Neural Network Abstraction for Accelerating Verification

Stefanie Müehlberger
Technical University of Munich

***Abstract*—While abstraction is a classic tool of verification, it is not often in use for verification of neural networks. We introduce an abstraction framework applicable to feed-forward networks. For the particular case of ReLU, we can provide error bounds incurred by the abstraction. We show how the abstraction reduces the size of the network, while preserving its accuracy and how verification results on the abstract network can be transferred back to the original network.**

I. Introduction

Neural Networks (NN) are successfully used to solve many hard problems reasonably well in practice. However, there is an increasing desire to use them also in safety-critical settings, such as perception in autonomous cars [1], where reliability has to be on a very high level and that level has to be guaranteed, preferably by a rigorous proof. This is a great challenge, in particular, since NN are naturally very susceptible to adversarial attacks, as many works have demonstrated in the recent years [2], [3]. Consequently, various verification techniques for NN are being developed these days. Most verification techniques focus on proving robustness of the neural networks , i.e. for a classification task, when the input is perturbed by a small ε, the resulting output should be labeled the same as the output of the original input. Unfortunately, verification tools struggle to scale when faced with real-world neural networks. Reducing the size of a NN by abstraction leads to several possibilities. Firstly, since the abstracted NN is smaller, it may be preferred in practice because generally smaller networks are often more robust, smoother, and obviously less resource-demanding to run [4]. Note that there is a large body of work on distilling smaller NN from larger ones, e.g. [5], i.e. training a smaller NN based on the output of a bigger one. Secondly, and more interestingly in the safety-critical context, we can use the smaller abstract NN to obtain a guaranteed solution (robust or satisfying other properties) to the original problem: We can analyze the abstract NN more easily as it is smaller and then transfer the results to the original one, provided the differences are small enough.

Abstraction is one of the very classic techniques used in formal methods to obtain more understanding of a system as well as its better analysis. Disregarding small details allows for constructing a smaller system with a similar behavior. Although abstraction-based techniques are ubiquitous in verification, improving the scalability, such ideas have not been often applied to the verification of NN, except for a handful of works discussed later.

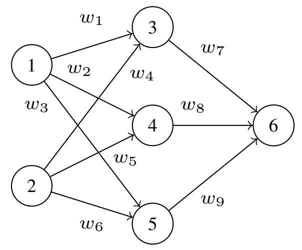
Thus, we developed an abstraction framework for NN. In contrast to syntactic similarities, such as having similar weights on the edges from the previous layer [6], our aim is to provide a behavioral, semantic notion of similarity, such as those of predicate abstraction, since such notions are more powerful.

Note that in many cases, such as recognition of traffic signs or numbers, there are finitely many (say k) interesting data points (images) on which and on whose neighborhood the network should work well. Intuitively, these are the key points that determine our focus, our scope of interest. Consequently, we propose the following equivalence on neurons. We evaluate the k inputs, yielding for each neuron a k-tuple of its activation values. This can be seen as a vector in $\mathbb{R}^k$. We stipulate that two neurons are similar if they have similar vectors, i.e, very close to each other. To determine reasonable equivalence classes over the vectors, we use the machine-learning technique of clustering, e.g. k-means [7].While other techniques, e.g. principal component analysis [8], might also be useful, simple clustering is fast and returns reasonable results.

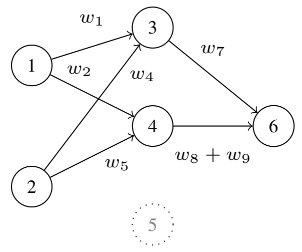
Once we have a way of determining similar neurons, we can merge each class into a single representative and obtain a smaller, abstract NN.

