SCALABLE NEURAL LEARNING FOR VERIFIABLE CONSISTENCY WITH TEMPORAL SPECIFICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Formal verification of machine learning models has attracted attention recently, and significant progress has been made on proving simple properties like robustness to small perturbations of the input features. In this context, it has also been observed that folding the verification procedure into training makes it easier to train verifiably robust models. In this paper, we extend the applicability of verified training by extending it to (1) recurrent neural network architectures and (2) complex specifications that go beyond simple adversarial robustness, particularly specifications that capture temporal properties like requiring that a robot periodically visits a charging station or that a language model always produces sentences of bounded length. Experiments show that while models trained using standard training often violate desired specifications, our verified training method produces models that both perform well (in terms of test error or reward) and can be shown to be provably consistent with specifications.

1 INTRODUCTION

While deep neural networks (DNNs) have shown immense progress on diverse tasks (Sutskever et al., 2014; Mnih et al., 2015; Silver et al., 2016), they are often deployed without formal guarantees of their correctness and functionality. Their performance is typically evaluated using test data, or sometimes with adversarial evaluation (Carlini & Wagner, 2017; Uesato et al., 2018; Ebrahimi et al., 2018; Wang et al., 2019). However, such evaluation does not provide formal guarantees regarding the absence of rare but possibly catastrophic failures (Administration; Board; Ross & Swetlitz, 2018).

Researchers have therefore started investigating formal verification techniques for DNNs. Most of the focus in this direction has been restricted to feedforward networks and robustness to adversarial perturbations (Tjeng et al., 2017; Raghunathan et al., 2018b; Ko et al., 2019). However, many practically relevant systems involve DNNs that lead to sequential outputs (e.g., an RNN that generates captions for a given image, or the states of an RL agent). These sequential outputs can be interpreted as real-valued, discrete-time signals. For such signals, it is of interest to provide guarantees with respect to temporal specifications (e.g., absence of repetitions in a generated sequence, or that a generated sequence halts appropriately). Temporal logic provides a compact and intuitive formalism for capturing such properties that deal with temporal abstractions.

In this work, we focus on Signal Temporal Logic (STL) (Donzé & Maler, 2010) as the specification language and exploit its quantitative semantics to integrate a verification procedure into training to provide meaningful guarantees with regard to temporal specifications. Our approach builds on recent work (Mirman et al., 2018a; Gowal et al., 2018), which is based on propagating differentiable numerical bounds through the DNN, to include specifications that go beyond simple adversarial robustness. Additionally, we propose extensions to Mirman et al. (2018a); Gowal et al. (2018) that enable us to train auto-regressive GRUs/RNNs to certifiably satisfy desired temporal specifications. The primary focus of our work is the problem of verified training for consistency with temporal specifications rather than post-facto verification of trained networks.

To summarize, our contributions are as follows:

- We present extensions to Mirman et al. (2018a); Gowal et al. (2018) that allow us to extend verified training to novel architectures and specifications, including complex temporal specifications. To handle the auto-regressive decoder commonly used in RNN-based systems,

we propose an approach based on differentiable approximations of the non-differentiable operations.

- We empirically demonstrate the applicability of our approach to ensure verifiable consistency with temporal specifications while maintaining the ability of neural networks to achieve high accuracy on the underlying tasks across domains. In the setting of supervised learning, verified training on the train-data enables us to provide similar verification guarantees for unseen test-data.
- We show that verified training results in robust neural networks whose conformance with specifications is significantly easier to guarantee than when compared to those trained adversarially or with data augmentation.

2 Related Work

With the growing emphasis on AI safety, a large body of work on verification and verified training of neural networks has emerged. Below, we discuss the most closely related approaches to our work.

2.1 Neural Network Verification

There has been considerable recent progress on developing techniques for neural network verification, starting with the pioneering work of Katz et al. (2017), where a satisfiability-modulo-theories (SMT) solver was developed to verify simple properties for piecewise linear deep neural networks. Subsequently, several mature solvers that rely on combinatorial search (Tjeng et al., 2017; Dutta et al., 2017; Bunel et al., 2018) have helped scale these techniques to larger networks.

More recently, verification of neural networks using incomplete over-approximations – using dual based approaches (Dvijotham et al., 2018b), and propagating bounds through the neural network (Gehr et al., 2018; Shiqi et al., 2018; Weng et al., 2018a; Singh et al., 2019) – has emerged as a more scalable alternative for neural network verification. Raghunathan et al. (2018a); Wong et al. (2018) showed that folding the verification procedure into the training loop (called verified training) enables us to obtain stronger guarantees. Building on this line of work, Gowal et al. (2018) showed that training with simple interval bounds, with carefully chosen heuristics, is an effective approach towards training for verifiability. However, the focus of the above mentioned works on verified training is limited to adversarial robustness properties, and feedforward networks with monotonic activation functions. In this work, we build on Mirman et al. (2018a); Gowal et al. (2018) to consider richer specifications that capture desired temporal behavior, and extend verified training to novel architectures with non-differentiable components. Table 1 provides a comparison of the different methods developed for verified training for neural networks.

Table 1: Comparison of methods developed for training for consistency with specifications.

	Beyond ReLU	Continuous Input Features	Components with Gating	Auto-regressive components	Temporal Specifications
Raghunathan et al. (2018a); Wong et al. (2018) Wang et al. (2018b)		✓			
Gowal et al. (2018); Mirman et al. (2018a)	✓	✓			
Ghosh et al. (2018); Xiao et al. (2018) Li et al. (2017a)	✓		✓	✓	✓
Ours	✓	✓	✓	✓	✓

In an independent and concurrent work, Jia et al. (2019) develop an approach for verifying LSTMs, convolutional networks and networks with attention-mechanism. They rely on a similar approach as the one developed in our paper to propagate bounds through the softmax function, word-substitutions, and also extend bound-propagation to handle the gating mechanism. Besides this overlap, the main focus of their work is on adversarial robustness under word substitution. In contrast, we consider complex temporal specifications, and auto-regressive architectures.

2.2 SATISFACTION OF TEMPORAL PROPERTIES

Temporal Specifications While training neural networks to satisfy temporal logic specifications has been considered before, it has largely been from the perspective of encouraging RL agents to do so through a modified reward (Icarte et al., 2018b; Aksaray et al., 2016; Hasanbeig et al., 2018; Icarte et al., 2018a; Sadigh et al., 2014; Wen et al., 2017; Li et al., 2017a). Further, the temporal logic specification is used to express the task to be performed by the agent, rather than as a verifiable property of the system. Ghosh et al. (2018) encourage specification conformance during training by regularizing with a loss arising from the desired logical specification. However, the specification is enforced on specific inputs and does not guarantee that the property holds across continuous regions of the input space (e.g., all inputs in the neighborhood of a given image). Consequently, there is no guarantee over large/unenumerable sets about the absence of failures arising from violations of the specification, as is often desirable. In contrast, we train neural networks to verifiably satisfy rich temporal logic specifications across multiple domains, where the set of inputs over which we want to satisfy the specification is often large/unenumerable.

