Recurrent Neural Network Properties and their Verification with Monte Carlo Techniques

Dmitry Vengertsev[1], **Elena Sherman**[1]
[1] Department of Computer Science, Boise State University
1910 University Drive, Boise, Idaho 83725
dmitryvengertsev@u.boisestate.edu, elenasherman@boisestate.edu

Abstract

As RNNs find its applications in medical and automotive fields, they became a part of critical systems, which traditionally require thorough verification processes. In this work we present how RNNs behaviors can be modeled as labeled transition systems and formally define a set of state and temporal safety properties for such models. To verify those properties we propose to use the Monte Carlo approach and evaluate its effectiveness for different type of properties. We perform empirical evaluation on two RNN models to determine to what extent they satisfy the properties and how many the samples of Monte Carlo required to provide accurate results. Our experiments show that our models satisfy better state properties than temporal properties. In addition, we show that Monte Carlo sampling is quite effective for state property verification, which commonly requires a small fraction of RNN's model to be explored. However, for verification of temporal properties Monte Carlo needs to analyze up to 20% of computations.

1 Introduction

Recurrent Neural Networks (RNNs) have become central components in many ubiquitous applications such as vehicle trajectory estimation (Kim et al. 2017), speech recognition (Graves, Mohamed, and Hinton 2013), machine translation and many others whose computations depend on a sequence of previously processed inputs. With proven successes in those areas, now RNNs become more prevalent in critical applications such as prediction of medical care pathways (Choi et al. 2016), human action recognition (Jain et al. 2016) and medical surgeries (Mayer et al. 2006). Therefore, as important components of safety critical applications, RNNs should undergo a verification process to ensure their safe behavior.

Current research on verifying machine learning models mainly focuses on verification of Feed Forward Deep Neural Networks (DNNs). For example, researches commonly use SMT solvers to verify some safety properties of neural networks (Katz et al. 2017; Huang et al. 2017; Pulina and Tacchella 2012; Kuper et al. 2018). However, DNNs are inherently different from RNNs, an RNN is architected to process a sequence of inputs so that the next element in the sequence is processed starting from the configuration resulted from the processing the input's predecessor. Therefore, while valuable, previous work on DNN verification might not be fully applicable to RNNs, and does not address all verification scenarios of RNNs. For instance, ReluPlex (Katz et al. 2017) can formally verify safety characteristics of neural networks using piecewise linearity property of ReLu function for systems without loops. Moreover, majority of properties defined in previous work were developed by academic researchers and not necessary practitioners, which might hinder their adaptation by industry practitioners.

In this work we focus on defining and formalizing new state and temporal properties that are specific to RNNs. Moreover, we also formalize previously defined practical properties that are applicable to NNs. In particular, we identify and formalize four state safety properties: *high confidence*, *decisiveness*, *robustness* and *coverage*, which describe desirable RNN configurations. Also, we introduce and formalize two temporal safety properties: *long-term relationship* and *memorization*, which characterize permissible series of the RNN's configurations as they appear after processing each element in the input sequence.

We define the properties in the context of a model that describes how an RNN processes an input sequence. We chose to model RNNs behaviors as a labeled state transitions system where a state represents values of RNN's inner nodes after processing a single symbol and transitions are labeled with input symbol values. Therefore, two states have a transition labeled with an input symbol if after processing this input symbol the RNN's inner node values described by the first state change to the values of the second state. In this preliminary work the proposed RNN model has no abstraction, hence it represents the concrete behavior of an RNN on all possible input sequences.

To verify the proposed properties on RNN models, we use Monte Carlo sampling since without abstraction the model has a large state space and its full verification is equivalent to exhaustive testing. Monte Carlo Sampling provides a more widely applicable and scalable alternative to property verification of stochastic systems using numerical and symbolic methods.

Monte Carlo Model Checking techniques have been successfully applied to analyze systems with large state spaces in areas such as computer networking, security, and systems biology (Grosu and Smolka 2005; Donaldson and Gilbert 2008; Nghiem et al. 2010).

In our paper, we would like to investigate whether Monte Carlo sampling is a suitable approach to verifying RNN models by investigating the percentage of state space sampling required to achieve the ground truth results. In our experiments we compute the ground truth by exhaustively exploring the entire state space of an RNN model.

We perform experiments on two large RNN systems, that we model as labeled transition systems. Next, we determine ground truth for six properties and evaluate applicability of Monte Carlo Model Checking. We use the evaluation results to answer the following research questions:

1. Do the proposed properties hold for our RNN models?
2. How efficient is Monte Carlo Model Checking for those models?
3. Does the Monte Carlo Model Checking's efficiency depend on the type of property?

