present our verification-based methodology for identifying DNNs that successfully generalize to OOD inputs. We present our evaluation in Sect. 4. Related work is covered in Sect. 5, and we conclude in Sect. 6.

2 Background

Deep Neural Networks (DNNs) [35] are directed graphs that comprise several layers. Upon receiving an assignment of values to the nodes of its first (input) layer, the DNN propagates these values, layer by layer, until ultimately reaching the assignment of the final (output) layer. Computing the value for each node is performed according to the type of that node's layer. For example, in weighted-sum layers, the node's value is an affine combination of the values of the nodes in the preceding layer to which it is connected. In *rectified linear unit* (*ReLU*) layers, each node y computes the value $y = \text{ReLU}(x) = \max(x, 0)$, where x is a single node from the preceding layer. For additional details on DNNs and their

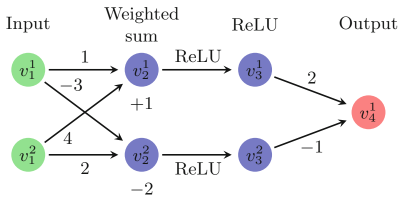


Fig. 1. A toy DNN.

training see [35]. Figure 1 depicts a toy DNN. For input $V_1 = [1, 2]^T$, the second layer computes the (weighted sum) $V_2 = [10, -1]^T$. The ReLU functions are subsequently applied in the third layer, and the result is $V_3 = [10, 0]^T$. Finally, the network's single output is $V_4 = [20]$.

Deep Reinforcement Learning (DRL) [56] is a machine learning paradigm, in which a DRL agent, implemented as a DNN, interacts with an *environment* across discrete time-steps $t \in 0, 1, 2....$ At each time-step, the agent is presented with the environment's *state* $s_t \in \mathcal{S}$, and selects an *action* $N(s_t) = a_t \in \mathcal{A}$. The environment then transitions to its next state s_{t+1}, and presents the agent with the *reward* r_t for its previous action. The agent is trained through repeated interactions with its environment to maximize the *expected cumulative discounted reward* $R_t = \mathbb{E}\big[\sum_t \gamma^t \cdot r_t\big]$ (where $\gamma \in \big[0, 1\big]$ is termed the *discount factor*) [38, 82, 90, 91, 97, 107].

DNN and DRL Verification. A sound DNN verifier [46] receives as input (i) a *trained* DNN N; (ii) a precondition P on the DNN's inputs, limiting the possible assignments to a domain of interest; and (iii) a postcondition Q on the DNN's outputs, limiting the possible outputs of the DNN. The verifier can reply in one of two ways: (i) `SAT`, with a concrete input x' for which $P(x') \wedge Q(N(x'))$ is satisfied; or (ii) `UNSAT`, indicating there does not exist such an x'. Typically, Q encodes the *negation* of N's desirable behavior for inputs that satisfy P. Thus, a `SAT` result indicates that the DNN errs, and that x' triggers a bug; whereas an `UNSAT` result indicates that the DNN performs as intended. An example of this process appears in Appendix B of our extended paper [7]. To date, a plethora of verification approaches have been proposed for general, feed-forward DNNs [3, 31, 41, 46, 61, 99], as well as DRL-based agents that operate within reactive environments [5, 9, 15, 22, 28].

3 Quantifying Generalizability via Verification

Our approach for assessing how well a DNN is expected to generalize on out-of-distribution inputs relies on the "Karenina hypothesis": while there are many (possibly infinite) ways to produce *incorrect results*, correct outputs are likely to be fairly similar. Hence, to identify DNN-based decision rules that generalize well to new input domains, we advocate training multiple DNNs and scoring the learned decision models according to how well their outputs are aligned with those of the other models for the considered input domain. These scores can be computed using a backend DNN verifier. We show how, by iteratively filtering out models that tend to disagree with the rest, DNNs that generalize well can be effectively distilled.

We begin by introducing the following definitions for reasoning about the extent to which two DNN-based decision rules are in agreement over an input domain.

Definition 1 (Distance Function). *Let $\mathcal{O}$ be the space of possible outputs for a DNN. A* distance function *for $\mathcal{O}$ is a function $d : \mathcal{O} \times \mathcal{O} \mapsto \mathbb{R}^+$.*

Intuitively, a distance function (e.g., the L_1 norm) allows us to quantify the level of (dis)agreement between the decisions of two DNNs on the same input. We elaborate on some choices of distance functions that may be appropriate in various domains in Appendix B of our extended paper [7].

Definition 2 (Pairwise Disagreement Threshold). *Let N_1, N_2 be DNNs with the same output space $\mathcal{O}$, let d be a distance function, and let Ψ be an input domain. We define the* pairwise disagreement threshold *(PDT) of N_1 and N_2 as:*

$$\alpha = PDT_{d,\Psi}(N_1, N_2) \triangleq \min \left\{ \alpha' \in \mathbb{R}^+ \mid \forall x \in \Psi \colon d(N_1(x), N_2(x)) \leq \alpha' \right\}$$

The definition captures the notion that for *any* input in Ψ, N_1 and N_2 produce outputs that are at most α-distance apart. A small α value indicates that the outputs of N_1 and N_2 are close for all inputs in Ψ, whereas a high value indicates that there exists an input in Ψ for which the decision models diverge significantly.