II. Preliminaries

We consider simple feedforward neural networks, denoted by D, consisting of one input layer, one output layer and one or more hidden layers. The layers are numbered $1, 2, \ldots, L$ with 1 being the *input layer*, L being the *output layer* and $2, \ldots, L-1$ being the *hidden layers*. Layer ℓ contains n_ℓ *neurons*. A neuron is a computation unit which takes an input $h \in \mathbb{R}$, applies an *activation function* $\phi : \mathbb{R} \to \mathbb{R}$ on it and gives as output $z = \phi(h)$. Common activation functions include tanh, sigmoid or ReLU [9], however we choose to focus on ReLU for the sake of simplicity, where ReLU(x) is defined as $\max(0, x)$. Neurons of one layer are connected to neurons of the previous and/or next layers by means of weighted connections. Associated with every layer ℓ that is not an output layer is a *weight matrix* $W^{(\ell)} = (w_{i,j}^{(\ell)}) \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$ where $w_{i,j}^{(\ell)}$ gives the weights of the connections to the i^{th} neuron in layer $\ell+1$ from the j^{th} neuron in layer ℓ. Note that $W_{i,*}^{(\ell)}$ and $W_{*,j}^{(\ell)}$ correspond to the i^{th} row and j^{th} column of $W^{(\ell)}$ respectively. The input and output of a neuron i in layer ℓ is denoted by $h_i^{(\ell)}$ and $z_i^{(\ell)}$ respectively. We call $\mathbf{h}^\ell = [h_1^{(\ell)}, \ldots, h_{n_\ell}^{(\ell)}]^\intercal$ the vector of *pre-activations* of layer ℓ



(a) Original network



(b) Network after merging neurons 4 and 5

Fig. 1: Before and after merge: neuron 4 is chosen as a representative of both 4 and 5. On merging, the incoming weights of neuron 5 are deleted and its outgoing weight is added to the outgoing weight of neuron 4.

and $\mathbf{z}^{\ell} = [z_1^{(\ell)}, \ldots, z_{n_\ell}^{(\ell)}]^\intercal$ the vector of *activations* of layer ℓ, where $z_i^{(\ell)} = \phi^{(\ell)}(h_i^{(\ell)})$. A vector $\mathbf{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$ called *bias* is also associated with all hidden layers ℓ.

In a feedforward neural network, information flows strictly in one direction: from layer ℓ_m to layer ℓ_n where $\ell_m < \ell_n$. For an n_1-dimensional input $\vec{x} \in \mathcal{X}$ from some input space $\mathcal{X} \subseteq \mathbb{R}^{n_1}$, the output $\vec{y} \in \mathbb{R}^{n_L}$ of the neural network D, also written as $\vec{y} = D(\vec{x})$ is iteratively computed as follows:

$$\begin{aligned} \mathbf{h}^{(0)} &= \vec{x} \\ \mathbf{h}^{(\ell+1)} &= W^{(\ell)}\mathbf{z}^{(\ell)} + \mathbf{b}^{(\ell+1)} \qquad (1) \\ \mathbf{z}^{(\ell+1)} &= \phi(\mathbf{h}^{(\ell+1)}) \qquad (2) \\ \vec{y} &= \mathbf{z}^{(L)} \end{aligned}$$

where $\phi(x)$ is the column vector obtained on applying ϕ component-wise to $\vec{x}$. We sometimes write $\mathbf{z}^{(\ell)}(\vec{x})$ to denote the output of layer ℓ when $\vec{x}$ is given as input to the network.

III. Abstraction

a) Merging: We propose to merge neurons which compute a similar function *on some set X of inputs*, i.e., for each input $x \in X$ to the network, they compute ε-close values. We refer to this as I/O-similarity. Further, we choose to merge neurons only within the same layer to keep the analysis and implementation straightforward. I/O-similar neurons can be merged easily without changing the behaviour of the NN too much. First, we explain the procedure on an example.

Consider the network shown in Figure 1a. The network contains 2 input neurons and 4 ReLU neurons. For simplicity, we skip the bias term in this example network. Hence, the activations of the neurons in the middle layer are given as follows:

$$\begin{aligned} z_3 &= ReLU(w_1 z_1 + w_4 z_2), \\ z_4 &= ReLU(w_2 z_1 + w_5 z_2), \\ z_5 &= ReLU(w_3 z_1 + w_6 z_2); \end{aligned}$$

and the output of neuron 6 is

$$z_6 = ReLU(w_7 z_3 + w_8 z_4 + w_9 z_5)$$

Suppose that for all inputs in the dataset, the activations of neurons 4 and 5 are 'very' close, denoted by $z_4 \approx z_5$.