Safe RL Safe exploration methods (Garcıa & Fernández, 2015) consider explorations that do not visit unsafe states. Temporal logics, in general, permit richer specifications of desired temporal behaviors than avoiding unsafe states. Junges et al. (2016) synthesize a scheduler that limits the agent explorations to safe regions of the environment specified using probabilistic computation tree logic (PCTL). This however requires a complete specification of the environment. An alternative mechanism, called shields, is a mechanism for monitoring actions of an agent and restricting those to a safe subset that ensures conformance to specifications in linear temporal logic (Alshiekh et al., 2018) or PCTL (Jansen et al., 2018). Instead of using external mechanisms to restrict the agent choices, we incorporate temporal logic objectives into training and achieve verified training of neural network agents. Furthermore, our work is not restricted to training verifiable RL agents. We demonstrate the generality of our approach on image captioning and language generation tasks involving RNNs.

Verification using Interpretable Policies PIRL (Verma et al., 2018) and VIPER (Bastani et al., 2018) extract interpretable policies that are also amenable to verification using traditional verification techniques. These approaches first learn neural network agents and then use imitation learning to extract interpretable and verifiable policies. PIRL uses a synthesis method to search for a program in a domain-specific policy language, whereas VIPER performs distillation to extract decision tree policies. Instead of extracting and then verifying abstractions of neural network policies, we directly train the neural networks verifiably. Wang et al. (2018a) analyze the robustness of RNNs by distilling them to a Deterministic Finite Automaton (DFA), and demonstrate successful distillation on recognizing Tommika Grammar. However, distillation to simple verifiable structures is often difficult, and it remains an open challenge to distill DNNs trained on tasks in the domains of vision and language processing. Further, this results in no guarantees about the original network itself. In contrast, we are able to provide guarantees for the network itself, by folding the verification problem into training.

3 FORMULATING TEMPORAL CONSTRAINTS ON ML MODELS WITH SIGNAL TEMPORAL LOGIC

We consider the problem of verified training of a model with respect to a desired property. In what follows, we describe how signal temporal logic provides a formalism to describe properties of interest.

3.1 PRELIMINARIES

We use Signal Temporal Logic (STL) (Donzé & Maler, 2010), an extension of Linear Temporal Logic (LTL) (Pnueli, 1977) that can reason about real-valued signals.

Syntax and Qualitative Semantics STL has the following syntax:

$$\varphi := \texttt{true} \mid q(s) \geq 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 \tag{1}$$

where `true` is the Boolean constant for truth, and $\neg$ and $\wedge$ are Boolean negation and conjunction operators. The symbol q is a quantifier-free non-linear real-valued arithmetic function over the

vector-valued state denoted by s; the formula $q(s) \geq 0$ is called an atom. The formula $\varphi_1 \mathcal{U}_I \varphi_2$ is the *until* temporal operator, meaning φ_1 holds until φ_2 holds in the interval I. We define $\Diamond_I \varphi$ (meaning φ *eventually* holds in the interval I) as $\texttt{true} \mathcal{U}_I \varphi$ and $\Box_I \varphi$ (meaning φ *always* holds in I) as $\neg \Diamond_I \neg \varphi$.

In this work, we interpret STL formulae over a trace σ which is a discrete-time, vector-valued signal; σ_t denotes the value of the signal at time t. We write $(\sigma, t) \models \varphi$ to indicate that φ holds for σ at time t. An atom $q(s) \geq 0$ holds at time t if $q(\sigma_t) \geq 0$. Trivially, $(\sigma, t) \models \texttt{true}$ always holds. An until formula $\varphi_1 \mathcal{U}_I \varphi_2$, with $I = [a, b]$, holds at a time instance t if $(\sigma, t') \models \varphi_2$ for some time $t' \in [t + a, t + b]$ and $(\sigma, t'') \models \varphi_1$ for all $t'' \in [t, t']$. In the rest of the paper, we use $t + I$ to denote $[t + a, t + b]$ for $I = [a, b]$. A formula φ is said to be *satisfied* over a trace σ if $(\sigma, 0) \models \varphi$. In this work, we restrict I to be bounded-intervals of time. We refer the reader to Donzé & Maler (2010) for a detailed introduction to the semantics of STL specifications.

While bounded-time STL properties can be unrolled through time into logical properties using the Boolean conjunction and disjunction operators (Raman et al., 2015), STL provides a succinct and intuitive notation for expressing desired temporal properties. In contrast with prior work on verified training that only considers adversarial robustness (a linear constraint on the logits), we consider general specifications that assert temporal properties over input-output behaviors of neural networks. Section 3.2 lists several examples of relevant properties that can be expressed in bounded-time STL, and go beyond adversarial robustness.

3.2 STL Specifications for Learning Tasks

To illustrate our approach, we consider three temporal properties of interest expressed in bounded-time STL that we want our networks to satisfy.

3.2.1 Bounding Caption Length for Image-Captioning

Figure 1: An MMNIST Image

Multi-MNIST images consist of non-overlapping MNIST digits on a canvas of fixed size (Figure 2). The number of digits in each image varies between 1 and 3. The task is to label the sequence of digits in the image, followed by an end of sequence token. Prior work on this task (Wang et al., 2019) has shown image-to-sequence models to be vulnerable to generating sequences longer than the true number of digits in the image, under small adversarially chosen perturbations. Here, we consider the task of training a network that does not output sequences longer than the desired length, while achieving similar nominal performance on unperturbed inputs.

Let $y := f(x)$ be the sequence of logits output by the RNN model when given input image x. For an image x, the termination specification is formalized as follows:

$$\forall \Delta x \in \{s : \|s\|_\infty \leq \epsilon\}.(f(x + \Delta x), 0) \models \varphi_x, \tag{2}$$

where $\varphi_x(y) := \Diamond_{[0, t_x^*]} \bigwedge_{i \neq e} (y[t]_e - y[t]_i) \geq 0$, t_x^* is the true number of digits in the image x, e is the label corresponding to the end of sequence token, $\epsilon > 0$ is the perturbation bound. Informally, this specification enforces that the end of sequence token is output no later than after the true number of digits have been output by the RNN, for all inputs within ϵ distance from a true-image.