To compute PDT values, our approach employs verification to conduct a binary search for the maximum distance between the outputs of two DNNs; see Algorithm 1.

Algorithm 1. Pairwise Disagreement Threshold

Input: DNNs $(N_i,\ N_j)$, distance func. d, input domain Ψ, max. disagreement $\mathtt{M} > 0$
Output: PDT(N_i, N_j)

```
1: low ← 0, high ← M
2: while (low < high) do
3:     $\alpha \leftarrow \frac{1}{2} \cdot (\texttt{low} + \texttt{high})$
4:     query ← SMT SOLVER $\langle P \leftarrow \Psi, [N_i; N_j], Q \leftarrow d(N_i, N_j) \geq \alpha \rangle$
5:     if query is SAT then: low ← $\alpha$
6:     else if query is UNSAT then: high ← $\alpha$
7: end while
8: return $\alpha$
```

Pairwise disagreement thresholds can be aggregated to measure the disagreement between a decision model and a *set* of other decision models, as defined next.

Definition 3 (Disagreement Score). *Let $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ be a set of k DNN-induced decision models, let d be a distance function, and let Ψ be an input domain. A model's* disagreement score *(DS) with respect to $\mathcal{N}$ is defined as:*

$$DS_{\mathcal{N},d,\Psi}(N_i) = \frac{1}{|\mathcal{N}| - 1} \sum_{j \in [k], j \neq i} PDT_{d,\Psi}(N_i, N_j)$$

Intuitively, the disagreement score measures how much a single decision model tends to disagree with the remaining models, on average.

Using disagreement scores, our heuristic employs an iterative scheme for selecting a subset of models that generalize to OOD scenarios—as encoded by inputs in Ψ (see Algorithm 2). First, a set of k DNNs $\{N_1, N_2, \ldots, N_k\}$ are *independently* trained on the training data. Next, a backend verifier is invoked to calculate, for each of the $\binom{k}{2}$ DNN-based model pairs, their respective pairwise-disagreement threshold (up to some ϵ accuracy). Next, our algorithm iteratively: (i) calculates the disagreement score for each model in the remaining subset of models; (ii) identifies the models with the (relative) highest DS scores; and (iii) removes them (Line 9 in Algorithm 2). The algorithm terminates after exceeding a user-defined number of iterations (Line 3 in Algorithm 2), or when the remaining models "agree" across the input domain, as indicated by nearly identical disagreement scores (Line 7 in Algorithm 2). We note that the algorithm is also given an upper bound (M) on the maximum difference, informed by the user's domain-specific knowledge.

Algorithm 2. Model Selection

Input: Set of models $\mathcal{N} = \{N_1, \ldots, N_k\}$, max disagreement M, number of ITERATIONS
Output: $\mathcal{N}' \subseteq \mathcal{N}$

```
1: PDT ← Pairwise Disagreement Thresholds($\mathcal{N}, d, \Psi$, M)    ▷ table with all PDTs
2: $\mathcal{N}' \leftarrow \mathcal{N}$
3: for $l = 1 \ldots$ITERATIONS do
4:     for $N_i \in \mathcal{N}'$ do
5:         currentDS[$N_i$] ← $DS_{\mathcal{N}'}(N_i,$ PDT$)$    ▷ based on definition 3
6:     end for
7:     if modelScoresAreSimilar(currentDS) then: break
8:     modelsToRemove ← findModelsWithHighestDS(currentDS)
9:     $\mathcal{N}' \leftarrow \mathcal{N}' \setminus$ modelsToRemove    ▷ remove models that tend to disagree
10: end for
11: return $\mathcal{N}'$
```

DS Removal Threshold. Different criteria are possible for determining the DS threshold above for which models are removed, and how many models to remove in each iteration (Line 8 in Algorithm 2). A natural and simple approach, used in our evaluation, is to remove the $p\%$ models with the *highest* disagreement scores, for some choice of p (25% in our evaluation). Due to space constraints, a thorough discussion of additional filtering criteria (all of which proved successful) is relegated to Appendix C of our extended paper [7].

4 Evaluation

We extensively evaluated our method using three DRL benchmarks. As discussed in the introduction, verifying the generalizability of DRL-based systems is important since such systems are often expected to provide robustly high performance across a broad range of environments, whose diversity is not captured by the training data. Our evaluation spans two classic DRL settings, Cartpole [17] and Mountain Car [68], as well as the recently proposed Aurora congestion controller for Internet traffic [44]. Aurora is a particularly compelling example for a fairly complex DRL-based system that addresses a crucial real-world challenge and must generalize to real-world conditions not represented in its training data.