Since neurons 4 and 5 behave similarly, we abstract the network by merging the two neurons as shown in Figure 1b. Here, neuron 4 is chosen as a representative of the "cluster" containing neurons 4 and 5, and the outgoing weight of the representative is set to the sum of outgoing weights of all the neurons in the cluster. Note that the incoming weights of the representative do not change. In the abstracted network, the activations of the neurons in the middle layer are now given by

$$\begin{aligned} \tilde{z}_3 &= ReLU(w_1 \tilde{z}_1 + w_4 \tilde{z}_2) = z_3 \text{ and} \\ \tilde{z}_4 &= ReLU(w_2 \tilde{z}_1 + w_5 \tilde{z}_2) = z_4 \end{aligned}$$

with neuron 5 being removed. The output of neuron 6 is therefore

$$\begin{aligned} \tilde{z}_6 &= ReLU(w_7 \tilde{z}_3 + (w_8 + w_9)\tilde{z}_4) \\ &= ReLU(w_7 z_3 + (w_8 + w_9) z_4) \\ &= ReLU(w_7 z_3 + w_8 z_4 + w_9 z_4) \approx z_6 \end{aligned}$$

which illustrates that merging preserves the behaviour of the network.

Formally, the process of merging two neurons p and q belonging to the same layer ℓ works as follows. We assume, without loss of generality, that p is retained as the representative. First, the abstract network $\tilde{D}$ is set to the original network D. Next, $\tilde{W}^{(\ell-1)}$ is set to $W^{(\ell-1)}$ with the q^{th} row deleted. Further, we set the outgoing weights of the representative p to the sum of outgoing weights of p and q, $\tilde{W}^{(\ell)}_{*,p} = W^{(\ell)}_{*,p} + W^{(\ell)}_{*,q}$. This procedure is naturally extendable to merging multiple I/O-similar neurons. It can be applied repeatedly until all desired neurons are merged.

b) Clustering: In the previous part, we saw that multiple I/O-similar neurons can be merged to obtain an abstract network behaving similar to the original network. However, the quality of the abstraction depends on the choice of neurons used in the merging. Moreover, it might be beneficial to have multiple groups of neurons that are merged separately. While multiple strategies can be used to identify such groups, in our approach we focused on one algorithm — the unsupervised learning approach of *k-means clustering* [10], as a proof-of-concept.

Our algorithm takes as input the original (trained) network D, an input set X and a function K_L, which for each layer gives the number of clusters to be identified in that layer.

Each $x \in X$ is input into $\tilde{D}$ and for each neuron i in layer ℓ, an $|X|$-dimensional vector of observed activations $\mathbf{a}_i^{(\ell)} = [z_i^{(\ell)}(x_1), \ldots, z_i^{(\ell)}(x_{|X|})]$ is constructed. These vectors of activations, one for each neuron, are collected in the set A. We can now use the k-means algorithm on the set A to identify $K_L(\ell)$ clusters. Intuitively, k-means aims to split the set A into $K_L(\ell)$ clusters such that the pairwise squared deviations of points in the same cluster is minimized. Once a layer is clustered, the neurons of each cluster are merged and the neuron closest to the centroid of the respective cluster is picked as the cluster representative. As described above, the outgoing connections of all the neurons in a cluster are added to the representative neuron of the cluster and all neurons except the representative are deleted.

c) Verification: So far, we have generated an abstract NN, on which we can run a verification algorithm, for which we chose DeepPoly [11]. Due to space issues, a description of DeepPoly is omitted here, but can be found in the original paper. Let just be noted that it is an incomplete algorithm that can only state that a property can be verified but cannot give a counterexample. After running the algorithm on an input x, we receive as a result either that the abstract NN is robust for this input or we don't get anything. Assume, that we could verify the safety property for input x on the abstracted NN $\tilde{D}$, we can now perform the *proof-lifting*, i.e. we can give error bounds for our abstraction. Given those bounds, we can then present guarantees on the original NN.

Theorem 1 (Lifting guarantees): Consider the abstraction $\tilde{D}$ obtained from a NN D. If we have the lower bound and upper bound vectors returned by DeepPoly and the vector of maximal distances of neurons from their cluster representatives. Then we can compute upper and lower bounds $\hat{u}^{(\ell)}$ and $\hat{l}^{(\ell)}$ for the activation values of all neurons in all layers.
Having upper and lower bounds for all values imposes an over-approximation of the behavior of the NN. Based on these values, one can check whether a specific property is fulfilled, just as DeepPoly does. The bounds already contain the error from the abstraction and are thus valid for the original network. This is how we can lift the satisfaction result of a specific property to the original NN.