3.2.2 Verifying that a Robot Never Runs Out of Charge

To demonstrate our approach in the RL setting, we consider a task with a vacuum cleaning robot. We summarize this task, and provide additional details in Appendix E. The agent (robot) operates in a continuous domain with its location in $(x, y) \in [0, 25]^2$ (Figure 3, Appendix). The room is divided into discrete cells, and the agent gets a reward for visiting any "dirty" cell which has not been visited in the previous T_{dirt} time-steps. The agent must visit one of the recharge cells every T_{recharge} time-steps, or the episode is terminated with no further reward. The policy maps observations (of the agent location and a map of the room) to continuous velocity controls. We use f_θ to denote the result of applying the policy, parameterized by θ, followed by the environment update. For this agent, we want to verify the specification:

$$\forall z \in S_\epsilon.(f_\theta(z), 0) \models \Box_{[0,T]} \Diamond_{[0, T_{\text{recharge}}]} \varphi_{\text{recharge}},$$

where $\varphi_{\text{recharge}}$ corresponds to the agent being in one of the recharge cells. S_ϵ corresponds to the states (x, y) within a l_∞ distance of ϵ from the centre of each of the cells. For a cell with center (x_c, y_c), this corresponds to the set of positions for the agent (x_a, y_a) such that $\|(x_a - x_c, y_a - y_c)\|_\infty \leq \epsilon$. This specification ensures that, for all starting positions of the agent in S_ϵ, for every time-step in the interval $[0, T]$, the agent recharges itself at least once within the next T_{recharge} time-steps.

3.2.3 Verifying Generated Outputs from a Language Model

A common failure mode for language models is their tendency to fall into degenerate loops, often repeating a stop-word (Wang et al., 2019). To illustrate the applicability of STL specifications in this setting, we show how to formalize the property that a GRU language model does not repeat words consecutively. We call this specification bigram non-repetition. More concretely, the desired specification is that the output sequence does not contain bigram repetition amongst the 100 most frequent tokens in the training corpus vocabulary.

We want to verify this property over a large set of possible conditioning inputs for the generative model. Concretely, we define an input set S of roughly 25 million prefixes generated from a syntactic template with the following syntax: `<pronoun>`, `<person>`, `<action-verb>`, `<connector>`, `<pronoun>`, `<person>`, `<action-verb>`. For example, one such prefix is: '*Our lord yielded and their king left*'. These prefixes are used as input to the LM, and we evaluate the specification on the outputs the model then produces. More details on the prefix template can be found in Appendix E.

Now, consider a prefix x and the sequence of logits y output by the recurrent GRU network f (i.e. $y = f(x)$), with $y(t)_k$ referring to the logit corresponding to the k^{th} most-frequent token in the vocabulary at time t. The formal specification φ_{bigram} ruling out bigram repetition can be compactly written as:

$$\varphi_{\text{bigram}} := \square_{[0,T_{\text{sample}}]} \bigwedge_{i=1,2,\ldots,100} \left(\left(\bigwedge_{j \neq i} y(t)_i \geq y(t)_j \right) \rightarrow \Diamond_{[0,1]} \neg \left(\bigwedge_{j \neq i} y(t)_i \geq y(t)_j \right) \right) \tag{3}$$

where T_{sample} denotes the length of the generated sample, in our case 10. The RNN f is required to satisfy the specification $\forall x \in S. (f(x), 0) \models \varphi_{\text{bigram}}$.

4 Scalable Training of Neural Networks Given STL Specifications

We consider the problem of learning a trace-valued function f_θ to verifiably satisfy a specification of the form $\forall x \in S. \, (f_\theta(x), 0) \models \varphi$ where x denotes an input ranging over an input space S, and $f_\theta(x)$ is the trace generated by f when evaluated on input x, θ represents the trainable parameters of f, and φ is an STL specification. Formally, our problem statement is as follows:

Given a set of inputs S, train the parameters θ of f_θ so that $\forall x \in S. \, (f_\theta(x), 0) \models \varphi$, where φ is a bounded-time STL specification.

We sometimes drop the parameters θ for brevity when denoting f.

4.1 Optimization Formulation of STL Verification

For an STL specification φ, its quantitative semantics can be used to construct a function $\rho(\varphi, f(x), t)$ whose scalar valued output is such that $\rho(\varphi, f(x), t) \geq 0 \iff (f(x), t) \models \varphi$ (Donzé & Maler, 2010). In terms of the quantitative semantics, the verification problem is equivalent to showing that $\forall x \in S. \, \rho(\varphi, f(x), 0) \geq 0$. This verification task can be written as the optimization problem of finding the sequence of inputs x such that the sequence of outputs $f(x)$ result in the strongest violation of the specification with regard to the quantitative semantics:

$$\min_x \rho(\varphi, f(x), 0) \text{ subject to } x \in S. \tag{4}$$

If the solution of this optimization problem is negative, then there exists an input x that leads to a violation of the specification φ.

4.2 BOUND PROPAGATION

The optimization problem in equation 4 itself is often intractable; even in the case when the specification is limited to robustness against perturbations in a classification task, it is NP-hard (Katz et al., 2017). There are tractable approaches to bounding the problem in equation 4 (Raghunathan et al., 2018a; Dvijotham et al., 2018b), but the bounds are often too loose to provide meaningful guarantees. To obtain a tighter bound tractably, interval bound propagation – which by itself provides loose bounds, but is efficient to compute – can be leveraged for verified training to give meaningful bounds on robustness under l_∞ perturbations (Mirman et al., 2018b; Gowal et al., 2018).

Our general approach for doing bound propagation on the function f is to use standard interval arithmetic. While this is straightforward when f is a feedforward neural network (Gowal et al., 2018; Dvijotham et al., 2018a), we demonstrate how to extend bound propagation to a much richer set of (temporal) specifications and architectures. First, we first highlight the novel aspects of bound propagation required for handling (a) auto-regressive RNNs/GRUs, (b) STL specifications and then in Section 4.3, we consider using these bounds for verified training.

4.2.1 BOUND PROPAGATION THROUGH GRUS

Computing bounds across the outputs of GRU cells involves propagating bounds through a multiplication operation (which is normally part of gating mechanisms), which can be handled by a straightforward application of interval arithmetic (Hickey et al., 2001) (see Appendix G for details).

4.2.2 BOUND PROPAGATION THROUGH AUTO-REGRESSIVE RNNS

For language modeling and image captioning, we use GRU decoders with greedy decoding. Greedy-decoding involves a composition of the `one-hot` and the `argmax` operations. Both of these operations are non-differentiable. To overcome this and compute differentiable bounds (during training), we approximate this composition with a `softmax` operator (with a low temperature T). In the limit, as $T \to 0$, the softmax operator converges to the composition $\texttt{one-hot}(\texttt{argmax}(\cdot))$ For propagating bounds through the softmax operator, we leverage that the bounds are monotonic in each of the individual inputs. Formally, given a lower and upper bound $\underline{p}$ and $\overline{p}$ on the input p to a softmax layer (i.e., $\underline{p} \leq p \leq \overline{p}$), the lower and upper bound $\underline{w}$ and $\overline{w}$ on the output can be computed as follows:

$$\overline{s} = \sum_{i=1}^{N} \exp \overline{p}_i, \quad \underline{s} = \sum_{i=1}^{N} \exp \underline{p}_i, \quad \Delta_i = \exp \overline{p}_i - \exp \underline{p}_i, \quad \overline{w}_i = \frac{\exp \overline{p}_i}{\underline{s} + \Delta_i} \quad \underline{w}_i = \frac{\exp \underline{p}_i}{\overline{s} - \Delta_i},$$

where p_i is the i^{th} coordinate of p and $p \in \mathbb{R}^N$.