The results show that not all states in RNN models have state properties, which suggest the weakness of our RNNs. Moreover, long term temporal properties have even lower satisfiability rate. This implies that the future evaluation should be performed on more complex RNNs. Our findings also indicate that Monte Carlo Model Checking can be quite efficient for verifying state properties, however, it requires more samples to converge to the ground truth for temporal properties. In addition to answering to the above research questions, our work makes the following contributions:

- It formally defines sate and temporal safety properties of RNNs.
- It implements the RNN models for two RNN systems.
- It applies Monte Carlo Model Checking to verify properties of the RNN models.

The rest of the paper is organized as follows. In Section 2, we describe the representation of an RNN as a labeled transition system. In Section 3, we describe the details of the proposed formalization of state and temporal safety properties. In section 4, we briefly introduce Monte Carlo Model Checking for the verification of the proposed representation. In Section 5, the experimental results are provided. Section 6 describes the related work, and we conclude the paper by outlining plans for future work.

2 Modeling Behavior of Recurrent Neural Networks

The first step in verification of a system process is to decide how to model the system's behavior in a way that is suitable to verify its properties. Commonly behavior of system is modeled as a state transition system. In this work we chose to model RNN behavior as a Labeled Transition System (LTS) (Tretmans 1996). An LTS consists of states and labeled transitions among these states, where labels identify some action performed on the system. The LTS formalism is used for modeling the behavior of processes, and it serves as a semantic model for various formal specification languages. Below we present the definition of LTS and then describe how we model an RNN behavior with this transition system.

A *Labeled Transition System* (Tretmans 1996) is a four-tuple (S, L, T, S_0) where:

1. S is a countable, non-empty set of *states*
2. L is countable set of *labels*
3. $T \subseteq S \times (L \cup \{\tau\}) \times S$ is a *transition relation*, where L represents observable interactions, and a special label $\tau \notin L$ used to model internal transitions.
4. $S_0 \subseteq S$ is the set of initial states.

We need to use LTS to model an RNN, which accepts inputs $(x_1, \ldots, x_t)$, and has hidden states $(h_1^l, \ldots, h_t^l)$, and produces outputs $(y_1, \ldots, y_t)$, where $x_t \in \mathbb{1}_\sigma$ is one-hot encoded vector of a character from the training set alphabet Σ of size $\sigma = |\Sigma|$, $y_t \in \mathbb{R}^\sigma$, $h_t^l \in \mathbb{R}^m$, where l is the indicator of a hidden layer, and m is the dimension of a hidden vector. In our presentation we use n to denote the input sequence length, and R to describe the number of layers.

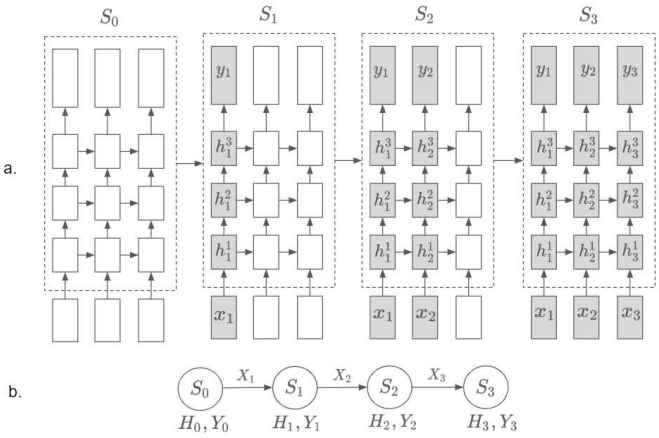

Figure 1: An example of the intuition behind state representation of an RNN with three layers (a) and corresponding RNN representation as a labeled state transitions system (b).

Similar to previous work (Du et al. 2018), an LTS state represents a configuration of an RNN. We use the tuple (H, Y) to describe such configuration, where H is the RNN's hidden state vector and Y is the output. For example (H_1, Y_1) is the state where $H_1 = (h_1^l, \bar{0}_{n-1})_{l \in R}$ and $Y_1 = (y_1, \bar{0}_{n-1})$ after processing a single symbol $X_1 = (x_1, \bar{0}_{n-1})$. We use $\bar{0}_{n-1}$notation to represent a zero value row vector of size $n-1$. Then, on the next input character $X_2 = (x_1, x_2, \bar{0}_{n-2})$, the model transitions to another state (H_2, Y_2) with $H_2 = (h_1^l, h_2^l, \bar{0}_{n-2})_{l \in R}$ and $Y_2 = (y_1, y_2, \bar{0}_{n-2})$. Fig.1 shows how an LTS models an RNN behavior on the input sequence (X_1, X_2, X_3).

