CS703 Project Final Report: Verification for Recurrent Neural Networks

Yuhao Zhang [1]

Abstract

Despite the large efforts made to verify the robustness of convolutional neural networks (CNNs), the robustness of recurrent neural networks (RNNs) has seldom been studied. This project aims to tackle the challenges of verifying the robustness of RNNs. We designed a novel abstract interpretation for the GRU cells used in RNNs. Moreover, dynamic programming is used to capture the global constraint on the number of local modifications. Furthermore, to improve the precision of the verification, unions box domains have been kept within a budget during propagation through the neural network. The experiments show that our verification tool can verify the robustness of a trained RNN with 100% precision on the artificial dataset.

Prediction	Normal examples x and adversarial examples x'
+	it ' s the kind of pigeonhole-resisting romp that hollywood too rarely provides .
-	it ' s the kind of pigeonhole-resisting romp that hollywood too rarely **gives** .
World	South Africa's historic Soweto township marks its 100th birthday on Tuesday in a mood of optimism.
Sci/Tech	South Africa's historic Soweto township marks its 100th birthday on Tuesday in a moo**P** of optimism.

Table 1. Adversarial examples with a single word/character change, which will be misclassified by two neural networks.

1. Introduction

In this paper, we studied the verification for the robustness property of RNNs, which are widely used in Natural Language Processing (NLP) classification tasks. The robustness property of a neural network f for classification is defined as follows:

$$\forall x \in \mathcal{D}.\forall x'.s.t.,\ |x - x'| \leq \delta \Rightarrow f(x) = f(x')$$

, where $\mathcal{D}$ is the dataset, $|.|$ is the distance metric, δ is the global constraint of the modification, and $f(x), f(x')$ are the outputs of neural network f for input x, x'. Moreover, we define the distance metric $|.|$ as the minimal number of substitutions that modify a normal example x to an adversarial example x'. Table 1 shows two adversarial examples taken from (Ebrahimi et al., 2018; Huang et al., 2019). The first adversarial example is a word-level adversarial example with $\delta = 1$, while the second one is a character-level adversarial example with $\delta = 1$. As a result, the robustness properties of the two above neural networks are violated.

Verification for the neural network is important since the neural networks are vulnerable to adversarial examples (Carlini & Wagner, 2017), and devastating outcomes can happen if an unverified neural network is deployed in the real world. Specifically, many research efforts have been made for adversarial attacks for natural language classification tasks (Ebrahimi et al., 2018; Zhang et al., 2019). Those works proposed approaches that generate adversarial examples using substitution, insertion, or deletion to fool the neural network classifiers.

Given a normal example x, it is hard to naively test on all possible adversarial examples x', since the number of adversarial examples would be $O(L^{\delta} * |V|)$, where L is the length of the input and $|V|$ is the vocabulary size. But we can use abstract domains (such as box and zonotope) as inputs to capture the exponential number of adversarial examples, then apply abstract interpretation to propagate the abstract values through RNNs. Nevertheless, no abstract interpretation has been designed for RNNs. Thus, how to design an abstract interpretation for RNNs, especially the recurrent cells such as LSTM and GRU is unknown. To design such abstract interpretation, we will face two challenges: (1) the abstract domains are imprecise on global

[*]Equal contribution [1]Department of Computer Science, University of Wisconsin-Madison, Madison, USA. Correspondence to: Yuhao Zhang <yuhaoz@cs.wisc.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

constraint δ; (2) the abstract interpretations are imprecise as the length of recurrent layer increases.

We designed a novel abstract interpretation for GRU cells (Cho et al., 2014) used in RNNs. Moreover, we proposed two approaches to overcome two challenges stated above respectively. We used **dynamic programming** to capture the global constraint δ. Specifically, we defined $F[i][j]$ as the output of the i-th hidden state after j substitutions are made. Then we have the following transition rule:

$$\begin{aligned} F[i][j] = & \text{GRU}(F[i-1][j], x_i) \\ & \sqcup \text{GRU}(F[i-1][j-1], \text{Abstract}(x_i)) \end{aligned}$$

, where $\sqcup$ is the join of two abstract values, and the $Abstract$ function compute the abstract value for all the character/-word that x_i can substitute to. To improve the precision of propagation, we kept **unions of abstract values within a budget** b. The precision increases as we increase the budget b, while the verification speed decreases.