Setup. For each of the three DRL benchmarks, we first trained multiple DNNs with the same architecture, where the training process differed only in the random seed used. We then removed from this set of DNNs all but the ones that achieved high reward values in-distribution (to eliminate the possibility that a decision model generalizes poorly simply due to poor training). Next, we defined out-of-distribution input domains of interest for each specific benchmark, and used Algorithm 2 to select the models most likely to generalize well on those domains according to our framework. To establish the ground truth for how well different models actually generalize in practice, we then applied the models to OOD inputs drawn from the considered domain and ranked them based on their empirical performance (average reward). To investigate the robustness of our results, the last step was conducted for varying choices of probability distributions over the inputs in the domain. All DNNs used have a feed-forward architecture comprised of two hidden layers of ReLU activations, and include 32-64 neurons in the first hidden layer, and 16 neurons in the second hidden layer.

The results indicate that models selected by our approach are likely to perform *significantly better* than the rest. Below we describe the gist of our evaluation; extensive additional information is available in [7].

4.1 Cartpole

Cartpole [33] is a well-known RL benchmark in which an agent controls the movement of a cart with an upside-down pendulum ("pole") attached to its top. The cart moves on a platform and the agent's goal is to keep the pole balanced for as long as possible (see Fig. 2).

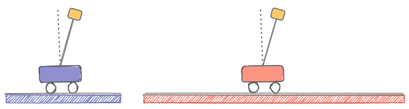


Fig. 2. Cartpole: in-distribution setting (blue) and OOD setting (red). (Color figure online)

Agent and Environment. The agent's inputs are $s = (x, v_x, \theta, v_\theta)$, where x represents the cart's location on the platform, θ represents the pole's angle (i.e., $|\theta| \approx 0$ for a balanced pole, $|\theta| \approx 90°$ for an unbalanced pole), v_x represents the cart's horizontal velocity and v_θ represents the pole's angular velocity.

In-Distribution Inputs. During training, the agent is incentivized to balance the pole, while staying within the platform's boundaries. In each iteration, the agent's single output indicates the cart's acceleration (sign and magnitude) for the next step. During training, we defined the platform's bounds to be $[-2.4, 2.4]$, and the cart's initial position as near-static, and close to the center of the platform (left-hand side of Fig. 2). This was achieved by drawing the cart's initial state vector values uniformly from the range $[-0.05, 0.05]$.

(OOD) Input Domain. We consider an input domain with larger platforms than the ones used in training. To wit, we now allow the x coordinate of the input vectors to cover a wider range of $[-10, 10]$. For the other inputs, we used the same bounds as during the training. See [7] for additional details.

Evaluation. We trained $k = 16$ models, all of which achieved high rewards during training on the short platform. Next, we ran Algorithm 2 until convergence (7 iterations, in our experiments) on the aforementioned input domain, resulting in a set of 3 models. We then tested all 16 original models using (OOD) inputs drawn from the new domain, such that the generated distribution encodes a novel setting: the cart is now placed at the center of a much longer, shifted platform (see the red cart in Fig. 2).

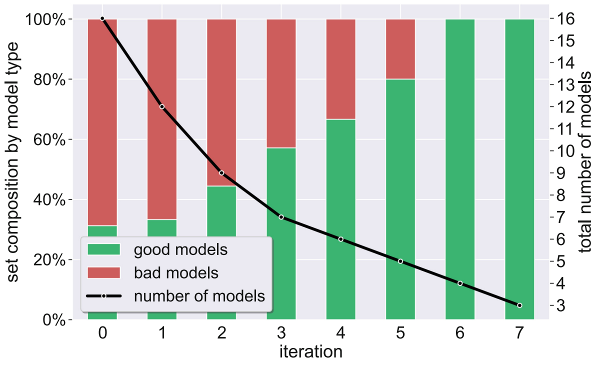


Fig. 3. Cartpole: Algorithm 2's results, per iteration: the bars reflect the ratio between the good/bad models (left y-axis) in the surviving set of models, and the curve indicates the number of surviving models (right y-axis).

All other parameters in the OOD environment were identical to those used for the original training. Figure 9 (in [7]) depicts the results of evaluating the models using 20,000 OOD instances. Of the original 16 models, 11 scored a low-to-mediocre average reward, indicating their poor ability to generalize to this new distribution. Only 5 models obtained high reward values, including the 3 models identified by Algorithm 2; thus implying that our method was able to effectively remove all 11 models that would have otherwise performed poorly in this OOD setting (see Fig. 3). For additional information, see [7].

4.2 Mountain Car

For our second experiment, we evaluated our method on the Mountain Car [79] benchmark, in which an agent controls a car that needs to learn how to escape a valley and reach a target. As in the Cartpole experiment, we selected a set of models that performed well in-distribution and applied our method to identify a subset of models that make similar decisions in a predefined input domain. We again generated OOD inputs (relative to the training) from within this domain, and observed that the models selected by our algorithm indeed generalize significantly better than their peers that were iteratively removed. Detailed information about this benchmark can be found in Appendix E of our extended paper [7].