Formally, an LTS for an RNN can be defined as following:

Definition 2.1. RNN behavioral model $M = (S, L, T, S_0)$, where

- $S = \{(H, Y)_i\}_{i \in N}$ is a set of *states* that defined as tuple of hidden state and the corresponding output value $(H, Y)_i$, where N is the number of states.
- $L = \{X_i\}_{i \in N}$ is a finite set of labels that is based on the input vectors X_i
- $T \subseteq S \times X \times S$ is a *transition*, i.e. every transition has an input vector X_i on it. Transitions in RNN are organized through the hidden vectors, so that output y_t at time step t becomes a function of all input vectors up to t, $\{x_1, \ldots, x_t\}$. The recurrence $(h^l_{t-1}, h^{l-1}_t) \rightarrow h^l_t$ is defined by the RNN system.
- $S_0 = (H_0, Y_0)$ is the initial state $H_0 = \mathbb{0}^{m \times n \times R}$, and $Y_0 = \mathbb{0}^{\sigma \times n}$, where $\mathbb{0}^{\sigma \times n}$ is zero matrix of size σ by n.

In such transition system we can define predicate functions that can reason about H and Y state components. Moreover, we can define a trace tr, which is a sequence of states $(S_0, S_1, S_2, \ldots, S_n)$ where $(S_i, S_{i+1}) \in T, \forall i \in (0, \ldots, n-1)$.

3 Properties of Recurrent Neural Networks

In formal verification, properties of a model are defined as formulas over atomic propositions, which are state predicate functions. Therefore, before we formalize state and temporal properties of RNNs, we need to define state predicates. To express desired properties below, we define four state predicates: Hi, Lo, Ro, and Cov.

In addition to Definition 2.1 we also use the following notations:

- $\bar{P}(.)$ is an *estimate of the confidence* with which an RNN performs prediction of Y. Often the *softmax* function is used for that purpose, however *softmax* is not a reliable approximation of confidence (Gal 2016) since it tends to provide more optimistic estimation.
- $a, b, c, d, e, r, K, z \in \mathbb{R}^+$ are positive constants that are used for predicate parametrization, i.e., thresholds.

Definition 3.1. State predicates are the following functions over a state $S = (H, Y)$

1. $Hi(a)$ - the high confidence state predicate
$$Hi(a) : \bar{P}(Y) \geq a$$
2. $Lo(b)$ - the low confidence state predicate
$$Lo(b) : \bar{P}(Y) \leq b$$
3. $Ro(r, K)$ - the robustness state predicate
$$Ro(r, K) : \|Y_j - Y_i\| \leq K\|r\|,$$
where $\forall S_i$, transitions: $(S, X_i, S_i) \in T$, $(S, X_j, S_j) \in T$, where $X_j = X_i + r$, and $r > 0$ - number of changed characters, refer to Fig.2
4. $Cov(c, z)$ - the coverage state predicate
$$Cov(c, z) : \frac{\|H > z\|}{dim(H)} \geq c,$$
where z is a threshold for a neuron to be considered activated (global for the whole network)

To define properties over a transition system, we use Linear Temporal Logic(LTL) (Pnueli 1981), which formalized reasoning about behaviors in finite-state systems. In a finite-state system LTL describes a desired behavior of a set of single computations of the tree. LTL is a powerful formalism, however, in our work we use only its fragment for which we provide a brief explanation.

LTL formulas are of the form $\boldsymbol{A}f$, where $\boldsymbol{A}f$ is a path quantifier meaning that all computation paths from a given state have property f, and f is a LTL path formula that contains only atomic propositions and formally defined as (Clarke Jr et al. 2018):

1. If $p \in AP$, then p is an LTL path formula.
2. If f is an LTL path formula, then $\neg f, \boldsymbol{X}f, \boldsymbol{F}f$, and $\boldsymbol{G}f$ are LTL path formulas.
3. If f and g are LTL path formulas, then $f \wedge g$ and $f \vee g$ are LTL path formulas.

Here $\boldsymbol{X}$ is read as "next time", $\boldsymbol{F}$ is read as "eventually", $\boldsymbol{G}$ is read as "always".

Now using these four predicates and the above logic as building blocks, we can define both state safety properties and temporal safety properties, where the former are defined on a state and the latter defined over traces, i.e., the sequences of state in an RNN behavioral model. Safety properties assert that observed behavior of the system always stays within some allowed set of finite behaviors, in which nothing bad happens. Intuitively, a property ϕ is a safety property if every violation of ϕ occurs after a finite execution of the system. In this paper we not only define them, but also provide explanations why we were motivated to include them in this work.

In all the properties below it is implied that they are of the form $\boldsymbol{A}f$, and therefore $\boldsymbol{A}$ is omitted in all the properties below.

3.1 Safety State Properties

High-Confidence With more reliance on machine learning technologies to partially automate complex decisions, the demand for high confidence prediction has grown. For example, in the case of an air traffic control application, when an AI-powered video systems is confident, it can better monitor runways, taxiways and gate areas compare to humans. However, when this system is not confident it transfers control to an air traffic operator. High confidence model requires less involvement from an operator. To describe such desired behavior, we define the High Confidence property that ensures that an RNN provides high-confidence responses on any possible input sequence.

