Verifying Deep Neural Networks

CSPC 513 : Project Report

Shivam Thukral
Department of Computer Science
The University of British Columbia
tshivam2@cs.ubc.ca

I. Introduction

Deep Neural Networks are being used in a variety of applications ranging from image segmentation, image classification to game play, speech recognition and in various other fields. The software we build to tackle such complex problems is difficult to interpret and takes a lot of time to train and test. Catching bugs in such programs is a very time consuming and strenuous task. Initial attempts to verify such networks involved constructing safety cases. Such test cases involve characterizing a set of environmental conditions to produce a desired output followed by testing of learnt model to ensure that this required output is generated under all considerable environmental conditions. For example, in character recognition system, we can define that all images which are close to a given example image should be classified as the digit given in the sample example.

A. *Why is this problem difficult?*

Majority of the networks used in such complex problems have piece-wise linear activation functions within the nodes. Such activation functions are amenable to formal verification as we can verify their behaviour using SMT solvers with theory of linear arithmetic. In such an approach, the solver selects the phase of the node, and then applies a LP-solver to check if there exists concrete real valued inputs to the network such that all the nodes have selected phases. These phases represent the part of piece-wise linear activation function which is being used by each node. SMT instances stemming from such encodings are difficult to solve as they need to iterate through many phase combinations before an instance is found to be satisfiable or unsatisfiable.

II. Problem Statement

In this project, I plan to verify the properties of a deep neural network which is being used in a real world application. through a framework consisting of SMT solver. Developing a framework to test neural networks on safety cases would help with certification as well as provide valuable feedback to the system engineer.

A. *Real-world application*

ACAS Xu [1] is an airborne safety critical aircraft collision avoidance system implemented using deep neural network. Under certain circumstances DNN can behave unexpectedly which can be very dangerous for such safety critical systems. Hence, we should have some method to formally guarantee the behaviour of the models used.

III. Literature Survey

This survey summarizes a list of papers in chronological order starting from simplex method, invented by George Dantzig in 1947, to current state-of the art approaches used to formally verify deep networks. After summarising each paper, I have added a short section of my learning takeaways for each paper. This provides us a broader overview of the approach and helps in comparison amongst different approaches as well.

A. *Simplex Method*

Reluplex is one of the tools used to verify deep neural networks. This tool extends Simplex Method for formal verification. To understand Reluplex, we first need to understand how Simplex works.
Simplex[2] is a method for solving problems in linear programming. The central problem of linear programming consists of finding values of a set of non-negative variables that satisfy a system of linear equations and at the same time minimises an objective function. The first step of simplex method is addition of artificial variables into the standard form. The resulting auxiliary problem is in canonical form. At this point simplex algorithm is employed which consists of a sequence of pivot operations referred to as ‘Phase 1’. These operations produce a sequence of different canonical forms. The objective of this phase is to find a feasible solution if it exists. If the final canonical form gives a solution, leading to ‘Phase 2’, we perform a second succession of pivot operations at this stage again. The objective of this phase is to find an optimal feasible solution if it exists.

What I have learnt?

For Reluplex, we are more concerned about ‘Phase 1’, where we put our efforts in finding a solution rather than optimizing the found solution. In preference to explaining each and every step of the algorithm, I would rather prefer to show a simple example in which simplex is applied to solve a problem. Please

refer supplementary material for this example. This example uses a very simple deep neural network to solve a verification problem.

B. NeVeR : Abstraction and Refinement for verification for neural networks

This paper presents an approach[3] to verify neural networks with sigmoid activation functions. This approach can broadly be divided into two phases. In the first phase, sigmoid functions are substituted with boolean combinations of linear arithmetic constraints. For the neural network being considered, the abstraction (over-approximation) is done in such a way that it is consistent with the concrete network i.e. once the abstract neural network is proven to be safe, the same should hold for the concrete network. Such abstractions come at the price of spurious counterexamples ie. there may exist some abstract counter examples that don't correspond to the concrete network. A spurious counter example calls for refinement of the abstraction which is done in the second phase.
The second phase highlights automated neural network repair in which misbehaviour's are corrected. Intuitively, the abstract neural network over approximates the concrete one and spurious counterexamples are weak spots of the abstract neural network which can also be used as critical points for the concrete network. Introducing such spurious examples to the training set of the network could lead to a safer network as compared to the original one. This approach is referred to as CETAR which stands for Counter Example Triggered Abstraction Refinement.
This approach is tested using a tool NeVeR, a neural network verifier, which uses HySAT to verify the abstract network. Authors did a case study on an industrial manipulator and demonstrated how it can be used to improve the safety of the system design in a completely automated way.
What I have learnt?
The major limitations of this approach is in terms of scalability. A spurious counter example would call for a refinement of the abstract neural network which in turn can make the verification process more expensive. This situation is worsened in larger neural networks.
This work was one of the stepping stones towards formal verification of machine learning techniques. Two key ideas were presented in this paper:

- By consistent abstraction (over-approximation) of the neural network, we can verify the safety properties in reasonable amount of time.
- Over-approximation can lead to faulty behavior of the abstract neural network which can be repaired to a certain extent automatically with spurious counterexamples.

C. Challenging SMT solvers to verify neural networks

This paper[4] depicts a thorough picture of achievements of SMT solvers and also highlights the open problems which were needed to be addressed in 2012. The solvers used in this study for comparison are HySAT[5], MATHSAT[6], Yices[7] and for baseline, CVC[8] has been considered.
Authors have described three groups of experiments in this paper:

- In the first set of experiments, different types and sizes of neural networks at different degrees of abstractions have been considered. The numbers of hidden neurons in the range of 5,10,20 have been varied and either one or six size output neurons have been considered. As part of the results, all solvers except CVC were able to solve 70% of the test sets. Yices outperformed other solvers by solving 96% of the tests, while HySAT and MATHSAT were able to solve 83% and 81% respectively.
- Second set of experiments were done to analyze the scalability of SAT solvers in finding solutions to SAT and UNSAT problems. For the experiments, they varied fine-grained abstractions, different sized networks in terms of hidden neurons and considered satisfiable results from the solvers. Generally, fine grained abstractions are difficult to handle as it takes a lot of time to find counterexamples as compared to coarse grained abstractions, which are much easier to solve. For the encoding that was more challenging, no solver was able to solve all test cases within a certain amount of time. In the initial set of experiments, HySAT performed the best followed by Yices. In the next set of problems related to larger network architectures, no solver was able to solve for satisfiability.
- Last set of experiments were performed using the NeVeR tool equipped with various solvers as a backend. From the experiments, it was concluded that Yices was the best candidate because it scales better with encodings that are related to small scale abstraction values.

What I have learnt?
The goal of this paper was to compare state of the art SMT solvers on challenging test cases of large networks for verification and provide some insights. The study shows that although SMT solvers have the capability of attacking several non-trivial (sub) problems in the neural network verification arena. However, the overall verification process which involves realistic sized networks and fine grained abstraction remains a standing problem.

D. Reluplex : ReLU + Simplex

Neural networks involving activation function as ReLU can be directly encoded as logical formulas due to piece-wise linear nature of the activation function. We can use this to verify properties of such neural networks. This does not scale well with large networks where each node/neuron needs to be encoded as a disjunction of phases. To tackle this issue, authors extended the theory of linear real arithmetic with

ReLU activation function and proposed a solver.
Katz et al [9] proposed an algorithm called Reluplex which starts with a set of initial assignments and then attempts to fix violated constraints at each step. Reluplex handles two types of constraints; linear and ReLU activation function constraints. Firstly, the algorithm tries to fix linear constraints. If no solution to this problem (containing only linear constraints) is found, the counter example search is said to be UNSAT. If the solution is found, the algorithm then attempts to fix one of the violated ReLU constraints, potentially leading to more violated linear constraints. Since this process is not guaranteed to converge, they handle this by generating case splitting of ReLU functions for two cases(active and inactive phase). Now, they try to solve two problems each having one phase of ReLU.
Reluplex is shown to outperform existing state of the art SMT solvers for verifying the safety properties of neural networks. For experimental study, they used ACAS Xu aircraft collision avoidance system. The neural network has around 300 ReLU neurons and 8 layers. However, Reluplex takes several hours to terminate and becomes unscalable on larger networks,
What I have learnt?
Reluplex is obtained from Simplex by adding rules specific to ReLU constraints. Authors only highlighted their working with ReLU activation functions. Current implementation of Reluplex does not support other piece-wise linear functions like Max-Pooling units. Also, it can be concluded from the results that Reluplex does not work well with large neural networks.