4.3 Aurora Congestion Controller

In our third benchmark, we applied our method to a complex, real-world system that implements a policy for Internet congestion control. The goal of congestion control is to determine, for each traffic source in a communication network, the pace at which data packets should be sent into the network. Congestion control is a notoriously difficult and fundamental challenge in computer networking [59,69]; sending packets too fast might cause network congestion, leading to data loss and delays. Conversely, low sending rates might under-utilize available network bandwidth. *Aurora* [44] is a DRL-based congestion controller that is the subject of recent work on DRL verification [9,28]. In each time-step, an Aurora agent observes statistics regarding the network and decides the packet sending rate for the following time-step. For example, if the agent observes excellent network conditions (e.g., no packet loss), we expect it to increase the packet sending rate to better utilize the network. We note that Aurora handles a much harder task than classical RL benchmarks (e.g., Cartpole and Mountain Car): congestion controllers must react gracefully to various possible events based on nuanced signals, as reflected by Aurora's inputs. Here, unlike in the previous benchmarks, it is not straightforward to characterize the optimal policy.

Agent and Environment. Aurora's inputs are t vectors $v_1, \ldots, v_t$, representing observations from the t previous time-steps. The agent's single output value indicates the change in the packet sending rate over the next time-step. Each vector $v_i \in \mathbb{R}^3$ includes three distinct values, representing statistics that reflect the network's condition (see details in Appendix F of [7]). In line with previous work [9,28,44], we set $t = 10$ time-steps, making Aurora's inputs of size $3t = 30$. The reward function is a linear combination of the data sender's throughput, latency, and packet loss, as observed by the agent (see [44] for additional details).

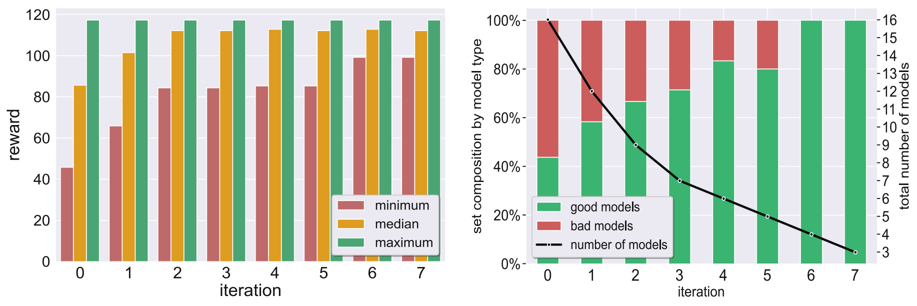
In-Distribution Inputs. Aurora's training applies the congestion controller to simple network scenarios where a *single* sender sends traffic towards a *single* receiver across a *single* network link. Aurora is trained across varying choices of initial sending rate, link bandwidth, link packet-loss rate, link latency, and size of the link's packet buffer. During training, packets are initially sent by Aurora at a rate corresponding to $0.3 - 1.5$ times the link's bandwidth.

(OOD) Input Domain. In our experiments, the input domain encoded a link with a *shallow packet buffer*, implying that only a few packets can accumulate in the network (while most excess traffic is discarded), causing the link to exhibit a volatile behavior. This is captured by the initial sending rate being up to 8 times the link's bandwidth, to model the possibility of a dramatic decrease in available bandwidth (e.g., due to competition, traffic shifts, etc.). See [7] for additional details.

Evaluation. We ran our algorithm and scored the models based on their disagreement upon this large domain, which includes inputs they had not encountered during training, representing the aforementioned novel link conditions.

Experiment (1): High Packet Loss. In this experiment, we trained over 100 Aurora agents in the original (in-distribution) environment. Out of these, we selected $k = 16$ agents that achieved a high average reward in-distribution (see Fig. 20a in [7]). Next, we evaluated these agents on OOD inputs that are included in the previously described domain. The main difference between the training distribution and the new (OOD) ones is the possibility of extreme packet loss rates upon initialization.

Our evaluation over the OOD inputs, within the domain, indicates that although all 16 models performed well in-distribution, only 7 agents could successfully handle such OOD inputs (see Fig. 20b in [7]). When we ran Algorithm 2 on the 16 models, it was able to filter out *all* 9 models that generalized poorly on the OOD inputs (see Fig. 4). In particular, our method returned model $\{16\}$, which is the best-performing model according to our simulations. We note that in the first iterations, the four models to be filtered out were models $\{1, 2, 6, 13\}$, which are indeed the four worst-performing models on the OOD inputs (see Appendix F of [7]).



(a) *Reward statistics of remaining models* (b) *Ratio between good/bad models*

Fig. 4. Aurora: Algorithm 2's results, per iteration.

Experiment (2): Additional Distributions over OOD Inputs. To further demonstrate that, in the specified input domain, our method is indeed likely to keep better-performing models while removing bad models, we reran the previous Aurora experiments for additional distributions (probability density functions) over the OOD inputs. Our evaluation reveals that all models removed by Algorithm 2 achieved low reward values also for these additional distributions. These results highlight an important advantage of our approach: it applies to all inputs within the considered domain, and so it applies to *all distributions over these inputs*.

Additional Experiments. We also generated a new set of Aurora models by altering the training process to include significantly longer interactions. We then repeated the aforementioned experiments. The results (summarized in [7]) demonstrate that our approach (again) successfully selected a subset of models that generalizes well to distributions over the OOD input domain.