Definition 3.2. High Confidence property: We say that M is high confidence if for the whole system the property Hi holds for all the paths globally, therefore

$$\boldsymbol{G}Hi(a) \tag{1}$$

is the formula for the verification of the whole system.

Decisiveness While the high-confidence property might not be achievable for all RNN behaviors, in those cases an RNN should provide low confidence decision to indicate the need of a human intervention. Researchers discovered that the softmax layer in a neural network tends to be "too confident" (Gal 2016; Guo et al. 2017). The estimated predicted probabilities by a softmax layer cannot always be reliably used as a confidence factor, and identifying biases towards higher confidence is critical for safe behaviors of systems.

We define the Decisiveness property as an evaluation of a given model to provide low-confidence output, when it cannot provide high-confidence output.

Definition 3.3. Decisiveness property: We say that M is decisive if for the whole system the property Lo holds when Hi does not hold for all paths globally, therefore

$$\boldsymbol{G}(\neg Hi(a) \wedge Lo(b)) \tag{2}$$

is the formula for the verification of the whole system.

Robustness Previous work shows that simple gradient methods can be susceptible to small modifications of the inputs, which results in changes to the output class (Szegedy et al. 2013). Even though the properties of neural networks related to robustness are well studied in the literature, for completeness we define them in our RNN behavioral model.

In order for an RNN to be robust to adversarial attacks, it should have internal diverse representation of hidden layers, i.e. the small perturbations in visible space r generates a robust perturbation in the latent space, and therefore produces responses $\{Y_j\}_{j\in J}$ that is close to non-perturbed response Y_i Fig.2.

Definition 3.4. Robustness property: We say that M is robust if for the whole system the property $Ro(r, K)$ holds for all paths globally, therefore

$$\boldsymbol{G}Ro(r, K) \tag{3}$$

is the formula for the verification of the whole system.

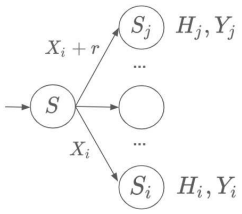

Figure 2: An example of a robust transition for the predicate $Ro(r, K)$.

Coverage Neuron coverage property is commonly used to test neural networks (Pei et al. 2017; Tian et al. 2018), but it has not been yet formally defined for verification. The coverage property is a safety property since it validates that the system response to all possible input by activating all the neurons. High coverage with removal of "dead" neurons is also a functional property, which is not discussed in our work, related to compression of neural networks via pruning, Huffman codes, representational precision reduction and quantization (Han, Mao, and Dally 2015).

We define the Coverage property as the evaluation of a given model to activate high percentage of the neurons on any possible input sequence.

Definition 3.5. Coverage property: We say that M provides adequate coverage if for the whole system the property $Cov(c, z)$ holds for all paths globally, therefore

$$\boldsymbol{G}Cov(c, z) \tag{4}$$

is the formula for the verification of the whole system.

3.2 Temporal Safety Properties

While state safety properties can be used for any NNs, temporal properties are specific for RNNs since they describe a desirable pattern of states in the traces of an RNN behavioral model. We express such patterns as regular expressions over state predicates. Similar to the state safety properties we first argue that there is a need for each temporal property and then formally define it.

Long-term Relationship One of the critical properties of RNN-based systems is the ability to capture a long-term relationship in an input sequence. A good RNN should have good predication not only on a trained length of sequences but also on longer ones. Such property is important for an RNN since for application such as a vehicle trajectory prediction (Kim et al. 2017) an RNN-based architecture is used to effectively predict the future coordinates of the surrounding vehicles, called occupancy grid. Estimation of the vehicle's occupancy grid is naturally a recurrent process, and therefore the ability to leverage long-term past actions of the surrounding vehicles is critical for vehicle safety.

We present a long-term relationship property as the evaluation of model's confidence on longer sequences in comparison to confidence on sequences it was trained on. For example, if model was trained on the sequences of length n, it is evaluated on the sequences of length $n + \rho(n)$, where $\rho(n) > 0$ is a linear function of n.

Formally, a long-term property is evaluated on a trace $tr_{n+\rho(n)} = (S_1, \cdots, S_n, S_{n+1}, \cdots, S_{n+\rho(n)}) = tr_n tr_{\rho(n)}$ to check whether this model provides u confident predictions on a trace tr_n, and on the adjacent trace $tr_{\rho(n)}$ the model's confidence did not decrease significantly, and still provided v confident predictions, where $|tr_n| = n$, $|tr_{\rho(n)}| = \rho(n)$.