During evaluation, the $\texttt{one-hot}(\texttt{argmax}(.))$ function is used as is, and the bounds are propagated through this function. Given bounds on each coordinate of p (i.e., $\underline{p} \leq p \leq \overline{p}$) and $s = \texttt{one-hot}(\texttt{argmax}(p))$, a lower bound on coordinate s_i can be computed as:

$$\underline{s}_i(x) = \begin{cases} 1 & \underline{p_i} \geq \overline{p}_j . \forall j \\ 0 & \text{otherwise.} \end{cases} \qquad \overline{s}_i(x) = \begin{cases} 0 & \exists j \neq i \text{ such that, } \underline{p}_j > \overline{p}_i \\ 1 & \text{otherwise.} \end{cases}$$

This is equivalent to the convex hull of all feasible `one-hot` vectors s, given bounds on p, and forms a sound over-approximation.

4.2.3 BOUND PROPAGATION THROUGH THE SPECIFICATION

First, we extend the quantiative semantics for STL specifications (Donzé & Maler, 2010) to allow us to reason over sets of inputs. For a STL specification φ in negation normal form (NNF) (See Appendix A for details on the quantitative semantics and conversion to NNF), we first define a *lower bound* for the quantitative semantics of φ over the set S, which we denote by $\omega_{S,f}(\varphi, 0)$. We define this bound assuming we have lower bounds on all the atoms occurring in φ. Specifically, let $\Omega_{S,f}(q, t)$ be a lower bound on $q(f(x)_t)$ over all inputs $x \in S$; in other words, at each time t we have $\forall x \in S . \Omega_{S,f}(q, t) \leq q(f(x)_t)$.

Now, we can define the lower bound on a specification φ inductively as follows:

- $\omega_{S,f}(\texttt{true}, t) = +\infty, \qquad \omega_{S,f}(\neg\texttt{true}, t) = -\infty$
- $\omega_{S,f}(q(s) \geq 0, t) = \Omega_{S,f}(q, t)$
- $\omega_{S,f}(\varphi_1 \wedge \varphi_2, t) = \min(\omega_{S,f}(\varphi_1, t), \omega_{S,f}(\varphi_2, t))$
- $\omega_{S,f}(\varphi_1 \vee \varphi_2, t) = \max(\omega_{S,f}(\varphi_1, t), \omega_{S,f}(\varphi_2, t))$
- $\omega_{S,f}(\varphi_1 \mathcal{U} \varphi_2, t) = \max\limits_{t' \in t+I} \min\Big(\omega_{S,f}(\varphi_2, t'), \min\limits_{t'' \in [t,t']} \omega_{S,f}(\varphi_1, t'')\Big)$.

Lemma 1. *For any time t, given lower bounds $\Omega_{S,f}(q, t)$ on all the atoms $q(s) \geq 0$ in φ, we have:*

$$\forall x \in S. \omega_{S,f}(\varphi, t) \leq \rho(\varphi, f(x), t)$$

Proof. See Appendix B. □

Corollary 1. *If $\omega_{S,f}(\varphi, t) \geq 0$, then $\forall x \in S.(f(x), 0) \models \varphi$.*

In order to compute the lower bounds $\Omega_{S,f}(q, t)$ required for Lemma 1, given bounds on the input x, we can first compute bounds on the outputs $f(x)_t$ at each time t. For the atoms $q(s) \geq 0$ appearing in φ, given bounds on the input s we can compute bounds on $q(s)$. These bounds can then be propagated through the specification inductively using the extended semantics above.

4.2.4 Bounds for Discrete Inputs

Tasks with discrete inputs, such as language generation tasks, encode a prefix sentence as conditioning before decoding a follow-up sequence of words. Consider prefixes of the form $x = x_0, x_1, \ldots$ such that $x_i \in S_i$, where S_i is a finite set of tokens that can appear at position i in the input sequence. We can propagate perturbations in the prefix by first projecting the tokens S_i through the embedding layer E , and then considering the maximum and the minimum value along each embedding dimension to bound the output from E. Formally,

$$\underline{E}_j(x_i) = \min_{x_i \in S_i} E_j(x_i) \leq E_j(x_i) \leq \max_{x_i \in S_i} E_j(x_i) \leq \overline{E}_j(x_i). \tag{5}$$

Jia et al. (2019); Huang et al. (2019) also consider bound propagation for word substitutions in the context of adversarial robustness.

4.3 Verified Training for STL Specifications

In this section, we describe how to train a network to satisfy an STL specification φ. The quantitative semantics $\rho(\varphi, \sigma, 0)$ gives a degree to which σ satisfies φ. First, we compute lower bounds on the values of the atoms in φ at each instance of time. Then, by application of Lemma 1, we can compute the lower bound $\omega_{S,f}(\varphi, 0)$ satisfying $\forall x \in S. \omega_{S,f}(\varphi, 0) \leq \rho(\varphi, f(x), 0)$. Subsequently we optimize the lower bound $\omega_{S,f}(\varphi, 0)$ to be non-negative, thereby guaranteeing that the specification of interest holds: $\forall x \in S. \rho(\varphi, f(x), 0) \geq 0$.

Let L_{obj} be the loss objective corresponding to the base task, for example, the cross-entropy loss for classification tasks. Training thus requires balancing two objectives: minimizing loss on the base task by optimizing $L_{obj}(f_\theta)$, and ensuring the positivity of ω_{S,f_θ}.

Due to our construction of ω_{S,f_θ}, we can use gradient descent to directly optimize the joint loss:

$$L_{obj}(f_\theta) - \lambda \omega_{S,f_\theta}(\varphi, 0),$$

where λ is a scalar hyper-parameter. In practice, to avoid having to carefully balance the two losses, the regularization from the specification loss $\omega_{S,f_\theta}(\varphi, 0)$ is clipped at a positive scalar threshold $\tau \in \mathbb{R}_+$. The weights of the neural network are learned so as to minimize:

$$L_{obj}(f_\theta) - \lambda \min\{\omega_{S,f_\theta}(\varphi, 0), \tau\}.$$

The quantitative semantics of an STL specification φ is a non-smooth function of the weights of the neural network, and is difficult to optimize directly with gradient descent over the entire set S. We find in practice that curriculum training, similar to Gowal et al. (2018), works best for optimizing the specification loss, starting with enforcing the specification over a subset $S' \subset S$, and gradually covering the entire S. Empirically, the curriculum approach means that the task performance does not degrade much with regard to L_{obj}.