E. Measuring Neural Network Robustness with Constraints

Recent work[10] has shown that it is often possible to construct examples for neural networks which can lead to mis-classification. Even small perturbations in a carefully chosen direction can cause a complete system to fail. Such failures are very dangerous in safety critical systems where we cannot afford to fail on adversarial inputs.
A typical approach in this domain is to first find adversarial examples for a neural network F using an algorithm A. Then, we need to augment the training dataset with these examples and retrain to get a new model F'. Robustness is evaluated by using the same algorithm A to find adversarial examples on F'. If the algorithm discovers less adversarial examples for F' than for F, then F' is said to be more robust. The limitation of this approach is that it can cause the network to overfit on adversarial examples generated by algorithm A. For this we can use a different algorithm A' and try to find adversarial examples for both F and F'.
Bastani et al. [10] also proposed two statistics to measure the robustness of a neural network. The first statistic measures the frequency with which adversarial examples occur and the second one measures the severity of such examples.
The authors propose a technique for finding local adversarial examples in DNN with ReLU's. Given an input point x, they encode the problem as a linear programming problem and solve this using a linear programming solver.
Results are shown on image recognition networks in which they demonstrate examples of adversarial inputs which can lead to mislabeling of digits in the MNIST dataset. Similar examples were shown for CIFAR-10 trained networks. Proposed algorithm is sound but incomplete since discovered adversarial inputs are correct but it can miss some adversarial inputs.
What I have learnt?
Although this study did not include verification in great detail, however, the core idea presented in this paper was useful for verification in which we reduced a verification problem to a linear programming problem. Another, important aspect of this paper was replacement of ReLU constraints with linear constraints. ReLU activation function, y = ReLU(x), can be divided into two phases :

- Active phase : $(x >= 0)$ and $(y = x)$
- Inactive phase : $(x < 0)$ and $(y = 0)$

F. Planet : Verification of Piece-Wise Linear Neural Networks

Planet[11] was the first attempt that can support almost all the linear-piecewise activation functions such as max pooling and ReLU nodes. Unike Reluplex which can only support ReLU activation functions this approach employs a novel approximation to overall network behaviour. The approach splits the problem over possible states of the activation function. At each step it assigns true/false values and verifies the feasibility of the partial assignments. This approximation allows to quickly rule out large search space parts of the phases where it knows that the solution is infeasible.
In the beginning, Planet deploys a linear programming solver and a SAT solver. SAT solver guides the approach in the searching process by determining the phases (true/false) for each node and maintains a set of constraints over node phase combinations. SAT solver performs unit propagation, clause learning, branching and back tracking as usual but the interesting case comes when the solver is about to branch, here they use LP solver to check linear approximation for the feasibility of the behaviour. If a conflict occurs it learns the conflict clause and infer implied node phases in the search process.
This approach is tested on two case studies, collision avoidance and digits recognition networks. The comparison is done in terms of computation time against SMT solvers like Yices and LP solvers like Gurobi. Results show Yices is slower and Gorubi is faster due to high optimisation and sophisticated

heuristics resulting from its industrial applications.

What I have learnt?

This approach is better than Reluplex but is still limited to network types where all components have piece-wise linear activation functions. It's still a challenge to solve for advanced activation functions like exponential linear units. Also, this approach cannot be scaled to a very deep neural network

G. AI^2 : Abstract Interpretation for Artificial Intelligence

Until now, none of the approaches have provided a sound analysis of neural networks mentioning the use of large networks. Such tasks become more challenging when we have to verify properties with precision. Gehr et al. [12] presents a system called AI^2 (Abstract Interpretation for Artificial Intelligence), which is a fast scalable analyser that can handle common network types with linear activation functions. The basic idea behind this approach is given a neural network and its inputs, abstract interpretation computes an abstract output which is an overapproximation of the outputs for the concrete network. This abstract output is a strict superset of the possible concrete output. In the concrete domain, we can use this abstract interpretation to verify the properties of the network.

This approach is sound but incomplete, i.e the abstract domain might not be able to prove that all the properties of the network are true, but for the ones it's able to prove, it holds for concrete networks as well. The authors describe examples of abstract domains like box domain, zonotopes, and polyhedra.

AI^2 is evaluated on two neural networks trained to recognize digits and objects. The tool was able to generate results for networks that have 53000 neurons. This approach scales better than Reluplex, as it was able to find out results for large networks within minutes as compared to Reluplex which can take upto several hours.