Definition 3.6. Long-term Relationship property: We say that M satisfies Long-term Relationship property if for the whole system the property $Hi(a)$ holds at least u times on a trace tr_n, and property $Hi(d)$ holds v times on the adjacent trace $tr_{\rho(n)}$ for all paths globally, i.e. :

$$\begin{aligned} &\boldsymbol{G}\eta(u, v, a, d) \\ &\eta : \eta_n(u, a) \wedge \eta_{\rho(n)}(v, b) \\ &\eta_n(u,a) : \ (Hi(a)(\neg Hi(a))^*)^u \\ &\eta_{\rho(n)}(v, d) : \ (Hi(d)(\neg Hi(d))^*)^v \end{aligned} \tag{5}$$

where d is a constant threshold for property Hi on a longer sequence, such that $d \leq a$.

Memorization Memorization property of RNN-based systems was presented in literature mostly in relation to data privacy issues. In particular, with RNN models there is a risk that rare or unique training-data sequences are unintentionally memorized when such a model is trained on sensitive user data (Carlini et al. 2018).

Several methods to test the memorization property were investigated such as memory exposure measurement (Carlini et al. 2018) and inspection of gradient magnitudes in long-term context (Madsen 2019).

In terms of our representation, we define memorization property as situations when a model gives exact answer with the absolute confidence to some subsequence of states in a trace.

Definition 3.7. Memorization property: We say that M satisfies Memorization property if for the whole system the property Hi holds on all the paths globally with a very high threshold $e = 1-\epsilon$, for a small value of $\epsilon > 0$. This property can be summarized as the following regular expression:

$$\begin{gathered} \boldsymbol{G}\mu(q,e) \\ \mu \ : ((\neg Hi(e))^* Hi(e)(\neg Hi(e))^*)^q \end{gathered} \tag{6}$$

where $q < n$ is the number of memorization cases on a given trace path.

However, memorization property is not desirable, so we are checking an RNN systems for the **lack of memorization** $\boldsymbol{G}\neg\mu(q,e)$.

We now present a verification method that is used for evaluating these properties.

4 Verification of RNNs Behavioral Models

When a system under verification has a large state space, verifying it with a traditional model checking techniques becomes infeasible. Some techniques investigate various heuristics to guide a model checker through a large state state space (Henriques et al. 2012). Another promising approach deals with state space explosion through random sampling of the state space and then applying the probability theory to reason about the overall correctness of the model (Grosu and Smolka 2005). In such randomized sampling approach a model is executed repeatedly using discrete event simulations (Monte Carlo experiments) without the need for an explicit representation. The path formula ϕ is checked on each trace generated by the simulation. A checked trace provides a sample point of a Bernoulli random variable, which is 1 if ϕ holds and 0 otherwise. The model is simulated finitely many times and if a sampling finds a violation of a property then it is reported, otherwise a probability of the model to violate a property is calculated.

However, for this work we expect our systems have many paths where properties are violated. Thus the traditional Monte Carlo Model-Checking that stops on at the first property violation trace would not give us complete description of the model. In order to estimate how many the degree to which properties hold for our models, we perform sampling based on posteriori probability evaluations.

4.1 Posteriori Probability Evaluations

The idea of Monte Carlo sampling is to sample traces tr from the representation model in i.i.d. fashion. We can model $Pr(\phi) \sim Bernoulli(\rho)$ using Bernoulli distribution and apply the sampled trajectories to statistically estimate the Bernoulli success parameter ρ.

As we mentioned before, comparing to the traditional Monte Carlo-Model checking (Grosu and Smolka 2005), our Monte Carlo simulation continues sampling even in a presence of a counter-example. Therefore, for the correct evaluation of a success parameter ρ, we need to evaluate a posteriori probability on traces tr:

$$\begin{gathered} Pr\left(\rho|tr\right) = \frac{Pr(tr|\rho)Pr(\rho)}{Pr(tr)} = \\ \frac{\rho^{\sum tr_i}(1-\rho)^{n-\sum tr_i}Pr(\rho)}{Pr(tr)} \end{gathered} \tag{7}$$

Out of total number of traces n sampled so far, we denote k to be the number of traces tr that satisfy property ϕ. The conjugate prior of a Bernoulli distribution $Pr(\rho)$ is the Beta distribution $\frac{1}{Beta(\alpha,\beta)}\rho^{\alpha-1}(1-\rho)^{\beta-1}$, where α and β are shape parameters of Beta distribution. Using $\sum tr_i = k$, we can re-write the a posteriori equation (7) in the following form:

$$\begin{gathered} Pr\left(\rho|tr\right) = \frac{\rho^{\sum tr_i+\alpha-1}(1-\rho)^{n-\sum tr_i+\beta-1}Pr(\rho)}{Pr(tr)} \\ \sim Beta(\alpha+k, \beta+n-k) \end{gathered} \tag{8}$$

Since posteriori is distributed by Beta with new parameters given in (8), the estimated probability of having a trajectory satisfying property ϕ is given via estimated mean $\hat{\rho}$ and variance $\hat{\nu}$ below as:

$$\begin{gathered} Pr(\phi) = \hat{\rho} \pm \sqrt{\hat{\nu}} = \\ \frac{k+\alpha}{\alpha+\beta+n} \pm \sqrt{\frac{(\alpha+k)(n-k+\beta)}{(\alpha+n+\beta)^2(\alpha+n+\beta+1)}} \end{gathered} \tag{9}$$

For each property ϕ, we will use the convergence of the corresponding estimated probability $\hat{\rho}(n)$ with the number of samples n to decide when to stop sampling. From our empirical evaluations we compute the value of the convergence rate when ρ's values are close to the ground truth. We define *ρ-convergence* as a following:

$$\sum_{i=1}^{W} \mid \hat{\rho}(i) - \hat{\rho}(i-1) \mid /W$$

where W is the sampling window over which we compute the average between values of estimated probabilities.