We implemented our verification tool using unions of abstract boxes. And we evaluated our verification tool for RNNs on an artificial dataset. The results show that our tool can precisely verify for a trained RNN and outperform the naive approaches.

We briefly summarize our contributions in this paper:

- To the best of our knowledge, this paper is the first to introduce verification for the RNN classifier for natural language classification.
- We used experiments to show that our novel abstract interpretation for the GRU cell and the two approaches proposed can (1) precisely verify for an RNN trained on the artificial dataset; (2) outperform the naive approaches on the artificial dataset.

2. Overview

The general process of verification for robustness property of neural networks is presented as follows:

1. Capture the input space under certain abstract domain
2. Do propagation through the neural network under the abstract domain using the abstract interpretations designed for all the operation in the neural network
3. Verify the abstract value obtained at the last layer of the neural network

The whole verification process is sound since the abstract value of the input space and the abstract interpretations are sound.

The overview of our verification process is presented as follows:

1. Capture the input space using **dynamic programming** under **unions of abstract values**
2. Do propagation through the neural network under **unions of abstract values** using the abstract interpretations designed for all the operation in the neural network
3. Verify the **unions of abstract values** obtained at the last layer of the neural network

We will illustrate the abstract domains and abstract interpretations for RNNs in Section 3. Moreover, we discuss the **dynamic programming** for capturing input space and the **unions of abstract values** in Section 4.

3. Abstract Domain and Abstract Interpretation

In this section, we introduce the abstract domains used in existing literature and the novel abstract interpretation designed for the GRU cells.

3.1. Abstract Domain

Abstract domains used in verification of neural networks are abstracts of concrete numerical vectors. There three abstract domains that are used for verification of neural networks (Singh et al., 2018; 2019b;a): box domain, zonotope domain, and polyhedron domain.

Box domain, also known as interval domain, maintains the lower and upper bound of each dimension in concrete numerical vectors. A box domain of vectors in $\mathbb{R}^n$ can be formally by a center $c \in \mathbb{R}^n$ and a radius $r \in \mathbb{R}^n \wedge \forall i. r_i \geq 0$. Every point in that box domain can be represented as $c + rE$, where $E = \text{diag}\{e_1, e_2, \ldots, e_n\}$ and $e_i \in [-1, 1]$. The lower bound of the vectors can be computed by $c - r$, and the upper bound, $c + r$.

Zonotope domain maintains a zonotope, known as the Minkowski sum of vectors, containing all concrete numerical vectors. A zonotope domain of vectors in $\mathbb{R}^n$ can be formally by a center $c \in \mathbb{R}^n$ and a set of vectors $r_i \in \mathbb{R}^n$. Every point in that box domain can be represented as $c + \sum_i e_i r_i$, where $e_i \in [-1, 1]$. A zonotope domain is also an extension of a box domain.

Polyhedron domain maintains a polyhedron containing all concrete numerical vectors. To compute a polyhedron explicitly is time-consuming and not necessary for verification. Existing works maintain a set of inequalities as an implicit polyhedron propagating through the neural network. In this

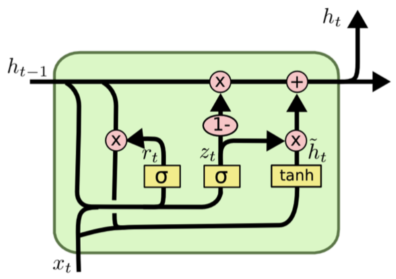


Figure 1. A GRU Cell

case, the final verification problem is equivalent to solve a mixed integer linear programming (MILP), which can be handled by a constraint solver.