Table 2: Comparison of GRU training methods on the MMNIST task. We evaluate against the termination specification on different metrics, and also report nominal accuracy. '–' indicates a trivial verified accuracy of 0% obtained with bound propagation. The entries with *Verified Termination Accuracies* corresponding to 0.0 are those where we were able to generate adversarial examples (counter-examples) to the specification for every point in the test-set. The existence of a counter-example guarantees that the specification is not satisfied, allowing us to provide an exact measure of the verified termination accuracy. We found that adversarial training is difficult because of the presence of the sigmoid & tanh activation functions commonly used in GRUs. To have a meaningful baseline, we performed adversarial training on an RNN (feedforward cells with ReLU activation). For $\epsilon = 0.1$, attacking the loss from Wang et al. (2019) to produce longer sequences performs better, while for the other ϵ values we find that adversarial training with the STL quantitative loss performs better. Adversarial training performs well but is difficult to verify. At larger ϵ, verified training results in both better guarantees (specification conformance), and better nominal accuracies.

Perturbation ϵ	Training	Nominal Accuracy	Verified Termination Accuracy	Adversarial Termination Accuracy
0.1	Verifiable	94.9	**98.3**	**100.0**
	Adversarial	94.1	–	**100.0**
	Nominal	**95.9**	–	33.5
0.2	Verifiable	94.5	**98.7**	**100.0**
	Adversarial	93.3	–	**100.0**
	Nominal	**95.9**	–	20.94
0.3	Verifiable	94.4	**98.7**	**100.0**
	Adversarial	90.0	–	99.7
	Nominal	**95.9**	0.0	0.0
0.5	Verifiable	94.1	**99.0**	**100.0**
	Adversarial	75.6	–	100.0
	Nominal	**95.9**	0.0	0.0

5 Experimental Results

5.1 Sequential Captioning of Multi-MNIST Images

For this task, we perform verified training to enforce the termination specification (equation 4) on the training data with a regularizer as discussed in Section 4.3. Post training, for unseen test-set images, we evaluate the quantitative specification loss corresponding to equation 4 with bound propagation. For an image from the test-set, if the quantitative specification loss is positive, it is guaranteed that there is no input within an l_∞ radius of ϵ around the current image that can cause the RNN to generate a longer sequence than the number of true digits in the image.

Table 3: We train the RNN with ReLU activations from (Wang et al., 2019) to be verifiable, and compare its verifiability with MILP based verification reported in Wang et al. (2019). For larger perturbations, the MILP solver times out. '–' indicates that we were unable to certify robustness for any of the points in the test-set, for the given perturbation within the time-out window of 30 minutes.

Perturbation Radius ϵ	Training	Verification Method	Verified Termination Accuracy
0.002	Nominal	MILP	83.00
	Verifiable	Bound Prop.	**99.01**
0.02	Nominal	MILP	–
	Verifiable	Bound Prop.	**98.95**
0.3	Nominal	MILP	–
	Verifiable	Bound Prop.	**94.3**

In Tables 2 and 3, *verified termination accuracy* refers to the fraction of unseen data for which we were able to verify the absence of counter-examples to the termination property as stated in equation 4. Table 2 compares verified training with nominal and adversarial training of a GRU. Verified training outperforms both adversarial and nominal training on both adversarial and verified termination accuracy metrics. The pixel values are scaled to be in the range $[0, 1]$. At perturbations of size $\epsilon = 0.5$, the images become noise; however, the network learns to be robust to such large perturbations by predicting that the image has no more than a single digit, while showing only a small degradation in nominal performance. Adversarial accuracy is evaluated with the iterative optimization-based attack described in Wang et al. (2019) (10000 steps). For $\epsilon = 0.1$, adversarial

training with the loss from Wang et al. (2019) performs better; for larger ϵ, adversarial training with the STL specification loss performs better.

Run-time Considerations As another baseline, we compare against verified termination accuracies from Wang et al. (2019) in Table 3. In Wang et al. (2019), the greedy-decoding and the bounded-time STL specification are unrolled into a MILP-query solved solved with the SCIP solver (Gleixner et al., 2018). Further, we are restricted to using ReLU activations for this comparison here because GRUs are currently not amenable to MILP solvers. Verified training results in us being able to certify specification conformance for significantly larger perturbations ($\approx$ 2 orders of magnitude larger) when compared with the MILP based verification approach.

5.2 AN RL MOBILE-ROBOT AGENT

We consider the recharging specification $\varphi_{\text{recharge}}$ over a time-horizon of $T = 10$, for all points starting within a l_∞ distance of ϵ from the center of the grids. To regularize the network to be verifiable with regard to the specification $\varphi_{\text{recharge}}$, the specification loss is obtained by rolling out the current policy through time, and propagating bounds through the rolled out policy and the environmental dynamics. We note that this propagation assumes that we have a deterministic model of the environment dynamics.

We compare our verifiably trained agent to both a vanilla RL agent, and an agent trained with reward shaping as in Li et al. (2017b) on 10^6 randomly drawn samples. All agents achieve a similar reward, and do not violate the specification on a set of 10^6 random samples from the set of feasible initial states. To compare verifiability, we discretize a region within a distance of ϵ to each cell-center into 10^2 l_∞ balls, and verify with bound-propagation that the agent satisfies $\varphi_{\text{recharge}}$ for each sub-region within a distance of ϵ from the cell centers. We find that agents with verified training are significantly more verifiable than agents trained otherwise, with almost no degradation in performance (Table 4).This observation is consistent with prior work on adversarial and verified training on classification tasks, where models not trained to be verifiable remain difficult to verify (Wong & Kolter, 2018). Regularizing for verifiability gives us formal guarantees about the DNN, using a computationally tractable algorithm, without sacrificing performance.

Table 4: Mean/Variance performance (across 5 agents of each type) across different metrics. For each agent, reward is computed as mean across 100 episodes. ϵ is distance from the center of the grid cells, and for each ϵ we report the fraction of the cells for which we are able to certify that $\varphi_{\text{recharge}}$ holds.

Training	% of cells verified ($\epsilon = 1.0$)	% of cells verified ($\epsilon = 0.1$)	% of cells verified ($\epsilon = 0.01$)	% of cells verified ($\epsilon = 0.001$)	% of cells verified ($\epsilon = 0.0001$)	Reward
Verifiable	100.0/0.0	100.0/0.0	100.0/0.0	100.0/0.0	100.0/0.0	12.71/0.19
Standard	15.8/9.0	64.9/6.8	77.6/5.3	90.3/1.8	99.2/0.0	12.85/0.06
Reward Shaping	39.1/21.9	74.3/8.6	83.2/5.9	92.0/1.9	100.0/0.0	12.76/0.22

5.3 LANGUAGE GENERATION

Our language model consists of a 2-layer GRU with 64 hidden nodes per layer, trained on the tiny Shakespeare corpus using a word embedding dimension of 32, and vocabulary truncated to the 2500 most frequent training words. We evaluate the model's ability to satisfy φ_{bigram}. We compare both a nominal model trained using log-likelihood, a model that randomly samples prefixes from the input space and penalizes violations to the specification, and verified training that covers the full input space (Details in Appendix F). We measure both test set perplexity and the count of violations observed over the full space of 25M sequences from the prefix space (Table 5).

Table 5: Language model perplexity, number of failures during an exhaustive enumerative search over the 25M perturbations, and computational cost of verification (number of forward passes).