5 Experiments

The goal of the evaluations is to answer the following research questions:

1. Do the proposed properties hold for our RNN models?
2. How efficient is Monte Carlo sampling for those models?
3. Does the Monte Carlo sampling efficiency depend on property types?

In this section we first describe the experiments set up, their results and the answers for the above research questions.

Table 1: The ground truth values for the ratio of properties satisfaction and the number of Monte Carlo samples required to achieve the same rate values.

Property	**Notation**	**Params**	**Ground Truth**		**Samples**		$\hat{\rho}$ **convergence**	
			M_1	M_2	M_1	M_2	M_1	M_2
High Confidence	$\boldsymbol{G}Hi(a)$	$a = 0.7$	**29.2**	20.6	5,371(0.9%)	3,055(0.5%)	8.2e-05	1.1e-04
Decisiveness	$\boldsymbol{G}(\neg Hi(a) \wedge Lo(b))$	$b = 0.2$	22.8	**26.0**	5,343(0.9%)	3,833(0.6%)	5.8e-05	1.1e-04
Robustness	$\boldsymbol{G}Ro(r, K)$	$r = 2$	39.0	40.2	2,409(0.5%)	4,655(0.8%)	2.1e-04	1.0e-04
Coverage	$\boldsymbol{G}Cov(c, z)$	$c = 0.5$	90.2	**95.7**	1,530(0.2%)	1,564(0.3%)	1.0e-04	5.1e-05
Long-term Relation	$\boldsymbol{G}\eta(u, v, a, d)$	$d = 0.65$	**9.7**	5.0	5,459(0.9%)	45,487(7.8%)	2.5e-05	1.9e-06
No memorization	$\boldsymbol{G}\neg\mu(q, e)$	$e = 0.99$	98.1	**99.6**	104,467(18.0%)	8,577(1.5%)	1.8e-07	4.2e-07

5.1 Experiment Setup

We use a character-level language model as an interpretable test for analyzing the performance of Monte Carlo Model Checking for verifying the safety properties of RNNs.

RNN Systems Without the loss of generality we consider RNN with Long-Short Term Memory (LSTM) cell (Hochreiter and Schmidhuber 1997). In the experiment, we use the next character prediction model character-RNN. When applied to the character sequence, a character-RNN predict a next character in this sequence.

We evaluate performance of two RNN systems M_1 and M_2, as well as their behavioral models representations. These systems have different numbers of hidden layers and different dimensions of hidden spaces. Table. 2 presents the summaries of the models' key parameters. We use two models in order to evaluate properties across RNNs with different number of parameters and different loss values. In particular M_1 was trained to get higher accuracy, whereas the M_2's large number of parameters suggests its potentially learning capacity.

Table 2: LSTMs and their Abstraction Models

	M_1	M_2
Hidden Layers	1	4
Hidden State Dimension, $dim(h_t^l)$	80	60
Total Parameters	43,884	115,964
Training Loss	1.70	1.84
Validation Loss	1.45	1.58
Abstraction States	580,741	580,741

Data and Verification Setup These next character RNNs are trained on Nietzsche dataset with the alphabet of size $\sigma = |\Sigma| = 44$, number of training examples $N = 580,741$, and sequence length $n = 50$. In order to evaluate the properties of the behavioral models for both systems M_1 and M_2, we first computed a ground truth. Next, we verify these models with our variation of Monte Carlo Model-Checking.

The ground truth values are computed by exhaustively evaluating all the properties on all of the states possible using for both RNN systems M_1 and M_2. The Monte Carlo simulations are performed by sampling test examples, and by running the simulation till a property satisfaction rate is within $0.1\rho_{100}$ % of the ground truth, where ρ_{100} is an average of the first hundred values of ρ which is defined in (9). We use ρ_{100} since rare properties need to have a tighter bound. The list of properties, notation and properties' parameters is presented in the first three columns of Table 1.

5.2 Results and Discussions

We evaluate properties satisfaction on TLS using our variation of the Monte Carlo Model-Checking and compare the results with the ground truth. Table 1 shows the results of properties verification with our Monte Carlo Model-Checking.