What I have learnt?

AI^2 was first of its kind in which they try to generate robustness certificates. The abstract interpretations help us to verify the robustness of a network given its inputs. Abstracting a norm-ball around an input sample using polyhedra, leads to over approximation of the norm ball. If a proposed method certifies the robustness for a given test sample and an area around it, then the concrete network is also robust in this area. If this is not true then the network is not robust or robustness certificates cannot be verified due to over-approximation.

H. PLNN : Piece-wise Linear Neural Network

The paper[13] is one of the most recent works which presents a unified summary of all the important methods used in verification of the neural networks. Within the paper, authors critically analyse each approach and propose a speedup technique by combining approaches from different algorithms. The second most important contribution of this work is augmentation of new test cases in the previously existing datasets. They used these benchmark test cases to evaluate existing algorithms in terms of strengths and weaknesses.

The neural networks used in this study represent the major of networks used in practice. These networks have fully-connected or convolutional layers and pooling units like MaxPool and activation functions like ReLU.

Verification as a satisfiability problem attempts to discover a counterexample that would make the property false. This is done by defining a set of equations representing input, hidden units and the outputs as equations and taking the opposite of the objective function followed by an effort to find an example which satisfies these constraints. If the problem is unsatisfied then no counterexample exists implying that the original property is true. Authors discussed a generic algorithm for verification called branch and bound by providing its pseudo-code and later demonstrated how Reluplex and planet are special cases of this branch and bound framework.

For evaluation, they compared a simple baseline algorithm called black box (uses Gurobi solver), with SMT solvers like Reluplex and Planet. For this analysis, two types of networks have been considered. For shallow networks of collision detection dataset most of the solvers were able to pass all test cases. In particular, SMT inspired solvers were extremely fast. On deeper networks of ACAS, most of them timeout on challenging test cases. The baseline Reluplex reached 79.26% success rate.

What I have learnt?

This paper provides a holistic view of the current approaches being used to verify piece-wise linear neural networks.Authors demonstrate how Reluplex and Planet are special cases of branch and bound verification. Finally this paper gathered all the test cases in existing literature and extended to new benchmarks.

I. Towards Scalable Verification for Safety Critical Deep Neural Networks

DNN's can be applied to a wide variety of problems owing to its black-box structure with the need to supply input and output mappings. In order for them to be applied readily in safety critical systems like autonomous driving we need to provide formal guarantees about the behaviour of the network. Recent work revealing neural network vulnerability to adversarial inputs[14] including in physical world attacks makes meeting this challenge more urgent. Kuper et al. [15] highlights two complimentary directions for verifying the modern DNN's. First, formulation of efficient and specialized SMT-solvers that can

incorporate neural network verification problems. Second, designing DNN's in such a way that they are easily amenable to verification.
Following two paragraphs discuss these two approaches in detail:

- *Scaling up SMT-based solvers for verification:* DNN's have activation functions (sigmoid,tanh) which make them learn complex functions but at the same time introduce non-linearity in the network, making verification much harder. In some cases activation functions are piece-wise linear (max-pool) which can be expressed as dis-junction of clauses. Such activation functions can exponentially increase the search space owing to its structure but this can be avoided by using a lazy splitting approach similar to what is used in Relplex.
- *Designing verification friendly networks:* Recent work in model compression [16] has shown that we can reduce the storage size of a model without compromising on the accuracy of the model. We can exploit this fact to generate smaller models which can be efficiently verified by the solver. We also modify the network topology. Networks having many layers with few neurons in each layer are easier to verify than networks having fewer layers with many neurons in each layer. Another decision associated with scalability is the selection of activation functions. Verification of networks consisting of activation functions that are piece-wise linear are much easier than networks having activation functions like sigmoid.

What I have learnt?
Verification of DNN's can be done through two complementary approaches. Either, we can modify SMT-solvers to develop domain specific theory solvers that can handle unique verification problems, or, we can design DNN's in such a way that they are more amenable to verification.