3.2. Abstract Interpretation for GRU Cell

Abstract interpretation is a theory of sound approximation of the semantics of computer programs, based on monotonic functions over ordered sets, especially lattices.[1] Abstract interpretation for matrix multiplication,element-wise addition (substraction), non-linear activation functions have already been designed. However, there is no abstract interpretation for the cells in RNNs.

In this section, we describe a novel abstract interpretation for one of the cells used in RNN: the GRU cell (shown in Figure 1[2]). The GRU cell takes input as the previous hidden state and the current input, and outputs the current hidden state:

$$\begin{aligned} z_t &= \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \\ r_t &= \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \\ h'_t &= \tanh(Wx_t + r_t \odot Uh_{t-1}) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \end{aligned}$$

The abstract interpretation for element-wise addition, matrix multiplication, and activation functions have already been defined. Thus, the only undefined operation in the GRU cell is $\odot$, which is the element-wise multiplication.

We can define the abstract interpretation for $\odot$ under the box domain. To abstract $a \odot b$ to the box domain, where a and b are abstract values in the box domain, we show how to compute the upper bound and the lower bound of it, denoted as $\overline{a \odot b}$ and $\underline{a \odot b}$:

$$\overline{a \odot b} = \max(\overline{a} \odot \overline{b}, \overline{a} \odot \underline{b}, \underline{a} \odot \overline{b}, \underline{a} \odot \underline{b})$$
$$\underline{a \odot b} = \min(\overline{a} \odot \overline{b}, \overline{a} \odot \underline{b}, \underline{a} \odot \overline{b}, \underline{a} \odot \underline{b})$$

[1] https://en.wikipedia.org/wiki/Abstract_interpretation

[2] The figure is modified from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

For example, if we have

$$\overline{a} = [-1, 1], \underline{a} = [-2, 0], \overline{b} = [3, 4], \underline{b} = [2, 1]$$

, then we can compute the upper bound and the lower bound as:

$$\overline{a \odot b} = [-2, 4], \underline{a \odot b} = [-6, 0]$$

To define abstract interpretation for z_t, we use two abstract matrix multiplications to abstract $W^{(z)}x_t$ and $U^{(z)}h_{t-1}$, then we apply an abstract element-wise addition to these two items and followed by an abstract sigmoid function. The definition of r_t is similar to z_t. Also, for h'_t, we use two abstract matrix multiplications to abstract Wx_t and Uh_{t-1}. Then we use the abstract element-wise multiplication to abstract $r_t \odot Uh_{t-1}$. Finally, we apply an abstract element-wise addition to $Wx_t, r_t \odot Uh_{t-1}$ followed by an abstract $\tanh$ function.

For h_t, a naive approach is that we use two abstract element-wise additions to abstract $z_t \odot h_{t-1}$ and $(1 - z_t) \odot h'_t$, then we apply an abstract element-wise addition to these two items. However, this approach can be improved by directly defining a combined abstract interpretation for h_t:

$$\begin{aligned} \overline{h_t} = \max(&\overline{z_t} \odot \overline{h_{t-1}} + (1 - \overline{z_t}) \odot \overline{h'_t}, \\ &\overline{z_t} \odot \overline{h_{t-1}} + (1 - \overline{z_t}) \odot \underline{h'_t}, \\ &\overline{z_t} \odot \underline{h_{t-1}} + (1 - \overline{z_t}) \odot \overline{h'_t}, \\ &\overline{z_t} \odot \underline{h_{t-1}} + (1 - \overline{z_t}) \odot \underline{h'_t}, \\ &\underline{z_t} \odot \overline{h_{t-1}} + (1 - \underline{z_t}) \odot \overline{h'_t}, \\ &\underline{z_t} \odot \overline{h_{t-1}} + (1 - \underline{z_t}) \odot \underline{h'_t}, \\ &\underline{z_t} \odot \underline{h_{t-1}} + (1 - \underline{z_t}) \odot \overline{h'_t}, \\ &\underline{z_t} \odot \underline{h_{t-1}} + (1 - \underline{z_t}) \odot \underline{h'_t}) \end{aligned}$$