4.4 Comparison to Additional Methods

Gradient-based methods [40,53,62,63] are optimization algorithms capable of finding DNN inputs that satisfy prescribed constraints, similarly to verification methods. These algorithms are extremely popular due to their simplicity and scalability. However, this comes at the cost of being inherently incomplete and not as precise as DNN verification [11,101]. Indeed, when modifying our algorithm to calculate PDT scores with gradient-based methods, the results (summarized in Appendix G of [7]) reveal that, in our context, the verification-based approach is superior to the gradient-based ones. Due to the incompleteness of gradient-based approaches [101], they often computed sub-optimal PDT values, resulting in models that generalize poorly being retained.

Predictive uncertainty methods [1,74] are *online* methods for assessing uncertainty with respect to observed inputs, to determine whether an encountered input is drawn from the training distribution. We ran an experiment comparing our approach to uncertainty-prediction-based model selection: we generated ensembles [23,30,51] of our original models, and used a variance-based metric (motivated by [58]) to identify subsets of models with low output variance on OOD-sampled inputs. Similar to gradient-based methods, predictive-uncertainty techniques proved fast and scalable, but lacked the precision afforded by verification-driven model selection and were unable to discard poorly generalizing models. For example, when ranking Cartpole models by their uncertainty on OOD inputs, the three models with the lowest uncertainty included also "bad" models, which had been filtered out by our approach.

5 Related Work

Recently, a plethora of approaches and tools have been put forth for ensuring DNN correctness [2,6,10,15,19,24–27,29,31,32,34,36,37,41–43,46–49,52, 57,61,70,76,81,83,86,87,89,92,94,95,98,100,102,104,106], including techniques for DNN shielding [60], optimization [14,88], quantitative verification [16], abstraction [12,13,73,78,86,105], size reduction [77], and more. Non-verification techniques, including runtime-monitoring [39], ensembles [71,72,80,103] and additional methods [75] have been utilized for OOD input detection.

In contrast to the above approaches, we aim to establish *generalization guarantees* with respect to an *entire input domain* (spanning all distributions across this domain). In addition, to the best of our knowledge, ours is the first attempt to exploit variability across models for distilling a subset thereof, with improved *generalization* capabilities. In particular, it is also the first approach to apply formal verification for this purpose.

6 Conclusion

This work describes a novel, verification-driven approach for identifying DNN models that generalize well to an input domain of interest. We presented an iterative scheme that employs a backend DNN verifier, allowing us to score models based on their ability to produce similar outputs on the given domain. We demonstrated extensively that this approach indeed distills models capable of good generalization. As DNN verification technology matures, our approach will become increasingly scalable, and also applicable to a wider variety of DNNs.

Acknowledgements. The work of Amir, Zelazny, and Katz was partially supported by the Israel Science Foundation (grant number 683/18). The work of Amir was supported by a scholarship from the Clore Israel Foundation. The work of Maayan and Schapira was partially supported by funding from Huawei.

References

1. Abdar, M., et al.: A review of uncertainty quantification in deep learning: techniques, applications and challenges. Inf. Fusion **76**, 243–297 (2021)
2. Alamdari, P., Avni, G., Henzinger, T., Lukina, A.: Formal methods with a touch of magic. In: Proceedings 20th International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 138–147 (2020)
3. Albarghouthi, A.: Introduction to Neural Network Verification (2021). verifieddeeplearning.com
4. AlQuraishi, M.: AlphaFold at CASP13. Bioinformatics **35**(22), 4862–4865 (2019)
5. Amir, G., et al.: Verifying learning-based robotic navigation systems. In: Sankaranarayanan, S., Sharygina, N. (eds.) Proceedings 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 607–627. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30823-9_31
6. Amir, G., Freund, Z., Katz, G., Mandelbaum, E., Refaeli, I.: veriFIRE: verifying an industrial, learning-based wildfire detection system. In: Proceedings 25th International Symposium on Formal Methods (FM), pp. 648–656. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-27481-7_38
7. Amir, G., Maayan, O., Zelazny, O., Katz, G., Schapira, M.: Verifying generalization in deep learning. Technical report (2023). https://arxiv.org/abs/2302.05745
8. Amir, G., Maayan, O., Zelazny, T., Katz, G., Schapira, M.: Verifying generalization in deep learning: artifact (2023). https://zenodo.org/record/7884514#.ZFAz_3ZBy3B
9. Amir, G., Schapira, M., Katz, G.: Towards scalable verification of deep reinforcement learning. In: Proceedings 21st Internationl Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 193–203 (2021)
10. Amir, G., Wu, H., Barrett, C., Katz, G.: An SMT-based approach for verifying binarized neural networks. In: TACAS 2021. LNCS, vol. 12652, pp. 203–222. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_11
11. Amir, G., Zelazny, T., Katz, G., Schapira, M.: Verification-aided deep ensemble selection. In: Proceedings 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 27–37 (2022)

12. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: Proceedings 40th ACM SIGPLAN Conference on Programming Languages Design and Implementations (PLDI), pp. 731–744 (2019)
13. Ashok, P., Hashemi, V., Kretinsky, J., Mohr, S.: DeepAbstract: neural network abstraction for accelerating verification. In: Proceedings 18th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 92–107 (2020)
14. Avni, G., Bloem, R., Chatterjee, K., Henzinger, T.A., Könighofer, B., Pranger, S.: Run-time optimization for learned controllers through quantitative games. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 630–649. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_36
15. Bacci, E., Giacobbe, M., Parker, D.: Verifying reinforcement learning up to infinity. In: Proceedings 30th International Joint Conference on Artificial Intelligence (IJCAI) (2021)
16. Baluta, T., Shen, S., Shinde, S., Meel, K., Saxena, P.: Quantitative verification of neural networks and its security applications. In: Proceedings ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1249–1264 (2019)
17. Barto, A., Sutton, R., Anderson, C.: Neuronlike adaptive elements that can solve difficult learning control problems. In: Proceedings of IEEE Systems Man and Cybernetics Conference (SMC), pp. 834–846 (1983)
18. Bojarski, M., et al.: End to end learning for self-driving cars. Technical report (2016). http://arxiv.org/abs/1604.07316
19. Bunel, R., Turkaslan, I., Torr, P., Kohli, P., Mudigonda, P.: A unified view of piecewise linear neural network verification. In: Proceedings 32nd Conference on Neural Information Processing Systems (NeurIPS), pp. 4795–4804 (2018)
20. Chen, W., Xu, Y., Wu, X.: Deep reinforcement learning for multi-resource multi-machine job scheduling. Technical report (2017). http://arxiv.org/abs/1711.07440
21. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. (JMLR) **12**, 2493–2537 (2011)
22. Corsi, D., Marchesini, E., Farinelli, A.: Formal verification of neural networks for safety-critical tasks in deep reinforcement learning. In: Proceedings 37th Conference on Uncertainty in Artificial Intelligence (UAI), pp. 333–343 (2021)
23. Dietterich, T.: Ensemble methods in machine learning. In: Proceedings 1st International Workshop on Multiple Classifier Systems (MCS), pp. 1–15 (2020)
24. Dong, G., Sun, J., Wang, J., Wang, X., Dai, T.: Towards repairing neural networks correctly. Technical report (2020). http://arxiv.org/abs/2012.01872
25. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Proceedings 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC), pp. 157–168 (2019)
26. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Learning and verification of feedback control systems using feedforward neural networks. IFAC-PapersOnLine **51**(16), 151–156 (2018)
27. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Proceedings 15th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 269–286 (2017)

28. Eliyahu, T., Kazak, Y., Katz, G., Schapira, M.: Verifying learning-augmented systems. In: Proceedings Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), pp. 305–318 (2021)
29. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: toward safe control through proof and learning. In: Proceedings 32nd AAAI Conference on Artificial Intelligence (AAAI) (2018)
30. Ganaie, M., Hu, M., Malik, A., Tanveer, M., Suganthan, P.: Ensemble deep learning: a review. Eng. Appl. Artif. Intell. **115**, 105151 (2022)
31. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, E., Chaudhuri, S., Vechev, M.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: Proceedings 39th IEEE Symposium on Security and Privacy (S&P) (2018)
32. Geng, C., Le, N., Xu, X., Wang, Z., Gurfinkel, A., Si, X.: Toward reliable neural specifications. Technical report (2022). https://arxiv.org/abs/2210.16114
33. Geva, S., Sitte, J.: A cartpole experiment benchmark for trainable controllers. IEEE Control Syst. Mag. **13**(5), 40–51 (1993)
34. Goldberger, B., Adi, Y., Keshet, J., Katz, G.: Minimal modifications of deep neural networks using verification. In: Proceedings 23rd Proceedings Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), pp. 260–278 (2020)
35. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
36. Gopinath, D., Katz, G., Păsăreanu, C., Barrett, C.: DeepSafe: a data-driven approach for assessing robustness of neural networks. In: Proceedings 16th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 3–19 (2018)
37. Goubault, E., Palumby, S., Putot, S., Rustenholz, L., Sankaranarayanan, S.: Static analysis of ReLU neural networks with tropical Polyhedra. In: Proceedings 28th International Symposium on Static Analysis (SAS), pp. 166–190 (2021)
38. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings Conference on Machine Learning, pp. 1861–1870. PMLR (2018)
39. Hashemi, V., Křetínsky, J., Rieder, S., Schmidt, J.: Runtime monitoring for out-of-distribution detection in object detection neural networks. Technical report (2022). http://arxiv.org/abs/2212.07773
40. Huang, S., Papernot, N., Goodfellow, I., Duan, Y., Abbeel, P.: Adversarial attacks on neural network policies. Technical report (2017). https://arxiv.org/abs/1702.02284
41. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Proceedings 29th International Conference on Computer Aided Verification (CAV), pp. 3–29 (2017)
42. Isac, O., Barrett, C., Zhang, M., Katz, G.: Neural network verification with proof production. In: Proceedings 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 38–48 (2022)
43. Jacoby, Y., Barrett, C., Katz, G.: Verifying recurrent neural networks using invariant inference. In: Proceedings 18th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 57–74 (2020)
44. Jay, N., Rotman, N., Godfrey, B., Schapira, M., Tamar, A.: A deep reinforcement learning perspective on internet congestion control. In: Proceedings 36th International Conference on Machine Learning (ICML), pp. 3050–3059 (2019)