IV. Research Idea

Methods discussed in previous sections are still far from being scaled to large neural networks. Literature survey gives a clear understanding of state-of-the art techniques, I plan to combine strengths of these approaches to speed up the computation time required to verify a network.
For this I plan to work in three different directions:

- *Extend Reluplex to work with Max-Pooling activation functions*: Current implementation of Reluplex does not support Max-Pooling units. Since, Max-Pooling is a piece-wise linear activation function, it could fit well in the existing implementation of the Reluplex.
- *Propose a solution to decrease branch and bound time:* A major reason why these verification tools don't scale well to large networks is because they explore huge search spaces to find satisfiability. I believe, smart branching and a better use of bounding strategy can drastically improve the number of sub domains to explore.
- *Apply Reluplex to a trained model :* My first preference would be to select a network used in our lab or in the industry and try to verify the properties of this network. Please refer thesis proposal for further details.

Note, these three tasks should be done in a sequence. Starting from extending Reluplex to use Max-Pooling units and then implementing better branching strategy. Finally, test this tool on a trained model to verify its properties.

V. Proof of Concept

Section is divided into two parts. First, a very simple one-layer neural network is trained and then tested on its safety property. Second, employ Reluplex to verify properties of a collision detection neural network.

A. *Toy Example*

Consider a simple network shown in the Fig. 1. Suppose we want to check if we can have $x_1 \in [-2, 2]$, $x_2 \in [-2, 2]$ and output in $x_5 \in [-5, -2]$. For this problem, we expect Reluplex to return a SAT. The initial configuration of Reluplex is obtained by introducing new basic variables x_6, x_7, x_8 and encoding the network equations as:

$$x_3b - x_1 - x_2 = x_6 \qquad x_4b + x_1 + x_2 = x_7 \qquad x_3f + x_4f + x_5 = x_8$$

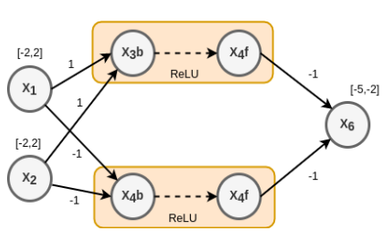

Fig. 1: Simple Neural Network

The initial tableau T_o, along with set of basic variables x_6, x_7, x_8 and set of ReLU constraints $\{(x_3b, x_3f), (x_4b, x_4f)\}$ are supplied as input to the Reluplex. Initial assignments of all variables as well

-	x1	x3b	x3f	x2	x4b	x4f	x5	x6	x7	x8
0	-	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-	-	-
7	-1	1	-	-1	-	-	-	1	-	-
8	1	-	-	1	1	-	-	-	1	-
9	-	-	1	-	-	1	1	-	-	1

Tableau T_o

as bounds for basic variables are set to 0 and the bounds for input and output variables are set according to the problem. Finally, the hidden variables are unbounded except the forward facing variables (x_3f and x_4f), by definition of ReLU are non-negative.

Fig 2 shows the final output generated by Reluplex. Clearly, x_1 = -2 and x_2 = 0 is a feasible solution. Please refer *toy-example-solution.txt*, in the supplementary material, for each pivot iteration of Reluplex.

```
      |     x1   x3b   x3f    x2   x4b   x4f    x5    x6    x7    x8 | Assignment
-------------------------------------------------------------------------------------
    x1|                                                               | -2.00 <= -2.00   <=  2.00
 B  x3b|        -1.00             -1.00                               | -4.00 <= -2.00   <=  4.00
   x3f|                                                               |  0.00 <=  0.00   <=  4.00
 B   x2|  -1.00             -1.00 -1.00                               | -2.00 <=  0.00   <=  2.00
   x4b|                                                               | -4.00 <=  2.00   <=  4.00
 B  x4f|              -1.00             -1.00 -1.00                   |  0.00 <=  2.00   <=  4.00
    x5|                                                               | -5.00 <= -2.00   <= -2.00
    x6|                                                               |  0.00 <=  0.00   <=  0.00
    x7|                                                               |  0.00 <=  0.00   <=  0.00
    x8|                                                               |  0.00 <=  0.00   <=  0.00
```

Fig. 2: Reluplex Assignments

B. *Collision Detection Network*

This network predicts collision between two vehicles that follow curved paths at different speeds. The model takes input a tuple (x, y, s, d, c_1, c_2) which is defined as follows:

- x : Relative distance of vehicles in X domain
- y : Relative distance of vehicles in Y domain
- s : speed of the second vehicle
- d starting direction of second vehicle
- c_1 and c_2 : Rotational speeds of two vehicles

