Summary of One paper about Verification on machine learning

Presented by : Ji Gao

[1]Department of Computer Science, University of Virginia
https://qdata.github.io/deep2Read/

August 26, 2018

Overview

1. Reluplex: An efficient SMT solver for verifying deep neural networks

Outline

1. Reluplex: An efficient SMT solver for verifying deep neural networks

Abstract

Abstract: Deep neural networks have emerged as a widely used and effective means for tackling complex, real-world problems. However, a major obstacle in applying them to safety-critical systems is the great difficulty in providing formal guarantees about their behavior. We present a novel, scalable, and efficient technique for verifying properties of deep neural networks (or providing counter-examples). The technique is based on the simplex method, extended to handle the non-convex Rectified Linear Unit (ReLU) activation function, which is a crucial ingredient in many modern neural networks. The verification procedure tackles neural networks as a whole, without making any simplifying assumptions. We evaluated our technique on a prototype deep neural network implementation of the next-generation airborne collision avoidance system for unmanned aircraft (ACAS Xu). Results show that our technique can successfully prove properties of networks that are an order of magnitude larger than the largest networks verified using existing methods.

Verification of Machine Learning Models

- Neural Networks are popular. Applications of Neural Networks are widespread.
- However, it has been observed that DNNs can react in unexpected and incorrect way. (Adversarial samples)
- Automatic verification tools are needed.
- However, it's hard.

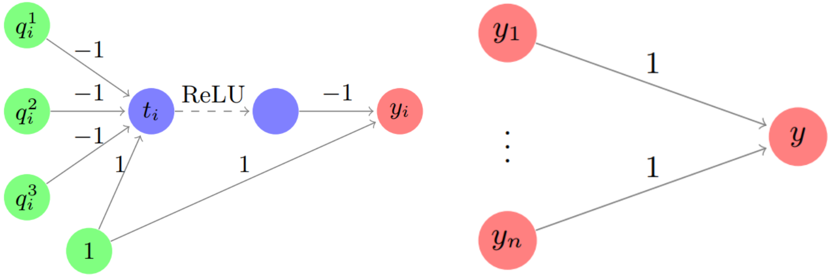
Hardness of verify DNN model

- DNNs are large, non-linear and non-convex systems.
- Verify even simple properties of DNN is NP-complete.
- Current tools can only work on very small models (A single hidden layer, with 10 to 20 hidden nodes).

NP-Complete

- Suppose $\phi = \phi_1(x) \wedge \phi_2(y)$, where x is the input and y is the output of DNN (which only include relu activation). Then determining whether ϕ is satisfiable is NP-complete.
- Proof:
  1. NP. As a pair of answer can be easily checked in polynomial time.
  2. NP-Hard. 3-SAT formula can be convert to the DNN form.

  3-SAT: $\psi = C_1 \wedge C_2 \wedge .. \wedge C_n$, each C_i is disjuction of three literals $q_1 \vee q_2 \vee q_3$.



Satisfiability Modulo Theories

- Satisfiability Modulo Theories(SMT): generally, a formula in first-order logic.
- Example:

$$(\sin(x)^3 = \cos(\log(y) \cdot x) \vee b \vee -x^2 \geq 2.3y) \wedge \left(\neg b \vee y < -34.4 \vee \exp(x) > \frac{y}{x}\right)$$

- Could be viewed as an extension to SAT.

Solving Satisfiability Modulo Theories

- Need additional restrictions applied on SMT to have efficient solver
- In DNN case, we focus on a special linear form, in which $\{+, -, \cdot, \leq, \geq\}$ are supported operations, and $\cdot$ only work on one variable and one constant.
- Can be rewritten into the form $\sum_{x_i} c_i x_i \bowtie d$, in which $\bowtie \in \{=, \leq, \geq\}$. Called "Linear atoms"
- Linear Atoms can be solved with an algorithm called the simplex method.

Simplex Algorithm: Setting

- A state of the simplex algorithm is the tuple $\langle B, T, l, u, a\rangle$ or $\{\mathrm{SAT}, \mathrm{UNSAT}\}$
- B: A set of basic variables
- T: tableau, contains how a basic variable are the combination of nonbasic varabiles. Equation: $x_i = \sum_{x_j \notin B} c_j x_j$
- l,u: Current lower and upper bound for variable x_i
- a: assignment of the variables, maps the variable to a real value

Simplex Algorithm

- Each time follows a rule
- Simplex algorithm has been proven to be sound and complete.