IV. Results

We performed experiments on different networks and different datasets. With abstraction, we are able to reduce the size of the networks up to 50%. By using DeepPoly, we can show that the reduction of the size correlates with the reduction of time that is needed for verification. Thus, we can speed up the verification time in the best case by the factor 10.

Additionally, we ran experiments to demonstrate the working of the full verification pipeline — involving clustering to identify the neurons that can be merged, performing the abstraction, running DeepPoly on the abstraction and finally lifting the verification proof to answer the verification query on original network.

We were interested in two parameters: (i) the time taken to run the full pipeline; and (ii) the number of verification queries that could be satisfied (out of 200). We ran experiments on a network with 6 layers and 300 nodes in each trained on the MNIST-dataset [12]. It could be verified to be locally robust for 197/200 images in 48 minutes by DeepPoly. In the best case, our preliminary implementation of the full pipeline was able to verify robustness for 195 images in 36 minutes — 13s for clustering and abstracting, 35 min for verification, and 5s for proof lifting. In other words, a 14.7% reduction in network size produced a 25% reduction in verification time. When we pushed the abstraction further to obtain a reduction of 19.4% in the network size, DeepPoly could still verify robustness of the abstracted network for 196 images in just 34 minutes (29% reduction). However, in this case, the proof could not be lifted to the original network as the over-approximations we obtained were too coarse.

V. Related Work

In contrast to compression techniques, our abstraction provides a mapping between original neurons and abstract neurons, which allows for transferring the claims of the abstract NN to the original one, and thus its verification.

The very recent work [13] suggests an abstraction, which is based solely on the sign of the effect of increasing a value in a neuron. While we can demonstrate our technique on e.g. 784 dimension input (MNIST) and work with general networks, [13] is demonstrated only on the Acas Xu [14] networks which have 5 dimensional input; our approach handles thousands of nodes while the benchmark used in [13] is of size 300. Besides, we support both classification and regression networks. Finally, our approach is not affected by the number of outputs, whereas the [13] grows exponentially with respect to number of outputs.

Further, [16] computes a similarity measure between incoming weights and then starts merging the most similar ones. It also features an analysis of how many neurons to remove in order to not lose too much accuracy. However, it does not use clustering on the semantic values of the activations, but only on the syntactic values of the incoming weights, which is a very local and thus less powerful criterion.

Similarly, [6] clusters based on the incoming weights only and does not bound the error. [17] clusters weights in contrast to our activation values) using the k-means clustering algorithm. However, the focus is on weight-sharing and reducing memory consumption, treating neither the abstraction mapping nor verification.

VI. Outlook

The most important difference to other compression frameworks is that this approach provides a semantic link between the original states and the abstract states. This in principle allows for abstraction-based verification of neural networks. The next steps are to provide more practical heuristics and to extend the framework to convolutional networks and other layer types, as well as trying different compression techniques than just clustering, e.g. principal component analysis.

References

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[2] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[3] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9185–9193.

[4] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *stat*, vol. 1050, p. 9, 2015.

[6] G. Zhong, H. Yao, and H. Zhou, "Merging neurons for structure compression of deep networks," in *ICPR*. IEEE Computer Society, 2018, pp. 1462–1467.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[8] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[9] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, vol. 30, no. 1, 2013, p. 3.

[10] C. M. Bishop, *Pattern recognition and machine learning, 5th Edition*, ser. Information science and statistics. Springer, 2007.

[11] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," *PACMPL*, vol. 3, no. POPL, pp. 41:1–41:30, 2019.

[12] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[13] Y. Yisrael Elboher, J. Gottschlich, and G. Katz, "An Abstraction-Based Framework for Neural Network Verification," *arXiv e-prints*, Oct 2019.

[14] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *CoRR*, vol. abs/1810.04240, 2018.

[15] P. Prabhakar and Z. R. Afzal, "Abstraction based output range analysis for neural networks," in *NeurIPS*, 2019.

[16] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *BMVC*, 2015.

[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016.