The property satisfaction ratios for the ground truth values are shown in the fourth column of Table 1. For example, M_1 only has 29.2% of states that satisfy the high confidence property. Property satisfaction ratios for High Confidence, Decisiveness, and Robustness are in the range of $(20\%, 40\%)$. These values are low, but expected, since M_1 and M_2 are simple RNNs. In addition, both models demonstrate a low satisfiability rate for the Long-term relationship property.

Even though the model M_1 has lower validation loss comparing to M_2 (Table 2), from the properties point of view, the benefit of M_1 over M_2 is ambivalent. Indeed, model M_1 has only 90% of coverage, which is an indicator of a significant number of "dead" neurons. Moreover, the model M_1 has almost 2% of memorization, which is inadequate, i.e., too high, to be considered safe.

Thus, we can answer our **first research question** that property satisfaction rates for both models are not sufficient to be considered safe, although M_2 performed well on coverage and had low memorization. One of the reason that coverage and memorization properties are better supported by the models is because there were accounted during the design of those models.

It is also interesting to see that temporal safety properties are significantly more computationally expensive compare to state safety properties. Indeed, the "Samples" columns of Table 1 shows the number of Monte Carlo samples needed to reach the ground truth. For temporal properties, one would need to sample up to 18% of all possible traces to get close to the ground truth. High verification complexity of temporal

properties is intuitive due to temporal events like memorization are applied to an entire trace and not to every state.

Thus, we can answer the **second research questions** that comparing to the entire state space Monte Carlo sampling is efficient for estimating properties of RNN models. The answer for the **third research question** is that of the state safety properties are more efficiently checked than the temporal safety properties. The difference ranges from two to four orders of magnitude required samples.

Note that Monte Carlo simulations are performed by sampling test examples, and by running the simulation until a property satisfaction rate is within $0.1\rho_{100}$ % of the ground truth. However, computing ground truth is not always computationally feasible. Monitoring the convergence of values of $\hat{\rho}(n)$ is an alternative to the ground truth calculations. From Table 1 we can see that state safety properties require convergence of order e-05, and temporal safety properties require convergence of order e-07. We can use those numbers to obtain sufficient amount of samples for evaluating property satisfaction rates of RNNs models.

6 Related Work

There have been several papers addressing the challenge of verification of the Recurrent Neural Networks. In recent work (Musau and Johnson 2018) the authors presented verification of continuous time RNNs based on reachability methods for nonlinear ordinary differential equation. Another work (Wang et al. 2018) abstracts an RNN as a Deterministic Finite Automaton in order to check a single property — the robustness of RNNs to adversarial attacks.

Another method for the robustness property analysis of RNNs is Propagated output Quantified Robustness for RNNs(POPQORN) (Ko et al. 2019). The robustness was estimated by computing two linear bounds on the output generated by a perturbed input sequence to the RNN. This method can be applied to diverse RNN structures including LSTMs and GRUs.

DeepStellar approach (Du et al. 2019) uses Markov Chain Representation of states of RNN, which limits the scale of the models due to state space explosion, probabilistic nature, and focuses on coverage metrics and traces similarity.

Critical safety properties of Deep Neural Nets (DNN) such as robustness and safety were introduced and discussed in literature (Canziani, Paszke, and Culurciello 2016). However, unlike the work presented here, those properties were neither formally defined nor verified on actual RNNs.

In the most recent work (Akintunde et al. 2019) the authors present a theoretical framework for verification of recurrent neural agent-environment systems (RNN-AES), which is an extension of the neural agent-environment systems (MAS) defined on feed-forward neural networks. This framework was practically applied in (Akintunde et al. 2018) for the analysis of reachability problem of the multi agent systems that are critical in control consensus problems and in reinforcement learning applications (Wooldridge 2009; Vengertsev et al. 2015; Brockman et al. 2016).

7 Conclusion and Future Work

In this paper, we defined the labeled state transition abstraction derived from a Recurrent Neural Network, we introduce four new safety properties and formalized six critical properties defined on the state space: high-confidence, decisiveness, coverage, robustness, long-term relationship, and memorization.

We applied the Monte Carlo Model Checking approach for verification of the proposed properties, and demonstrated that temporal safety properties require longer Monte Carlo verification compare to the state safety properties.

In future work, we will explore the approaches to improve verification of the temporal safety properties to achieve better verification performance. In order to achieve it we will develop a more sophisticated abstraction of the RNN system designed specifically for the safety temporal properties, and evaluate it on more sophisticated RNN model.

References

Akintunde, M.; Lomuscio, A.; Maganti, L.; and Pirovano, E. 2018. Reachability analysis for neural agent-environment systems. In *Sixteenth International Conference on Principles of Knowledge Representation and Reasoning*.

