A CEGAR Based Verification of Neural Network

Mohammad Afzal[1,2], Ashutosh Gupta[1], and Akshay S[1]

[1] Indian Institute of Technology, Bombay, India
[2] TCS Research, Pune, India

Abstract In this paper, we present a counter example guided refinement(CEGAR) based verification of neural networks.

1 Introduction

2 A Motivating Example

3 Preliminaries

4 Algorithm

A neural network is a collection of layers $l_0, l_1, l_2, ...l_k$, where k represents the number of layers. Layer l_0 and l_k represent the input and output layers respectively and all the other layers are hidden layers. Each layer l_i is a collection of nodes. A node $v_{i,j}$ represents the j^{th} node in the layer i. Let us say a vector $\overline{V_i} = [v_{i,0}, v_{i,1}, ...v_{i,m}]$ represents the values of each node in the layer i, where m is the number of nodes in the same layer. Each layer's values are computed using the weighted sum of the previous layer's values($W_i * V_{i-1} + B_i$) followed by an activation function $ReLU$. A function $y = max(0, x)$ is a $ReLU$ function which takes an arguments x as input and return the same value x as output if x is non-negative otherwise return the value 0.

A neural network is a function N which takes an input of m dimensions and gives an output of n dimensions. N can be represented as a composition of functions $f_l o f_{l-1} ... o f_1$, where each function f_i represents the linear combinations followed by an activation function.

Let us say P and Q are the predicates on the input and output space respectively. The goal is to find an input $\overline{x_0}$, such that the predicates $P(\overline{x_0})$ and $Q(N(\overline{x_0}))$ holds. The predicate Q usually is the negation of the desired property. The triple $\langle N, P, Q \rangle$ is our verification query.

The algorithm 1 represents the high level flow of our approach. Which is a counter example guided abstract refinement(CEGAR) based approach. At the first line in algorithm 1, we generate all the abstract constraints by *deeppoly*. These abstract constrains are the lower and upper constraints as well as the lower and upper bounds for each neurons in the neural network. The *isVerified* function in algorithm 1 calls either algorithm 2 or algorithm 3. Both the algorithms 2 and 3 takes *markedNeurons* as input. The *markedNeurons* is a subset of the neurons of the neural network. The *markedNeurons* represents the set of the culprit neurons. Algorithm 2 replace the abstract constraints of *markedNeurons*

Algorithm 1 A CEGAR based approach of neural network verification

Input: A verification problem $\langle N, P, Q\rangle$
Output: UNSAT or SAT

Build abstract constrains and bounds on each neuron using deeppoly.
A is a set of all abstract constraints and bounds.
markedNeurons = {}
while True **do**
    isVerified or spuriousCEX = isVerified($\langle N, P, Q\rangle$, A, markedNeurons)
    if isVerified is True **then**
        return UNSAT
    else
        if spuriousCEX is a real cex of $\langle N, P, Q\rangle$ **then**
            return spuriousCEX
        else
            marked, cex = getMarkedNeurons($\langle N, P, Q\rangle$, A, markedNeurons)
            if cex not None **then**
                return cex
            markedNeurons = markedNeurons $\cup$ marked

Algorithm 2 An approach to verify $\langle N, P, Q\rangle$ with abstraction A

Name: isVerified1
Input: $\langle N, P, Q\rangle$ with abstract constraints A and markedNeurons $\subseteq$ $N.neurons$
Output: verified or spurious counter example.

for nt in markedNeurons **do**
    $A = (A \cup exactConstr(nt)) \setminus abstractConstr(nt)$
$A = A \cup P \cup \neg Q$
isSat = checkSat(A)
if isSat is True **then**
    return spurious counter example
else
    return verified

by exact constraints and check for the property by MILP solver. If MILP solver return SAT then we return satisfying assignments as a spurious counter example, otherwise return verified. Algorithm 3 split each neuron of *markedNeurons* into two sub cases. Suppose the neuron $x \in [lb, ub]$ belongs to *markedNeurons*, then first case is when $x \in [lb, 0]$ and the second case is when $x \in [0, ub]$. After splitting neurons into two cases *deeppoly* run for both cases separately. In algorithm 3, *deeppoly* run exponential number of times in the the size of the *markedNeurons*. We return verified in algorithm 3 if verification query verified in all the *deeppoly* runs. If *deeppoly* fails to verify in any case then we return the spurious counter example.