And the lower bound of h_t can be computed by replacing $\max$ with $\min$ in the above formula. The combination can improve the precision because it take the relation between z_t and $(1 - z_t)$. Besides, we use an example to illustrate the improvement of precision using the combined abstract interpretation for $c = z \odot a + (1 - z) \odot b$:

$$\overline{a} = [2], \underline{a} = [-2], \overline{b} = [\tanh(2)], \underline{b} = [\tanh(-2)]$$

$$\overline{z} = [\sigma(2)], \underline{z} = [\sigma(-2)]$$

, where all the vectors have only one element. If we use the naive approach to compute the upper bound and the lower bound, we will get

$$\begin{aligned} \overline{z \odot a} &= [1.76], \underline{z \odot a} = [-1.76] \\ \overline{(1 - z) \odot b} &= [0.85], \underline{(1 - z) \odot b} = [-0.85] \\ \overline{c} &= [2.61], \underline{c} = [-2.61] \end{aligned}$$

Instead, if we use the combined abstract interpretation for c, we will get

$$\bar{c} = [1.87],\ \underline{c} = [-1.87]$$

, which is more precise than the result of the naive approach.

4. Approaches

In this section, we introduced two approaches to improve the precision of the verification for RNNs.

4.1. Dynamic Programming

The dynamic programming is used to capture the global constraint δ. Moreover, it can be seen as the abstract interpretation for the unfolded L GRU cells. In other words, this abstract takes input as the input string and the initial hidden state, outputs every hidden state at each time t. The transition rule of dynamic programming is shwon as:

$$\begin{aligned} F[i][j] =& \text{GRU}(F[i-1][j], x_i) \\ &\sqcup \text{GRU}(F[i-1][j-1], \text{Abstract}(x_i)) \end{aligned}$$

, where $\sqcup$ is the join of two abstract values, and the Abstract function compute the abstract value for all the character/-word that x_i can substitute to.

4.2. Unions of Abstract Values

We need to carefully design the $\sqcup$ join operation between two abstract values. And two naive methods will fail:

- To merge two abstract values into a new abstract value. This method is light-weighted but imprecise.
- To keep track of unions of all abstract values. This method is slow since it leads to exponential size of unions, while it is precise.

To better trade-off the efficiency and the precision of them, we combined these two methods: we keep track of unions of abstract values within a budget b. During propagation, we keep track of unions of abstract values, and we will merge some of them when the size of the union exceeds budget b. Moreover, we need to (1) design abstract interpretations under the abstract domain that contains unions of abstract values; (2) design merging strategy for abstract values when the size exceeds b.

4.2.1. Abstract Interpretation for Unions of Abstract Values

We first divide operations to four groups and then we show how to design the abstract interpretation for unions of abstract values in each operation group.

Operations are divided into four groups: (1) the operation that takes only one variable as its input, such as matrix multiplication (with a constant matrix) and activation functions; (2) the operation that takes more than one variable as its inputs, such as element-wise addition, multiplication, and the combined h_t operations defined in Section 3; (3) the join operation $\sqcup$; (4) the verification function at the output layer.

Suppose f is an operation that takes only one variable $\mathbf{x}$ which is a union of abstract values $\{x_1, x_2, \ldots, x_b\}$, as its input, the abstract interpretation of f can be defined using the abstract interpretation of f under single abstract value:

$$f(\mathbf{x}) = \{f(x_1), f(x_2), \ldots, f(x_b)\}$$

The output of f will still be a union containing b abstract values.

Suppose g is an operation that takes k variables $\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^k}$, each of which is a union of abstract values $\{x^i_1, x^i_2, \ldots, x^i_b\}$, as its inputs, the abstract interpretation of g can be defined by the Cartesian product of the abstract interpretation of g under single abstract values:

$$\begin{aligned} g(\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^k}) = \{&g(x^1_1, x^2_1, \ldots, x^k_1), \\ &g(x^1_1, x^2_1, \ldots, x^k_2), \ldots, g(x^1_b, x^2_b, \ldots, x^k_b)\} \end{aligned}$$

Because we use the Cartesian product, the output of g will be a union containing b^k abstract values.