Training	Perplexity	# Failures	# Verification Cost
Verifiable	228.91	**0**	$\approx$**2**
Sampled	174.89	**0**	2.57×10^7
Nominal	**153.63**	1.79×10^7	2.57×10^7

We find that while standard training achieves the best perplexity results, it also produces numerous specification failures. Sampling prefixes and regularizing them to avoid bigram repetition using

$\rho(\varphi_{\text{bigram}}, f(x), 0)$ eliminates failures, but the overall evaluation cost of the exhaustive search is large. Verifiable training with bound propagation, by contrast, comes with a constant computational cost of ≈ 2 forward passes. This is because matrix multiplications form a significant majority of the computational cost during a forward pass, and propagating bounds through a layer of the form $y = \sigma(Wx + b)$, where σ is a monotonic activation function (e.g. ReLU, sigmoid, tanh), can be performed in an efficient manner such that it only costs twice as much as a normal forward pass (See (Gowal et al., 2018) for a detailed analysis). In practice, these forward passes are 2-4x slower.

Run-time Considerations Verification with propagating bounds through the DNN can be performed in under **0.4 seconds** (including propagating bounds through the specification), while exhaustive search through the full space of 25M prefixes for specification violations takes over **50 minutes**. Further, as the possible word substitutions increase, the cost for exhaustive search grows exponentially while that for bound propagation stays constant.

6 Conclusion

Temporal properties are commonly desired from DNNs in settings where the outputs have a sequential nature. We extend verified training to tasks that require temporal properties to be satisfied, and to architectures such as auto-regressive RNNs whose outputs have a sequential nature. Our experiments suggest that verified training leads to networks that are substantially more verifiable, and often with fewer failures with regard to the desired specification.

Future work includes extending verification and verified training to unbounded temporal properties. Another important direction is to develop better bound propagation techniques that can be leveraged for verified training. In the RL setting, a key avenue for future work is extending DNN verification/verified training to environments with stochastic dynamics, and data-driven verification for neural networks in the absence of a known model of the environment.

References

US National Highway Traffic Safety Administration. Investigation pe 16-007. https://static.nhtsa.gov/odi/inv/2016/INCLA-PE16007-7876.PDF.

Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. *CoRR*, abs/1609.07409, 2016. URL http://arxiv.org/abs/1609.07409.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pp. 2494–2504, 2018.

US National Transportation Safety Board. Preliminary report highway: Hwy18mh010. https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf.

Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 4795–4804, 2018. URL http://papers.nips.cc/paper/7728-a-unified-view-of-piecewise-linear-neural-network-verification.

Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 3–14. ACM, 2017.

Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, pp. 92–106, 2010. doi: 10.1007/978-3-642-15297-9_9. URL https://doi.org/10.1007/978-3-642-15297-9_9.

Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 92–106. Springer, 2010.

Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In Natasha Sharygina and Helmut Veith (eds.), *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pp. 264–279. Springer, 2013. doi: 10.1007/978-3-642-39799-8_19. URL https://doi.org/10.1007/978-3-642-39799-8_19.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. *CoRR*, abs/1709.09130, 2017. URL http://arxiv.org/abs/1709.09130.

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018a.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*, 2018b.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 31–36, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2006. URL https://www.aclweb.org/anthology/P18-2006.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, May 2018. doi: 10.1109/SP.2018.00058.

Shalini Ghosh, Amaury Mercier, Dheeraj Pichapati, Susmit Jha, Vinod Yegneswaran, and Patrick Lincoln. Trusted neural networks for safety-constrained autonomous control. *CoRR*, abs/1805.07075, 2018. URL http://arxiv.org/abs/1805.07075.

Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 6.0. Technical report, Optimization Online, July 2018. URL http://www.optimization-online.org/DB_HTML/2018/07/6692.html.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR*, abs/1810.12715, 2018. URL http://arxiv.org/abs/1810.12715.

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained neural fitted q-iteration. *CoRR*, abs/1809.07823, 2018. URL http://arxiv.org/abs/1809.07823.

T. Hickey, Q. Ju, and M. H. Van Emden. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068, September 2001. ISSN 0004-5411. doi: 10.1145/502102.502106. URL http://doi.acm.org/10.1145/502102.502106.

Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving Verified Robustness to Symbol Substitutions via Interval Bound Propagation. *arXiv e-prints*, art. arXiv:1909.01492, Sep 2019.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pp. 2112–2121, 2018a.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pp. 452–461, 2018b. URL http://dl.acm.org/citation.cfm?id=3237452.

Nils Jansen, Bettina Könighofer, Sebastian Junges, and Roderick Bloem. Shielded decision-making in mdps. *arXiv preprint arXiv:1807.06096*, 2018.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified Robustness to Adversarial Word Substitutions. *arXiv e-prints*, art. arXiv:1909.00986, Sep 2019.

Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. Safety-constrained reinforcement learning for mdps. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 130–146. Springer, 2016.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Ching-Yun Ko, Zhaoyang Lyu, Tsui-Wei Weng, Luca Daniel, Ngai Wong, and Dahua Lin. POPQORN: quantifying robustness of recurrent neural networks. *CoRR*, abs/1905.07387, 2019. URL http://arxiv.org/abs/1905.07387.

Xiao Li, Yao Ma, and Calin Belta. A policy search method for temporal logic specified reinforcement learning tasks. *CoRR*, abs/1709.09611, 2017a. URL http://arxiv.org/abs/1709.09611.

Xiao Li, Cristian Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pp. 3834–3839, 2017b. doi: 10.1109/IROS.2017.8206234. URL https://doi.org/10.1109/IROS.2017.8206234.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 3578–3586, 2018a.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3578–3586, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018b. PMLR. URL http://proceedings.mlr.press/v80/mirman18b.html.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pp. 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018a.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *CoRR*, abs/1811.01057, 2018b. URL http://arxiv.org/abs/1811.01057.

Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. Reactive synthesis from signal temporal logic specifications. In Antoine Girard and Sriram Sankaranarayanan (eds.), *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pp. 239–248. ACM, 2015. doi: 10.1145/2728606.2728628. URL https://doi.org/10.1145/2728606.2728628.

Casey Ross and Ike Swetlitz. Ibm's watson supercomputer recommended 'unsafe and incorrect'cancer treatments, internal documents show. *Stat News https://www. statnews. com/2018/07/25/ibm-watson-recommended-unsafe-incorrect-treatments*, 2018.

Dorsa Sadigh, Eric S. Kim, Samuel Coogan, S. Shankar Sastry, and Sanjit A. Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*, pp. 1091–1096. IEEE, 2014. doi: 10.1109/CDC.2014.7039527. URL https://doi.org/10.1109/CDC.2014.7039527.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. volume abs/1602.07868, 2016. URL http://arxiv.org/abs/1602.07868.

Wang Shiqi, Kexin Pei, Whitehouse Justin, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *32nd Conference on Neural Information Processing Systems (NIPS)*, Montreal, Canada, 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, January 2019. ISSN 2475-1421. doi: 10.1145/3290354. URL http://doi.acm.org/10.1145/3290354.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666*, 2018.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. *arXiv preprint arXiv:1804.02477*, 2018.