Output for this neural network is $b \in \{colliding, notcolliding\}$. The test cases of collision detection are adopted from Planet verification tool. For each test case, we need to supply bounds of the six input variables in tuple (x, y, s, d, c_1, c_2) along with initial Tableau T_o. For example, one of the test cases had the following bounds:

- $x \in [0.3232066288, 0.3419566288]$
- $y \in [0.7039027312, 0.7226527312]$
- $s \in [0.2181236754, 0.2368736754]$
- $d \in [0.4188856200, 0.4376356200]$
- $c_1 \in [0.1361680057, 0.1549180057]$
- $c_2 \in [0.0074619599, 0.0262119599]$

Reluplex returns SAT for this safety test case in 195 seconds. Please refer the *summary-stats.txt* and *reluplex-run-log.txt* for detailed results on this test case.

What I have learnt?

- Reluplex reads files in a specific format. Properties of neural network can be verified only if network files are converted into *.nnet* file format. Stanford provides a well document library for such conversions.
- Understood how to provide inputs and interpret the output of Reluplex. Starting from a very simple neural network I was able to verify properties of a network trained to detect collisions.
- Verifying properties of networks having large number of neurons can take several hours. Clearly, this is not feasible if we have thousands of neurons in the network.

VI. Thesis Proposal

A. *Problem Statement*

Formal verification of neural networks can be stated as:

Given a piece-wise linear neural network N which implements a complex function $f : R^n \rightarrow R^m$, having a

set of linear constraints C over the variables $V = \{x_1, ..., x_n, y_1, ..., y_m\}$, the neural network verification problem is to find node value assignments for each variables in V over the set of constraints C for which we have $f : (x_1....x_n) = (y_1....y_m)$, or we conclude that no such node value assignment is possible.

*B. **Proposed Solution***

The methods studied in literature survey don't scale well for very large networks. In this project, I plan to exploit the strengths of studied methods to make a verification tool that runs faster on deep neural networks. We can improve scalability of tools by solving two questions:

- *How to compute better bounds?* After splitting activation functions into phases, we work on smaller domain in which we try refine all the lower and upper bounds to obtain tighter bounds. I want to explore whether we can tighten the bounds on each step at each phase in order to minimise the search space. Intuitively, stronger bounds can be obtained by looking at higher levels of the tree. Currently implementations limit their search to one level only.
- *How to perform better branching?* Current branching strategies tend to split is on phases of the activation function. Another reasonable way to split on set of inputs variables. This is feasible because functions encoded by the neural networks are piece-wise linear in the input domain.

A combination of smart branching with better bounding strategy can drastically reduced number of sub-domains to be explored. For the problems listed above, I plan to work on neural network having only piece-wise linear activation functions. This excludes non-linear activation functions like tanh and sigmoid. This should not be a problem since, majority of networks used in real world apply piece-wise linear activation functions.

The implementation of Reluplex[9] does not provide support for Max-Pooling units. Authors mentioned this extension as one of the future goals. I plan to extend the current implementation of Reluplex for Max-Pooling units. Max-Pooling units can be presented as a combination of a linear function and ReLU units. Maximum over n-numbers is equivalent to pairwise maximum of all the n elements. More formally,

$$max(x_1, x_2, ...x_n) = max(max(x_1, x_2), ..., max(x_{n-1}, x_n))$$

where, each pairwaise unit can be represent as ReLU function :

$$max(x_1, x_2) = ReLU(x_1 - x_2, 0) + x_2$$

Hence, Max Pooling units can be broken down into pair-wise max units which individually can be rewritten as ReLU functions.

*C. **Application***

Once, the tool is ready with better bound and branching strategy along with extension to Max-Pooling units, I plan to apply this tool to verify the properties of various industrial manipulators like Fanuc, Kuka and UR robots. Generally, these manipulators have 6 degrees of freedom and are being used extensively in the manufacturing and warehousing industry.