The join operation $\sqcup$ takes two unions of abstract values $\mathbf{x^1}, \mathbf{x^2}$, each of which is a union of abstract values $\{x^i_1, x^i_2, \ldots, x^i_b\}$, as its inputs, the abstract interpretation of $\sqcup$ can be defined by merging two unions:

$$\mathbf{x^1} \sqcup \mathbf{x^2} = \{x^1_1, x^1_2, \ldots, x^2_b, x^2_1, x^2_2, \ldots, x^1_b\}$$

At the output layer of the neural network, we will get a union of abstract values $\mathbf{x}$ containing $\{x_1, x_2, \ldots, x_b\}$. For a single abstract value x_i, we define the verification function for robustness as $V(x_i)$ mapping from a abstract value to $\{True, False\}$, where $True$ means robust and $False$ means not robust. Then the verification function V for robustness under a union of abstract values can be defined as:

$$V(\mathbf{x}) = \bigwedge_{i=1}^{b} V(x_i)$$

4.2.2. Merging Algorithm

When the size of the union t exceeds the budget b, we need to merge some abstract values to meet the budget. There are C^{b-1}_{t-1} ways to merge t abstract values into b abstract values. We want to find the one that has the least volume after merging. For example, the volume of a union of box domains is the sum of the volume of each box domain (intersections only sum once).

However, we are not afford to compute the C_{t-1}^{b-1} ways of merging. In this paper, we propose two algorithms to merge them efficiently.

We can merge them greedily:

1. Pick out two abstract values under a certain heuristic
2. Merge them and put the one after merging into the union
3. Repeat 1 and 2 until only b abstract values remain

We can select as the heuristic the two abstract values that have the least volume increment after merging, and the first step will cost $O(t^2 \log(t))$ using heap data structure, which means that the algorithm costs $O((t-b+1)t^2 \log(t))$ in all.

Also, we can use a constrained k-means algorithm (COP k-means) for the box domain:

1. For each abstract box, we create two vertexes of the box in the space and create a must-link constraint for them.
2. Apply COP k-means with $k = b$ and max iteration $= d$.

Because the COP k-means algorithm takes $O(tbd)$ time, this algorithm also takes $O(tbd)$ time in all.

5. Evaluation

5.1. Research Questions

In this section, we use experiments to answer four research questions:

- What is the performance of our verification tool?
- What is the impact of the combined abstract interpretation for h_t?
- What is the impact of the two approaches **Dynamic Programming** and **Unions of Abstract Values**?
- What is the impact of different budgets and merging strategies?

5.2. Experiment Setup

5.2.1. Datasets

We generate one artificial dataset in the experiments: **REGEX**$(.^*a.^*a.^*a.^*)$ dataset is an artificial dataset that consists of 100,000 training and 1,000 testing examples. This dataset contains positive examples that are accepted by the regular expression $.^*a.^*a.^*a.^*$ (with more than two 'a's in the string) and negative examples that are rejected by the regular expression (with less than three 'a's in the string).

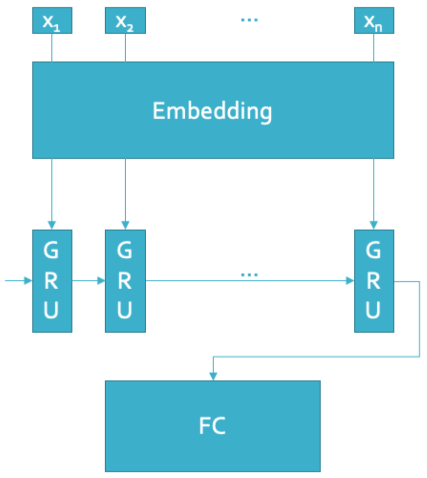


Figure 2. The RNN Architecture used

5.2.2. RNN Architecture

The RNN architecture (shown in Figure 2) contains a single RNN layer with GRU cells and passes the last hidden state to a fully-connected layer. For the **REGEX**$(.^*a.^*a.^*a.^*)$ dataset, we set the character-level embedding size to 2 and the dimension of GRU hidden states to 3.

