Engineering A Verifier
for Deep Neural Networks

ThanhVu (Vu) Nguyen

October 15, 2024 (latest version available on Github)

Preface

Having been involved in PhD admission committees for many years, I've realized that many **international** students, especially those in smaller countries or less well-known universities, lack a clear understanding of the Computer Science PhD admission process at US universities. This confusion not only discourages students from applying but also creates the perception that getting admitted to a CS PhD program in the US is difficult compared to other countries.

So I want to share some details about the admission process and advice for those who are interested in applying for a **PhD in Computer Science in the US**. Originally, this document was intended for international students, but I have expanded it to include information that might also be useful for *US domestic students*. Moreover, while this is primarily intended for students interested in CS, it might be relevant to students from various STEM (Science, Technologies, Engineering, and Mathematics) disciplines. Furthermore, although many examples are specifics for schools that I and other contributors of this document know about, the information should be generalizable to other R1[1] institutions in the US.

This information can also help **US faculty and admission committee** gain a better understanding of international students and their cultural differences. By recognizing and leveraging these differences, CS programs in the US can attract larger and more competitive application pools from international students.

I wish you the best of luck. Happy school hunting!

> This document will be updated regularly to reflect the latest information and updates in the admission process. Its latest version is available at
>
> nguyenthanhvuh.github.io/phd-cs-us/demystify.pdf,
>
> and its LaTeX source is also on GitHub. If you have questions or comments, feel free to create new GitHub issues or discussions.

[1] An R1 institution in the US is a research-intensive university with a high level of research activity across various disciplines. Currently, 146 (out of 4000) US universities are classified as R1.

Contents

Chapter 1

Basic of Neural Network

A *neural network* (**NN**) [1] consists of an input layer, multiple hidden layers, and an output layer. Each layer has a number of neurons, each connected to neurons from previous layers through a predefined set of weights (derived by training the network with data). A *Deep Neural Network* (**DNN**) is an NN with at least two hidden layers.

The output of a DNN is obtained by iteratively computing the values of neurons in each layer. The value of a neuron in the input layer is the input data. The value of a neuron in the hidden layers is computed by applying an *affine transformation* (§1.1) to values of neurons in the previous layers, then followed by an *activation function* (§1.2) such as the popular Rectified Linear Unit (ReLU) activation.

1.1 Affine Transformation

The affine transformation (AF) of a neuron is the sum of the products of the weights of the incoming edges and the values of the neurons in the previous layer, plus the bias of the neuron. More specifically, the AF of a neuron y with weights $w_1, \ldots, w_n$ and bias b and the values of neurons in the previous layer $v_1, \ldots, v_n$ is $w_1v_1 + \cdots + w_nv_n + b$.

For example, the AF of a neuron x_3 in Fig. 1.1 with (incoming arrows) weights $-0.5, 0.5$ and bias 1.0 and the values of neurons in the previous layer x_1, x_2 is $-0.5x_1 + 0.5x_2 + 1.0$.

For DNN verification, AF is straightforward to reason about because it is a linear function. However, AFs are often followed by non-linear activation functions, described next in §1.2, which make the verification problem more challenging.

1.2 Activation Functions

Several popular activation functions used in DNNs include ReLU, Sigmoid, Tanh, and Softmax. All of these are non-linear[1] functions that introduce non-linearity to the network, allowing it to learn complex patterns in the data.

- ReLU (Rectified Linear Unit): ReLU is the most popular activation function in DNNs. It returns 0 if the input is less than zero, and the input itself otherwise. It is often used in hidden layers and skipped in the output layer. A ReLU activated neuron is said to be *active* if its input value is greater than zero and *inactive* otherwise.

$$ReLU(x) = \max(x, 0)$$

- Sigmoid: Sigmoid is a smooth function that maps any real value to the range (0,1). It is often used in the output layer of a binary classification problem.