Akintunde, M. E.; Kevorchian, A.; Lomuscio, A.; and Pirovano, E. 2019. Verification of rnn-based neural agent-environment systems. In *Proceedings of the 33th AAAI Conference on Artificial Intelligence (AAAI19). Honolulu, HI, USA. AAAI Press. To appear*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Canziani, A.; Paszke, A.; and Culurciello, E. 2016. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

Carlini, N.; Liu, C.; Kos, J.; Erlingsson, Ú.; and Song, D. 2018. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232*.

Choi, E.; Bahadori, M. T.; Schuetz, A.; Stewart, W. F.; and Sun, J. 2016. Doctor ai: Predicting clinical events via recurrent neural networks. In Doshi-Velez, F.; Fackler, J.; Kale, D.; Wallace, B.; and Wiens, J., eds., *Proceedings of the 1st Machine Learning for Healthcare Conference*, volume 56 of *Proceedings of Machine Learning Research*, 301–318. Children's Hospital LA, Los Angeles, CA, USA: PMLR.

Clarke Jr, E. M.; Grumberg, O.; Kroening, D.; Peled, D.; and Veith, H. 2018. *Model checking*.

Donaldson, R., and Gilbert, D. 2008. A monte carlo model checker for probabilistic ltl with numerical constraints. *Dept. of Computing Science, University of Glasgow, Research Report TR-2008-282*.

Du, X.; Xie, X.; Li, Y.; Ma, L.; Zhao, J.; and Liu, Y. 2018. Deepcruiser: Automated guided testing for stateful deep learning systems. *arXiv preprint arXiv:1812.05339*.

Du, X.; Xie, X.; Li, Y.; Ma, L.; Liu, Y.; and Zhao, J. 2019.

Deep-stellar: Model-based quantitative analysis of stateful deep learning systems.

Gal, Y. 2016. Uncertainty in deep learning.

Graves, A.; Mohamed, A.-r.; and Hinton, G. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, 6645–6649. IEEE.

Grosu, R., and Smolka, S. A. 2005. Monte carlo model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 271–286. Springer.

Guo, C.; Pleiss, G.; Sun, Y.; and Weinberger, K. Q. 2017. On calibration of modern neural networks. *Proceedings of the 34th International Conference on Machine Learning-Volume 70* 1321–1330.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Henriques, D.; Martins, J. G.; Zuliani, P.; Platzer, A.; and Clarke, E. M. 2012. Statistical model checking for markov decision processes. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, 84–93. IEEE.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety verification of deep neural networks. 3—29.

Jain, A.; Zamir, A. R.; Savarese, S.; and Saxena, A. 2016. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5308–5317.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient smt solver for verifying deep neural networks. 97–117.

Kim, B.; Kang, C. M.; Kim, J.; Lee, S. H.; Chung, C. C.; and Choi, J. W. 2017. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 399–404. IEEE.

Ko, C.-Y.; Lyu, Z.; Weng, T.-W.; Daniel, L.; Wong, N.; and Lin, D. 2019. Popqorn: Quantifying robustness of recurrent neural networks. *arXiv preprint arXiv:1905.07387*.

Kuper, L.; Katz, G.; Gottschlich, J.; Julian, K.; Barrett, C.; and Kochenderfer, M. 2018. Toward scalable verification for safety-critical deep networks. *arXiv preprint arXiv:1801.05950*.

Madsen, A. 2019. Visualizing memorization in rnns. *Distill* 4.

Mayer, H.; Gomez, F.; Wierstra, D.; Nagy, I.; Knoll, A.; and Schmidhuber, J. 2006. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 543–548.

Musau, P., and Johnson, T. T. 2018. Verification of continuous time recurrent neural networks (benchmark proposal). In *ARCH@ ADHS*, 196–207.

Nghiem, T.; Sankaranarayanan, S.; Fainekos, G.; Ivancić, F.; Gupta, A.; and Pappas, G. J. 2010. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, 211–220. ACM.

Pei, K.; Cao, Y.; Yang, J.; and Jana, S. 2017. Deepxplore: Automated whitebox testing of deep learning systems. 1–18.

Pnueli, A. 1981. The temporal semantics of concurrent programs. *Theoretical computer science* 13(1):45–60.

Pulina, L., and Tacchella, A. 2012. Challenging smt solvers to verify neural networks. *AI Communications* 25(2):117—135.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tian, Y.; Pei, K.; Jana, S.; and Ray, B. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. *Proceedings of the 40th international conference on software engineering* 303–314.

Tretmans, J. 1996. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer networks and ISDN systems* 29(1):49–79.

Vengertsev, D.; Kim, H.; Seo, J. H.; and Shim, H. 2015. Consensus of output-coupled high-order linear multiagent systems under deterministic and markovian switching networks. *International Journal of Systems Science* 46(10):1790–1799.

Wang, Q.; Zhang, K.; Liu, X.; and Giles, C. L. 2018. Verification of recurrent neural networks through rule extraction. *arXiv preprint arXiv:1811.06029*.

Wooldridge, M. 2009. *An introduction to multiagent systems*. John Wiley & Sons.