Chenglong Wang, Rudy Bunel, Krishnamurthy Dvijotham, Po-Sen Huang, Edward Grefenstette, and Pushmeet Kohli. Knowing when to stop: Evaluation and verification of conformity to output-size specifications. In *CVPR*, 2019.

Qinglong Wang, Kaixuan Zhang, Xue Liu, and C. Lee Giles. Verification of recurrent neural networks through rule extraction. *CoRR*, abs/1811.06029, 2018a. URL http://arxiv.org/abs/1811.06029.

Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. *CoRR*, abs/1811.02625, 2018b. URL http://arxiv.org/abs/1811.02625.

Min Wen, Ivan Papusha, and Ufuk Topcu. Learning from demonstrations with high-level side information. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 3055–3061, 2017. doi: 10.24963/ijcai.2017/426. URL https://doi.org/10.24963/ijcai.2017/426.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for ReLU networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5276–5285, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018a. PMLR. URL http://proceedings.mlr.press/v80/weng18a.html.

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. *arXiv preprint arXiv:1804.09699*, 2018b.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5283–5292, 2018.

Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. *arXiv preprint arXiv:1805.12514*, 2018.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Kai Y Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. *arXiv preprint arXiv:1809.03008*, 2018.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *CoRR*, abs/1811.00866, 2018. URL http://arxiv.org/abs/1811.00866.

Appendices

A STL Quantitative Semantics

In addition to the qualitative semantics discussed in the main text, STL formulae have quantitative semantics (Donzé et al., 2013; Donzé & Maler, 2010) defined inductively by the function ρ below. For a given trace σ, with σ_t indicating the value of the signal at time t, the quantitative semantics is given by:

- $\rho(\texttt{true}, \sigma, t) = +\infty$
- $\rho(q(s) \geq 0, \sigma, t) = q(\sigma_t)$
- $\rho(\neg\varphi, \sigma, t) = -\rho(\varphi, \sigma, t)$
- $\rho(\varphi_1 \wedge \varphi_2, \sigma, t) = \min\left(\rho(\varphi_1, \sigma, t), \rho(\varphi_2, \sigma, t)\right)$
- $\rho(\varphi_1 \,\mathcal{U}_I\, \varphi_2, \sigma, t) = \max\limits_{t' \in t+I} \min\left(\rho(\varphi_2, \sigma, t'), \min\limits_{t'' \in [t,t']} \rho(\varphi_1, \sigma, t'')\right)$

One can obtain the qualitative semantics from the sign of the quantitative semantics. Specifically, $(\sigma, t) \models \varphi \iff \rho(\varphi, \sigma, t) \geq 0$.

We can convert the formulae to their equivalent negation normal form by following the standard procedure until negations are only associated with atoms and Boolean constants. In particular, we interpret $\rho(\neg\texttt{true}, \sigma, t) = -\infty$ and use the disjunction operator defined as $\rho(\varphi_1 \vee \varphi_2, \sigma, t) = \max(\rho(\varphi_1, \sigma, t), \rho(\varphi_2, \sigma, t))$. The normal form for $\neg(\varphi_1 \mathcal{U}_I \varphi_2)$ is obtained by pushing the negation into the subformulae, and swapping $\min$ with $\max$. Finally, we turn $\neg(q(s) \geq 0)$ into $-q(s) > 0$ which we approximate by $-q(s) - \delta \geq 0$ for some small $\delta > 0$.

B Proof of Theorem 1

Proof. We proceed by induction on φ. The base cases and the conjunction case are straightforward, and the atom case follows by assumption. The disjunction case requires us to show: $\omega_{S,f}(\varphi_1 \vee \varphi_2, t) \leq \min\limits_{x \in S}(\max(\rho(\varphi_1, f(x), t), \rho(\varphi_2, f(x), t)))$. Applying the max-min inequality, the right hand side is at least $\max(\min\limits_{x \in S}(\rho(\varphi_1, f(x), t)), \min\limits_{x \in S}(\rho(\varphi_2, f(x), t))$. Then using the inductive hypotheses, we know this is at least $\max(\omega_{S,f}(\varphi_1, t), \omega_{S,f}(\varphi_2, t))$, and the case follows. The case for the $\mathcal{U}$ operator has a similar proof based on the max-min inequality. □

C Detailed comparison with related work on verification/verified training

	Beyond ReLU	Training for Spec. Satisfaction	Continuous Input Features	Components with Gating	Auto-regressive components	Temporal Specifications
Dvijotham et al. (2018b) Zhang et al. (2018)	✓	✗	✓	✗	✗	✗
Raghunathan et al. (2018a); Wong et al. (2018) Wang et al. (2018b)	✗	✓	✓	✗	✗	✗
Gowal et al. (2018); Mirman et al. (2018a)	✓	✓	✓	✗	✗	✗
Dutta et al. (2017); Tjeng et al. (2017)	✗	✗	✓	✗	✗	✗
Ghosh et al. (2018); Xiao et al. (2018) Li et al. (2017a)	✓	✓	✗	✓	✓	✓
Weng et al. (2018b)	✗	✗	✓	✗	✗	✗
Ko et al. (2019)	✓	✗	✓	✓	✗	✗
Ours	✓	✓	✓	✓	✓	✓

Table 6: Comparison of methods developed for enforcing/checking consistency with specifications.

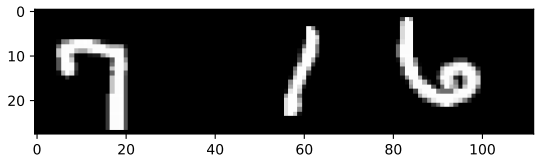

Figure 2: Sample MMNIST Image.

D MMNIST

Figure 2 depicts a sample image from the MMNIST dataset. For this task, for the GRU decoder, we train a 2 layer-convolution network with 32 filters in each layer, and the GRU decoder has 2 cells, each with a latent space of 64 dimensions. Further, to help training with the specification stabilize, we normalize the weights in the linear-layers of the GRU with the l_1 norm (similar to the Salimans & Kingma (2016), where the weights are normalized with the l_2-norm. For the curriculum training, we first train on the original task for 5000 steps to get the full clean accuracy, and then train with the specification loss for another 300000 steps to regularize the model to satisfy the specification. During this phase, we gradually ramp up the perturbation radius ϵ to the desired value, and also simultaneously increase the coefficient λ. The normalizing prevents the bounds from getting large during the wamring-up phase of training.

For adversarial training, we use a 7-step PGD attack from Madry et al. (2017), and we found that a similar curriculum helps stabilize adversarial training. Adversarial training with the largest perturbation radius as in Madry et al. (2017) degraded the performance on the nominal task signficiantly, while curriculum based adversarial training degrades performance to a much lesser extent

For the test-train data split on this task, we use the same as in Wang et al. (2019).