5.2.3. Abstract Domain

We used the box domain in our implementation. And we used the unions of boxes for the unions of abstract values.

5.2.4. Robustness Properties

For the **REGEX**$(.^*a.^*a.^*a.^*)$ dataset, we define two robustness properties for positive examples and negative examples respectively. For the positive examples containing more than four 'a's, the classification results should not be changed when substituting 'a' with other characters with $\delta = 2$, denoted as REGEX (pos). For the negative examples containing no 'a's, the classification results should not be changed when substituting other characters with 'a' with $\delta = 2$, denoted as REGEX (neg).

5.2.5. Metrics for Evaluation

To show the effectiveness of the verification tool. We focus on three metrics:

1. Precision: the precision of the verification tool is de-

Property	Precision	Avg. Vol	Avg. Time
REGEX (pos)	100%	18.09216	9.3s
REGEX (neg)	100%	22.51459	12.3s

Table 2. The evaluation results of our verification tool with COP k-means $b = 5$ and the max iteration is 30.

Property	Precision	Avg. Vol	Avg. Time
REGEX (pos)	0%	$0.9 * 10^8$	1.2s
REGEX (neg)	0%	$1.1 * 10^8$	1.8s

Table 3. Without the combined abstract interpretation for h_t, the evaluation results of our verification tool with COP k-means $b = 5$ and the max iteration is 30.

fined as $\frac{\#TP}{\#TP+\#FP}$, where $\#TP$ and $\#FP$ are the number of true positives and false positives. The false positives are those robust examples on which our verification tool reports not robust.

2. Average Volume: the average volume of the verification results is defined as the average volume of the abstract value obtained at the output layer of the neural network. Less average volume means the verification tool is more precise.

3. Average Execution Time: the average time takes to verify the given input examples.

5.3. Evaluation Results

In the experiments of REGEX($.^*a.^*a.^*a.^*$) dataset, we random generate 100 positive and 100 negative examples for evaluation. All the examples above are correctly classified by the trained models.

5.3.1. RQ1: What is the performance of our verification tool?

We show the results of our verification tool in Table 2. As results, our verification tool can verify the trained RNN model having the two robustness properties with 100% precision.

5.3.2. RQ2: What is the impact of the combined abstract interpretation for h_t?

To study the impact of the combined abstract interpretation for h_t, we keep other settings the same as before but used a naive approach to compute h_t. The results are shown in Table 3, which, when comparing with Table 2, shows that the verification tool does not work at all without the combined abstract interpretation.

5.3.3. RQ3: What is the impact of the two approaches?

To answer this research question, we evaluated the performance of verification under three different settings: (1) without dynamic programming or unions of abstract values, (2) using unions of abstract values but without dynamic programming, (3) using dynamic programming but only kept one box during propagation, i.e., immediately merging. The results are shown in Table 4.

It can be seen from the results that the dynamic programming approach has the greatest improvement for precision and average volume. And it increases the precision from 0% to 100%. The unions of abstract values approach further improves the average volume, which means it is more precise, but with more average execution time.

5.3.4. RQ4: What is the impact of different budgets and merging algorithms?

To answer this question we evaluated the verification tool with two different merging algorithms: COP k-means and greedy. We show the results of COP k-means under different budgets $b = 1, 2, 3, 4, 5$, while we only show the results of greedy under budgets $b = 1, 2, 3, 4$ since this algorithm takes much more time. The results are shown in Figure 3, notice that we omit the figures for metric precision since they are all 100.0%.

It can be seen from the results that the average volume of greedy algorithm is smaller than COP k-means algorithm. But the average execution time of greedy is longer than COP k-means. Moreover, the average volume of both two algorithms converges when $b > 2$.