45. Julian, K., Lopez, J., Brush, J., Owen, M., Kochenderfer, M.: Policy compression for aircraft collision avoidance systems. In: Proceedings 35th Digital Avionics Systems Conference (DASC), pp. 1–10 (2016)
46. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Proceedings 29th International Conference on Computer Aided Verification (CAV), pp. 97–117 (2017)
47. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: a calculus for reasoning about deep neural networks. Formal Methods Syst. Des. (FMSD) (2021)
48. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Proceedings 31st International Conference on Computer Aided Verification (CAV), pp. 443–452 (2019)
49. Könighofer, B., Lorber, F., Jansen, N., Bloem, R.: Shield synthesis for reinforcement learning. In: Proceedings International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), pp. 290–306 (2020)
50. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Proceedings 26th Conference on Neural Information Processing Systems (NeurIPS), pp. 1097–1105 (2012)
51. Krogh, A., Vedelsby, J.: Neural network ensembles, cross validation, and active learning. In: Proceedings 7th Conference on Neural Information Processing Systems (NeurIPS), pp. 231–238 (1994)
52. Kuper, L. Katz, G., Gottschlich, J., Julian, K., Barrett, C., Kochenderfer, M.: Toward scalable verification for safety-critical deep networks. Technical report (2018). https://arxiv.org/abs/1801.05950
53. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. Technical report (2016). http://arxiv.org/abs/1607.02533
54. Lekharu, A., Moulii, K., Sur, A., Sarkar, A.: Deep learning based prediction model for adaptive video streaming. In: Proceedings 12th International Conference on Communication Systems & Networks (COMSNETS), pp. 152–159. IEEE (2020)
55. Li, W., Zhou, F., Chowdhury, K.R., Meleis, W.: QTCP: adaptive congestion control with reinforcement learning. IEEE Trans. Netw. Sci. Eng. **6**(3), 445–458 (2018)
56. Li, Y.: Deep reinforcement learning: an overview. Technical report (2017). http://arxiv.org/abs/1701.07274
57. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. Technical report (2017). http://arxiv.org/abs/1706.07351
58. Loquercio, A., Segu, M., Scaramuzza, D.: A general framework for uncertainty estimation in deep learning. In: Proceedings International Conference on Robotics and Automation (ICRA), pp. 3153–3160 (2020)
59. Low, S., Paganini, F., Doyle, J.: Internet congestion control. IEEE Control Syst. Mag. **22**(1), 28–43 (2002)
60. Lukina, A., Schilling, C., Henzinger, T.A.: Into the unknown: active monitoring of neural networks. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 42–61. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88494-9_3
61. Lyu, Z., Ko, C.Y., Kong, Z., Wong, N., Lin, D., Daniel, L.: Fastened crown: tightened neural network robustness certificates. In: Proceedings 34th AAAI Conference on Artificial Intelligence (AAAI), pp. 5037–5044 (2020)
62. Ma, J., Ding, S., Mei, Q.: Towards more practical adversarial attacks on graph neural networks. In: Proceedings 34th Conference on Neural Information Processing Systems (NeurIPS) (2020)

63. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. Technical report (2017). http://arxiv.org/abs/1706.06083
64. Mammadli, R., Jannesari, A., Wolf, F.: Static neural compiler optimization via deep reinforcement learning. In: Proceedings 6th IEEE/ACM Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar), pp. 1–11 (2020)
65. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings 15th ACM Workshop on Hot Topics in Networks (HotNets), pp. 50–56 (2016)
66. Mao, H., Netravali, R., Alizadeh, M.: Neural adaptive video streaming with Pensieve. In: Proceedings Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), pp. 197–210 (2017)
67. Mnih, V., et al.: Playing Atari with deep reinforcement learning. Technical report (2013). https://arxiv.org/abs/1312.5602
68. Moore, A.: Efficient Memory-based Learning for Robot Control. University of Cambridge (1990)
69. Nagle, J.: Congestion control in IP/TCP internetworks. ACM SIGCOMM Comput. Commun. Rev. **14**(4), 11–17 (1984)
70. Okudono, T., Waga, M., Sekiyama, T., Hasuo, I.: Weighted automata extraction from recurrent neural networks via regression on state spaces. In: Proceedings 34th AAAI Conference on Artificial Intelligence (AAAI), pp. 5037–5044 (2020)
71. Ortega, L., Cabañas, R., Masegosa, A.: Diversity and generalization in neural network ensembles. In: Proceedings 25th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 11720–11743 (2022)
72. Osband, I., Aslanides, J., Cassirer, A.: Randomized prior functions for deep reinforcement learning. In: Proceedings 31st International Conference on Neural Information Processing Systems (NeurIPS), pp. 8617–8629 (2018)
73. Ostrovsky, M., Barrett, C., Katz, G.: An abstraction-refinement approach to verifying convolutional neural networks. In Proceedings 20th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 391–396 (2022)
74. Ovadia, Y., et al.: Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In: Proceedings 33rd Conference on Neural Information Processing Systems (NeurIPS), pp. 14003–14014 (2019)
75. Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., Song, D.: Assessing generalization in deep reinforcement learning. Technical report (2018). https://arxiv.org/abs/1810.12282
76. Polgreen, E., Abboud, R., Kroening, D.: Counterexample guided neural synthesis. Technical report (2020). https://arxiv.org/abs/2001.09245
77. Prabhakar, P.: Bisimulations for neural network reduction. In: Finkbeiner, B., Wies, T. (eds.) VMCAI 2022. LNCS, vol. 13182, pp. 285–300. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94583-1_14
78. Prabhakar, P., Afzal, Z.: Abstraction based output range analysis for neural networks. Technical report (2020). https://arxiv.org/abs/2007.09527
79. Riedmiller, M.: Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 317–328. Springer, Heidelberg (2005). https://doi.org/10.1007/11564096_32