E RL AGENT

The agent's observation at each time-step t contains i) its own coordinates (x_t, y_t), ii) for each cell, the time remaining until that cell is dirty, and iii) the (fixed) locations of the recharge cells. The agent learns a policy from these observations to a continuous control action $(a_{x,t}, a_{y,t}) \in \mathbb{R}^2$. The continuous part of the agent's state is updated as:

$$x_{t+1} = x_t + a_{x,t}, \qquad y_{t+1} = y_t + a_{y,t}. \tag{6}$$

The policy is represented in the parameters θ, and the result of applying the policy then the environment update is our function f_θ.

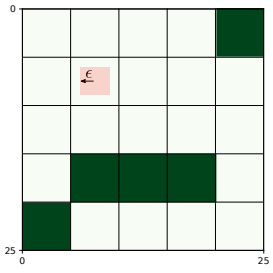

Figure 3: Domain for the robot. Recharge cells in green.

Here, we consider verifying an agents trained with Deep-Q learning in an environment with $T_{\text{recharge}} = 3$ and $T_{\text{dirt}} = 4$, i.e, every cell accumulates dust four time-steps after it was cleaned, and the robot needs to recharge itself every 4 time-steps. Additionally, the initial cell where the robot starts can have between $0 - 0.1\%$ uncertainty in the amount of dirt, the charge that can be acquired from the recharge station, and the initial battery (as another element of uncertainty in the initial position).

If the agent leaves the domain, it is clipped back into the problem domain. The agent's each have 4 discrete actions and are trained with a combination of vanilla Deep-Q learning, Deep-Q learning + reward-shaping, and Deep-Q learning + verifiable training. The actions corresponds to velocities in the 4 cardinal directions, i.e., $\{(0, 5), (5, 0), (-5, 0), (0, -5)\}$. For the agent trained to be verifiable, in addition to loss from Deep-Q learning, the agent is also trained to be verifiable with respect to the temporal specification presented in Section 3.2.2. The verification losses are optimized using the Adam optimizer Kingma & Ba (2015) with learning rate 10^{-3}. The weight of the verification loss anneals linearly between 0 and 1.5 during the first 70K steps. The model is trained to be verifiable starting at the center of the grids, and eventually covering the region around the centers, with $\epsilon = 0.015$ during the same 70k steps. We find that this regularizes the model to be verifiable in regions outside the region in which the model was trained to be verifiable.

F SHAKESPEARE

For the text generation task, we train a "seq2seq" model of Wu et al. (2016) with a GRU core. We use the standard test-train split for this task. The reconstruction and verification losses are optimized using the Adam optimizer (Kingma & Ba, 2015) with an initial learning rate of 10^{-1}. The learning is decayed every 1K steps by a factor 10, till it reaches 0.001. The weight of the verification loss which is computed from the specification in Section 3.2.3 varies linearly between 20 and 1 during 100000 steps. Finally, the verification loss is clipped between below 10. The gradients during optimization are clipped at 0.1 to prevent exploding gradients from the differentiable approximations of `one-hot(softmax(.))`

F.1 PREFIX TEMPLATE

The Language Model prefixes were defined by the following syntax: `<pronoun>, <person>, <action-verb>, <connector>, <person>, <pronoun>, <action-verb>` , where

`<pronoun>` = {*'my', 'your', 'his', 'her', 'our', 'their'*}

`<person>` = {*'sister', 'brother', 'father', 'mother', 'son', 'daughter', 'king', 'queen', 'knight', 'noble','lord','duke', 'duchess', 'cousin', 'palace', 'widow','nurse', 'marshal', 'archbishop', 'mayor', 'maid'*}

`<action-verb>` = {*['changed','despised', 'loved', 'married', 'accused', 'anointed', 'danced', 'rejoiced', 'killed', 'came', 'left', 'prayed', 'stood', 'read', 'consorted', 'denied', 'condemned', 'ruled', 'proved', 'parted' 'resolved', 'committed', 'raised', 'urged', 'painted', 'provoked', 'lived', 'charged', 'yielded', 'accursed', 'assured'],* }

`<connector>` = {*'but', 'while', 'yet', 'and', 'because']*}

The space of combinations holds 25779600 possibilities to condition the language model generation upon.

G BOUND PROPAGATION THROUGH f

We describe how to perform bound propagation for a general recurrent neural network. The neural network takes as input x and produces a sequence of outputs y_τ for $\tau = 0, ..., K$ so the overall output is $(y_0, y_1, \ldots, y_K)$. We assume that we are given bounds on the input x

$$l_0 \leq x \leq u_0.$$

Our goal is to obtain bounds on y_τ given bounds on x for each τ. Each output is produced conditioned on the preceding outputs: y_τ depends on $y_0, \ldots, y_{\tau-1}$.

We proceed recursively, assuming that we have already computed bounds on $y_0, \ldots, y_{\tau-1}$. We stack the set of inputs to the computation as $(x, y_0, \ldots, y_{\tau-1}) \in [l_{\tau;0}, u_{\tau;0}]$. We study the computation graph mapping these inputs to the output y_τ. At each node in this computation graph, we perform a computation of the form

$$z_{\tau,i} = w_i^T h\left(z_{\tau;-i}\right) + \tilde{w}_i^T \tilde{h}\left(z_{\tau;-i}\right)^T z_{\tau;-i} + b_i$$

where $h, \tilde{h}$ are element-wise nonlinear operations (sigmoid, tanh, relu etc.) and $z_{\tau;-i}$ denotes the elements of computational graph that are ancestors of the node i. The second term represents multiplicative interactions (gating interactions) common in recurrent networks like LSTMs and GRUs. Suppose we have already computed lower and upper bounds $l_{\tau;-i}, u_{\tau;-i}$ on the preceding elements. Then, we have

$$
\begin{aligned}
z_{\tau,i} &\geq \max\left(w_i, 0\right)^T h\left(l_{\tau;-i}\right) + \min\left(w_i, 0\right)^T h\left(u_{\tau;-i}\right) \\
&\quad + 1^T \min \begin{pmatrix} \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(l_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(l_{\tau;i}\right), \\ \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(u_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(u_{\tau;i}\right) \end{pmatrix} \\
z_{\tau,i} &\leq \max\left(w_i, 0\right)^T h\left(u_{\tau;-i}\right) + \min\left(w_i, 0\right)^T h\left(l_{\tau;-i}\right) \\
&\quad + 1^T \max \begin{pmatrix} \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(l_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(l_{\tau;i}\right), \\ \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(u_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(u_{\tau;i}\right) \end{pmatrix}
\end{aligned}
$$

Setting $l_{\tau,i}$ to the lower bound above and $u_{\tau,i}$ to the upper bound, we have computed bounds on $z_{\tau,i}$. Thus, we can recursively compute bounds until we obtain bounds $l_\tau \leq y_\tau \leq u_\tau$, which can then be used to compute bounds on $y_{\tau+1}$. Proceeding recursively, we obtain lower and upper bounds on $(y_0, y_1, \ldots, y_K)$.