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

- Tanh: Tanh is similar to the sigmoid function but maps any real value to the range (-1,1). It is often used in the output layer of a multi-class classification problem.

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Softmax: Softmax is a generalization of the sigmoid function that maps any real value to the range (0,1) and ensures that the sum of the output values is 1. It is often used in the output layer of a multi-class classification problem.

$$Softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

For DNN verification, these non-linear activation functions make verification difficult because it introduces multiple possible outcomes for any input, making it hard to reason about the output of the network. For example, ReLU has two possible outputs for any input: 0 if the input is less than zero, and the input itself otherwise, and Sigmoid has a smooth curve with infinite possible outputs for any input.

[1] Non-linear means that the output of the function is not a linear combination of its inputs.

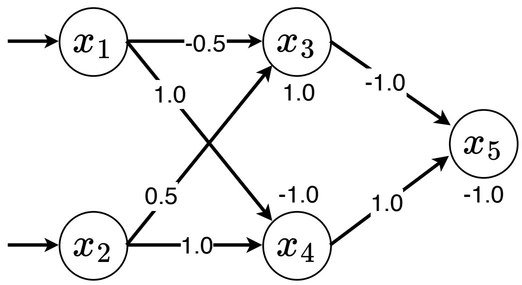

Fig. 1.1: An FNN with ReLU.

1.3 Example

Fig. 1.1 shows a simple DNN with two inputs x_1, x_2, two hidden neurons x_3, x_4, and one output x_5. The weights of a neuron are shown on its incoming edges , and the bias is shown above or below each neuron. The outputs of the hidden neurons are computed the affine transformation and ReLU, e.g., $x_3 = ReLU(-0.5x_1 + 0.5x_2 + 1.0)$. The output neuron is computed with just the affine transformation, i.e., $x_5 = -x_3 + x_4 - 1$.

1.4 Properties of Neural Networks

1.4.1 Robustness

1.4.2 Safety

Chapter 2

Verification of Neural Networks

DNN Verification Given a DNN N and a property ϕ, the *DNN verification problem* asks if ϕ is a valid property of N. Typically, ϕ is a formula of the form $\phi_{in} \Rightarrow \phi_{out}$, where ϕ_{in} is a property over the inputs of N and ϕ_{out} is a property over the outputs of N. A DNN verifier attempts to find a *counterexample* input to N that satisfies ϕ_{in} but violates ϕ_{out}. If no such counterexample exists, ϕ is a valid property of N. Otherwise, ϕ is not valid and the counterexample can be used to retrain or debug the DNN [2].

Example A valid property for the DNN in Fig. 1.1 is that the output is $x_5 \leq 0$ for any inputs $x_1 \in [-1, 1], x_2 \in [-2, 2]$. An invalid property for this network is that $x_5 > 0$ for those similar inputs. A counterexample showing this property violation is $\{x_1 = -1, x_2 = 2\}$, from which the network evaluates to $x_5 = -3.5$. Such properties can capture *safety requirements* (e.g., a rule in an collision avoidance system in [3, 5] is "if the intruder is distant and significantly slower than us, then we stay below a certain threshold") or *local robustness* [4] conditions (a form of adversarial robustness stating that small perturbations of a given input all yield the same output).

2.1 Complexity

Chapter 3

Search Algorithms

Chapter 4

Constraint Solving

4.1 SMT

4.2 MILP

Chapter 5

Abstraction

5.1 Interval

5.2 Zotope

5.3 Polytope

Chapter 6

Popular Techniques and Tools

Chapter 7

Verifying the Verifiers

Chapter 8

Conclusion

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. https://www.deeplearningbook.org, last accessed October 15, 2024.

[2] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International conference on computer aided verification*, pages 3–29. Springer, 2017.

[3] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[4] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Towards proving the adversarial robustness of deep neural networks. *Proc. 1st Workshop on Formal Verification of Autonomous Vehicles (FVAV), pp. 19-26*, 2017.

[5] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.