6. Related Work

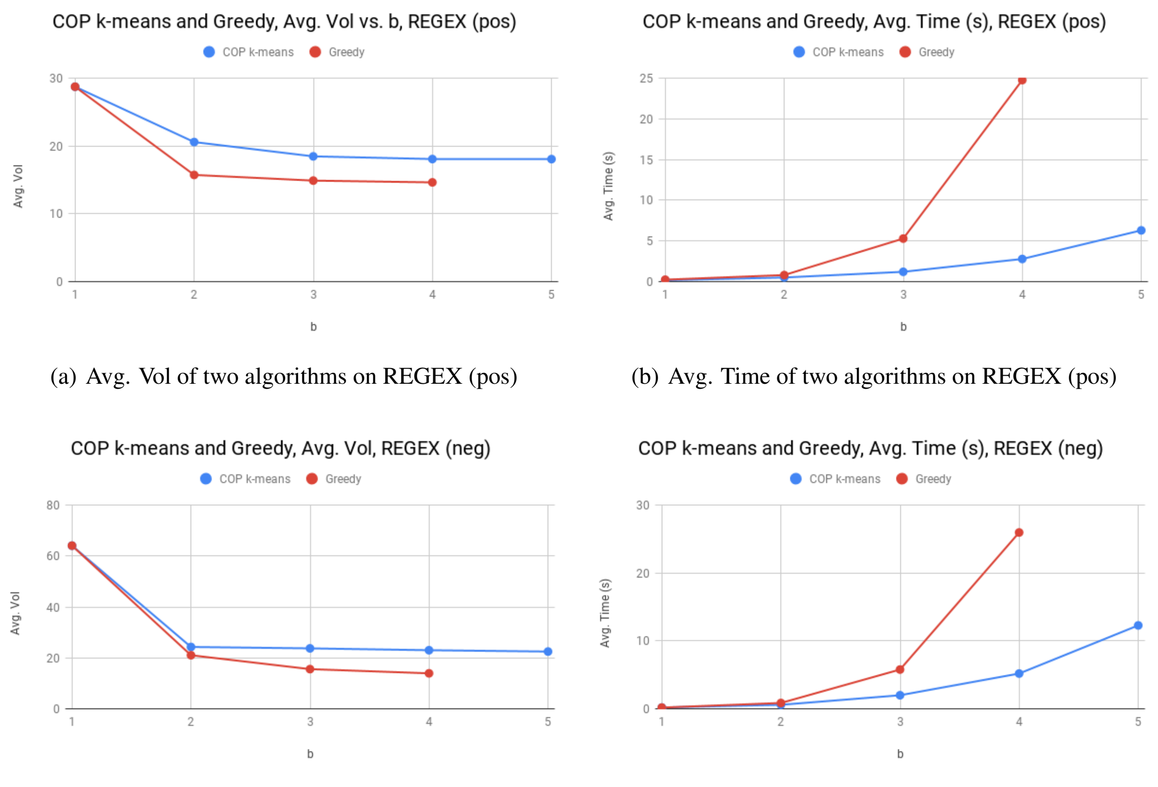
Verification of image classifiers: ERAN[3] is a state-of-the-art sound, precise, and scalable analyzer based on abstract interpretation for the complete and incomplete robustness verification of CNNs for image classification tasks. It was proposed and gradually improved by a series of papers: DeepZ (Singh et al., 2018) contains specialized abstract Zonotope transformers for handling ReLU, Sigmoid and Tanh activation functions. DeepPoly (Singh et al., 2019b) is based on a domain that combines floating point Polyhedra with Intervals. RefineZono (Singh et al., 2019a) combines DeepZ analysis with MILP and LP solvers for more precision. The tool ERAN is designed for the verification of image classification but not for natural language classification.

Verification of natural language classifiers: Huang et al. proposed the verification and robust training of CNN-based

[3]https://github.com/eth-sri/eran

Property	REGEX (pos)			REGEX (neg)		
Settings	Precision	Avg. Vol	Avg. Time	Precision	Avg. Vol	Avg. Time
Boxes	0%	2571.54559	0.07s	0%	2597.64948	0.07s
Unions (b=5)	0%	2571.31219	2.53s	0%	2597.58475	3.48s
DP+Boxes	100%	28.78412	0.17s	100%	64.16382	0.19s
DP+Unions (b=5)	100%	18.09216	6.31s	100%	22.51459	12.3s

Table 4. The comparison among the tool without any of two approaches (Boxes), with only unions of abstract values (Unions), with only dynamic programming (DP+Boxes), and with two approaches (DP+Union).