5 Experiments

6 Conclusion and Future work

References

Algorithm 3 An approach to verify $\langle N, P, Q \rangle$ with abstraction A

Name: isVerified2
Input: $\langle N, P, Q \rangle$ with abstract constraints A and markedNeurons
Output: verified or spurious counter example.

for all combination in $2^{markedNeurons}$ **do**
run deeppoly
if not verified by deeppoly **then**
A = Set of all abstract constraints generated by deeppoly
$A = A \cup P \cup \neg Q$
isSat = checkSat(A)
if isSat **then**
return spurious counter example
return verified

Algorithm 4 A pullback approach to get mark neurons or counter example

Name: getMarkedNeurons1
Input: $\langle N, P, Q \rangle$ with abstract constraints A and satisfying assignments $\overrightarrow{x}$ and $\overrightarrow{y}$ on input and output layers respectively
Output: marked neurons or real counter example.

Run neural network on $\overrightarrow{x}$.
return $\overrightarrow{x}$ as counter example if it's output violate the Q.
for currLayer in [outputLayer ... inputLayer] **do**
if currLayer is affine layer **then**
layerConstraints = []
for nt in currLayer.neurons **do**
nt.constr = (nt.affineExpr == nt.satval)
layerConstrains.append(nt.constr)
for nt in currLayer.prevLayer.neurons **do**
layerConstrains.append(nt.bounds)
isSat = checkSat(layerConstrains)
if isSat **then**
assign sat values to previous layers neurons
else
markedNeurons = {nt | nt $\in$ currLayer.neurons $\wedge$ nt.constr $\in$ unsatCore}
return markedNeurons
else ▷ Relu layer
for nt in currLayer.neurons **do**
if nt.satval > 0 **then**
prevNt.satval = nt.satval ▷ prevNt is input node of nt in prevLayer
else
prevNt.lb $\leq$ prevNt.satval ≤ 0 ▷ lb,ub are bounds
ce = {nt.satval | nt $\in$ inputLayer.neurons} ▷ If pullbacked upto input layer
return ce

Algorithm 5 An optimization based approach to get mark neurons or counter example

Name: getMarkedNeurons2

Input: $\langle N, P, Q \rangle$ with abstract constraints A and satisfying assignments $\overrightarrow{x}$ and $\overrightarrow{y}$ on input and output layers respectively

Output: marked neurons or real counter example.

```
Run neural network on x⃗.
return x⃗ as counter example if it's output violate the Q.
Let us say layer.nt.val is the value evaluated on x⃗ for each layer and each neuron nt.
for currLayer in [firstLayer ... outputLayer] do                ▷ inputLayer excluded
    if currLayer is affine layer then
        execute currLayer.prevLayer.vals on currLayer ▷ simple matrix muliplication
    else
        constraints = A[currLayer ... outputLayer]      ▷ Abstract constraints from currLayer to outputLayers
        constraints.add(currLayer.prevLayer.vars == currLayer.prevLayer.vals)
        constrains.add(outputLayer.vars == outputLayer.vals)
        softConstraints = for each nt ∈ currLayer.neurons (nt.vars == nt.vals)
        maximize the number of neurons of layer which satisfy the softConstraints.
        if all neurons of the currLayer satisfy the softConstraints then
            continue
        else
            markedNeurons = all neurons of currLayer which does not satisfy the softConstraints.
            return markedNeurons
```