Safety is of paramount importance while working with these manipulators in industrial setting, especially in human-centered environments. These robots are required to solve its kinematics while performing operations in industry. For example, the inverse kinematics for joints are needed to be solved while performing pick and place operations. For 6 degree of freedom, there can be many solutions for joint trajectory that can help in reaching the target location. Some of these solutions can be unsafe for the environment or robot itself. Thus, such kind of scenarios require verification of solutions before execution to ensure safety of robot and its nearby environment. Following are some of the properties that can be verified:

- Due to the constraints in the environment and robot design, joints can only move in certain range of motion. To verify this, we can supply a set of inputs that could yield joint angles which are out of bounds and hence making the joints collide with each other (self-collision) or other objects in the enforcement.
- Another known problem in robotic manipulation is singularity condition. There are various kinds of singularities. The condition in which the robot is fully extended and trying to move beyond this position is known as workspace singularity. The condition in which some of the joint movements gets blocked due to inappropriate configuration is known as joint-space singularity. In singularity condition, robot's inverse kinematic solution does not exist. Sometimes robot may experience infinite joint velocities in singularity configuration, which can damage the internal motors. The singular positions of the robot can be found by analysing the Jacobian matrix of manipulator. In this case, we can verify the network for these joint positions.
- Maximum speed limit is usually set in industrial manipulators to ensure safe and jerk-free motion. In this case , we can verify for a set of inputs that can drive robots at very high speeds, say 95 % of the maximum speed.

Since safety is very important for industrial manipulators, thus it is essential to verify the above set of properties. We can encode these properties by trying to find a counter-example with the same set of constraints. For the initial set of experiments, my plan is to run everything in simulation. Due to the

high cost of real robots, its safe to test things in simulation first. After a successful attempt on a small network in simulation, my plan is to run the the extended Reluplex tool on the larger network. In this, the safety properties of the robotic manipulator in real world environment will be verified.

D. ***Resources***

- For this work, I require a powerful GPU machine to train the network and also to verify the properties of these networks.
- For real world testing, I require to work with industrial manipulators. This phase should come only after neural network has passed all the safety properties in the simulation.

E. ***Challenges***

- Improving branching and bounding strategy would surely help in decreasing the search space and increasing the scalability of the tool. I believe there are other possible directions through which we can reduce computation time significantly. I still need to read more papers to discover if such technique exists.
- Industrial robots are very expensive and not readily available to students for research purposes. I hoping that UBC Mechanical Engineering Department would have industrial robotic manipulators. I am hopeful that I can collaborate with one the research groups in mechanical engineering department for this research.

F. ***Timeline***

Following is a rough timeline of the project. This may vary according to the progress I make and unforeseen challenges I encounter.

- *May 2020 - September 2020* : Explore different approaches to find better bounds and branching strategies. Extend Reluplex to use these techniques along with extension for Max-Pooling units.
- *October 2020* : Re-run the test-cases given to original paper. Extended Reluplex should return similar results and should take lesser time than previous implementation. If something goes wrong, fix it within this time frame.
- *November 2020 - January 2020* : Train small network in simulation for industrial manipulators and/or use any pre-trained model and try to verify its safety properties. In the second phase, train a neural network for manipulator used in UBC Mechanical Engineering Department. Encode and test safety properties of this model.
- *January 2021-March 2021* Target IROS: Prepare a write-up on the approach used. Generate and compare results for all the state-of-the-art approaches used in this domain. Discuss strengths and weakness of extended Reluplex.
- *March 2021-August 2021*: Complete all experiments and thesis write up. Along with this, prepare for thesis defense.

REFERENCES

[1] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2016.

[2] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[3] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.

[4] Luca Pulina and Armando Tacchella. Challenging smt solvers to verify neural networks. *Ai Communications*, 25(2):117–135, 2012.

[5] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(3-4):209–236, 2006.

[6] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The mathsat 4 smt solver. In *International Conference on Computer Aided Verification*, pages 299–303. Springer, 2008.

[7] Bruno Dutertre and Leonardo De Moura. A fast linear-arithmetic solver for dpll (t). In *International Conference on Computer Aided Verification*, pages 81–94. Springer, 2006.

[8] Clark Barrett and Cesare Tinelli. Cvc3. In *International Conference on Computer Aided Verification*, pages 298–302. Springer, 2007.

[9] 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*, pages 97–117. Springer, 2017.

[10] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, pages 2613–2621, 2016.

[11] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[12] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[13] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, pages 4790–4799, 2018.

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[15] Lindsey Kuper, Guy Katz, Justin Gottschlich, Kyle Julian, Clark Barrett, and Mykel Kochenderfer. Toward scalable verification for safety-critical deep networks. *arXiv preprint arXiv:1801.05950*, 2018.

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.