(a) Avg. Vol of two algorithms on REGEX (pos)

(b) Avg. Time of two algorithms on REGEX (pos)

(c) Avg. Vol of two algorithms on REGEX (neg)

(d) Avg. Time of two algorithms on REGEX (neg)

Figure 3. Results of Research Question 4.

natural language classifiers (Huang et al., 2019). A special convex-hull input domain is designed to precisely capture the discrete input space which allows substitutions. However, the convex-hull input domain only works when the first layer is an affine transformation layer and it does not work for RNN layers.

Verification of audio classifiers: Recently, an anonymous paper (Anonymous, 2020) has been proposed to verify the RNN-based audio classifiers using LSTM cells. The paper presented novel abstract interpretations for LSTM cells and some non-linear functions used in audio processing. The anonymous paper works on audio classification where the input space is continuous and can be precisely captured by box and zonotope domains, while our paper works on natural language classification where the input space is discrete and cannot be precisely captured by box and zonotope domains. Besides, we defined a novel abstract interpretation of GRU cells under unions of abstract values, while they defined a novel abstract interpretation of LSTM cells under Polyhedra.

7. Limitation and Future Work

We only designed and implemented the abstract interpretation for unions of box values. Because it is a challenge to design the abstract interpretation for unions of zonotopes and Polyhedrons, which also shows a promising research direction. For polyhedrons, it takes exponential time to construct a polyhedron precisely, not even say how to merge two polyhedrons. However, comparing the results of the the anonymous paper (Anonymous, 2020), polyhedron domain will be more precise than the box domain in terms of the abstract interpretation for RNN cells, such as GRU and LSTM. For zonotopes, even though we can construct them efficiently, it is still a challenge to design the merging algorithm for the zonotope domain. The greedy algorithm fails since it need to compute the volume of a unions of zonotopes, which takes exponential time to do that. However, COP k-means algorithm still works only if we can find representative vertexes for a zonotope.

We only do evaluate on the artificial dataset not on the real-world dataset. And it will be one of our future work.

8. Conclusion

In this paper, we studied the verification of recurrent neural networks. We proposed a novel abstract interpretation for the GRU cells used in RNNs. Moreover, we used dynamic programming and kept unions of abstract values to improve the precision of the verification. With the help of the abstract interpretation and two approaches, we were able to verify the recurrent neural network under an artificial dataset.

References

Anonymous. Certifying neural network audio classifiers. In *Submitted to International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxkvlBtwH. under review.

Carlini, N. and Wagner, D. A. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pp. 39–57, 2017. doi: 10.1109/SP.2017.49. URL https://doi.org/10.1109/SP.2017.49.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Ebrahimi, J., Rao, A., Lowd, D., and Dou, D. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pp. 31–36, 2018. doi: 10.18653/v1/P18-2006. URL https://www.aclweb.org/anthology/P18-2006/.

Huang, P., Stanforth, R., Welbl, J., Dyer, C., Yogatama, D., Gowal, S., Dvijotham, K., and Kohli, P. Achieving verified robustness to symbol substitutions via interval bound propagation. *CoRR*, abs/1909.01492, 2019. URL http://arxiv.org/abs/1909.01492.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. T. Fast and effective robustness certification. 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. 10825–10836, 2018. URL http://papers.nips.cc/paper/8278-fast-and-effective-robustness-certification.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. T. Boosting robustness certification of neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019a. URL https://openreview.net/forum?id=HJgeEh09KQ.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. T. An abstract domain for certifying neural networks. *PACMPL*, 3 (POPL):41:1–41:30, 2019b. doi: 10.1145/3290354. URL https://doi.org/10.1145/3290354.

Zhang, H., Zhou, H., Miao, N., and Li, L. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5564–5569, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1559. URL https://www.aclweb.org/anthology/P19-1559.