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \text{slack}^+(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \text{slack}^-(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Update} \quad \frac{x_j \notin \mathcal{B}, \quad \alpha(x_j) < l(x_j) \vee \alpha(x_j) > u(x_j), \quad l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)}{\alpha := update(\alpha, x_j, \delta)}$$

$$\text{Failure} \quad \frac{x_i \in \mathcal{B}, \quad (\alpha(x_i) < l(x_i) \ \wedge \ \text{slack}^+(x_i) = \emptyset) \vee (\alpha(x_i) > u(x_i) \ \wedge \ \text{slack}^-(x_i) = \emptyset)}{\texttt{UNSAT}}$$

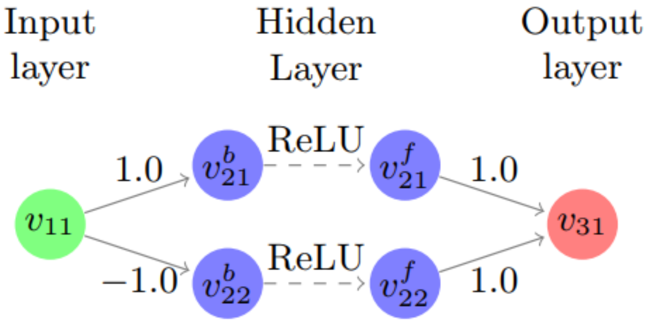
$$\text{Success} \quad \frac{\forall x_i \in \mathcal{X}.\ l(x_i) \leq \alpha(x_i) \leq u(x_i)}{\texttt{SAT}}$$

Fig. 3: Derivation rules for the abstract simplex algorithm.

$$\text{slack}^+(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} > 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} < 0 \wedge \alpha(x_j) > l(x_j))$$
$$\text{slack}^-(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} < 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} > 0 \wedge \alpha(x_j) > l(x_j))$$

Reluplex

- Linear atoms work for most cases in DNN, but not for the activation functions.
- Idea: Encode a single ReLU node v as a pair of variables, v_b and v_f, and then assert ReLU(v_b,v_f).



Reluplex: Update rules

- Need to add additional rules
- Update b and Update f coordinate the variables "before" and "after" the Relu operation.

$$\text{Update}_b \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := update(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\text{Update}_f \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := update(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

$$\text{PivotForRelu} \quad \frac{x_i \in \mathcal{B}, \quad \exists x_l.\ \langle x_i, x_l \rangle \in R \vee \langle x_l, x_i \rangle \in R, \quad x_j \notin \mathcal{B}, \quad T_{i,j} \neq 0}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, \quad l(x_i) < 0, \quad u(x_i) > 0}{u(x_i) := 0 \qquad l(x_i) := 0}$$

$$\text{ReluSuccess} \quad \frac{\forall x \in \mathcal{X}.\ l(x) \leq \alpha(x) \leq u(x), \quad \forall \langle x^b, x^f \rangle \in R.\ \alpha(x^f) = \max(0, \alpha(x^b))}{\texttt{SAT}}$$

Fig. 5: Additional derivation rules for the abstract Reluplex algorithm.

Experiment

Compare to other SMT and LP solvers. Clearly, previous models don't accept all kinds of inputs.

Model in the test: 6 layer with 300 relus.

Table 1: Comparison to SMT and LP solvers. Entries indicate solution time (in seconds).

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8
CVC4	-	-	-	-	-	-	-	-
Z3	-	-	-	-	-	-	-	-
Yices	1	37	-	-	-	-	-	-
MathSat	2040	9780	-	-	-	-	-	-
Gurobi	1	1	1	-	-	-	-	-
Reluplex	8	2	7	7	93	4	7	9

Experiment

Result of properties.

Table 2: Verifying properties of the ACAS Xu networks.

	Networks	Result	Time	Stack	Splits
ϕ_1	41	UNSAT	394517	47	1522384
	4	TIMEOUT			
ϕ_2	1	UNSAT	463	55	88388
	35	SAT	82419	44	284515
ϕ_3	42	UNSAT	28156	22	52080
ϕ_4	42	UNSAT	12475	21	23940
ϕ_5	1	UNSAT	19355	46	58914
ϕ_6	1	UNSAT	180288	50	548496
ϕ_7	1	TIMEOUT			
ϕ_8	1	SAT	40102	69	116697
ϕ_9	1	UNSAT	99634	48	227002
ϕ_{10}	1	UNSAT	19944	49	88520

Experiment - Adversarial Robustness

- Check for a certain point x, whether for every x' that $|x' - x|_\infty < \delta$, the output are the same.

Table 3: Local adversarial robustness tests. All times are in seconds.

	$\delta = 0.1$		$\delta = 0.075$		$\delta = 0.05$		$\delta = 0.025$		$\delta = 0.01$		Total
	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time	Time
Point 1	SAT	135	SAT	239	SAT	24	UNSAT	609	UNSAT	57	1064
Point 2	UNSAT	5880	UNSAT	1167	UNSAT	285	UNSAT	57	UNSAT	5	7394
Point 3	UNSAT	863	UNSAT	436	UNSAT	99	UNSAT	53	UNSAT	1	1452
Point 4	SAT	2	SAT	977	SAT	1168	UNSAT	656	UNSAT	7	2810
Point 5	UNSAT	14560	UNSAT	4344	UNSAT	1331	UNSAT	221	UNSAT	6	20462