80. Rotman, N., Schapira, M., Tamar, A.: Online safety assurance for deep reinforcement learning. In: Proceedings 19th ACM Workshop on Hot Topics in Networks (HotNets), pp. 88–95 (2020)
81. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Proceedings 27th International Joint Conference on Artificial Intelligence (IJCAI) (2018)
82. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. Technical report (2017). http://arxiv.org/abs/1707.06347
83. Seshia, S., et al.: Formal specification for deep neural networks. In: Proceedings 16th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 20–34 (2018)
84. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
85. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Technical report (2014). http://arxiv.org/abs/1409.1556
86. Singh, G., Gehr, T., Puschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proceedings 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL) (2019)
87. Sotoudeh, M., Thakur, A.: Correcting deep neural networks with small, generalizing patches. In: Workshop on Safety and Robustness in Decision Making (2019)
88. Strong, C., et al.: Global optimization of objective functions represented by ReLU networks. J. Mach. Learn., 1–28 (2021)
89. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC) (2019)
90. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (2018)
91. Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Proceedings 12th Conference on Neural Information Processing Systems (NeurIPS) (1999)
92. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. Technical report (2017). http://arxiv.org/abs/1711.07356
93. Tolstoy, L.: Anna Karenina. The Russian Messenger (1877)
94. Urban, C., Christakis, M., Wüstholz, V., Zhang, F.: Perfectly parallel fairness certification of neural networks. In: Proceedings ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), pp. 1–30 (2020)
95. Usman, M., Gopinath, D., Sun, Y., Noller, Y., Păsăreanu, C.: NNrepair: constraint-based repair of neural network classifiers. Technical report (2021). http://arxiv.org/abs/2103.12535
96. Valadarsky, A., Schapira, M., Shahaf, D., Tamar, A.: Learning to route with deep RL. In: NeurIPS Deep Reinforcement Learning Symposium (2017)
97. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: Proceedings 30th AAAI Conference on Artificial Intelligence (AAAI) (2016)
98. Vasić, M., Petrović, A., Wang, K., Nikolić, M., Singh, R., Khurshid, S.: MoËT: mixture of expert trees and its application to verifiable reinforcement learning. Neural Netw. **151**, 34–47 (2022)

99. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proceedings 27th USENIX Security Symposium, pp. 1599–1614 (2018)
100. Wu, H., et al.: Parallelization techniques for verifying neural networks. In: Proceedings 20th International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 128–137 (2020)
101. Wu, H., Zeljić, A., Katz, K., Barrett, C.: Efficient neural network analysis with sum-of-infeasibilities. In: Proceedings 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 143–163 (2022)
102. Xiang, W., Tran, H., Johnson, T.: Output reachable set estimation and verification for multi-layer neural networks. IEEE Trans. Neural Netw. Learn. Syst. (TNNLS) (2018)
103. Yang, J., Zeng, X., Zhong, S., Wu, S.: Effective neural network ensemble approach for improving generalization performance. IEEE Trans. Neural Netw. Learn. Syst. (TNNLS) **24**(6), 878–887 (2013)
104. Yang, X., Yamaguchi, T., Tran, H., Hoxha, B., Johnson, T., Prokhorov, D.: Neural network repair with reachability analysis. In: Proceedings 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), pp. 221–236 (2022)
105. Zelazny, T., Wu, H., Barrett, C., Katz, G.: On reducing over-approximation errors for neural network verification. In: Proceedings 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 17–26 (2022)
106. Zhang, H., Shinn, M., Gupta, A., Gurfinkel, A., Le, N., Narodytska, N.: Verification of recurrent neural networks for cognitive tasks via reachability analysis. In: Proceedings 24th European Conference on Artificial Intelligence (ECAI), pp. 1690–1697 (2020)
107. Zhang, J., Kim, J., O'Donoghue, B., Boyd, S.: Sample efficient reinforcement learning with REINFORCE. Technical report (2020). https://arxiv.org/abs/2010.11364
108. Zhang, J., et al.: An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Proceedings of the 2019 International Conference on Management of Data (SIGMOD), pp. 415–432 (2019)