Fine-Grained Neural Network Abstraction for Efficient Formal Verification

Zhaosen Wen[1], **Weikai Miao**[1,2], **Min Zhang**[1,2]

[1] Shanghai Key Laboratory for Trustworthy Computing, East China Normal University
[2] Shanghai Institute of Intelligent Science and Technology, Tongji University
51184501057@stu.ecnu.edu.cn, {wkmiao,zhangmin}@sei.ecnu.edu.cn

Abstract

The advance of deep learning makes it possible to empower safety-critical systems with intelligent capabilities. However, its intelligent component, i.e., deep neural network, *is difficult to formally verify due to the large scale and intrinsic complexity of the verification problem. Abstraction has been proved to be an effective way of improving the scalability. A challenging problem in abstraction is that it is difficult to achieve a balance between the size reduced and output overestimation caused by abstraction. In this work, we propose an effective fine-grained approach to abstract neural networks. Our approach is fine-grained in that we identify four cases that should be abstracted independently under a certain neuron prioritization strategy. This allows us to merge more neurons in networks and meanwhile maintain a relatively low output overestimation. Experimental results show that our approach outperforms other existing abstraction approaches by significantly reducing the scale of target deep neural networks with small overestimation.*

1 Introduction

In recent years, Deep Neural Networks (DNNs) have been achieving remarkable performance in many complex tasks and are increasingly deployed in safety-critical systems, such as autonomous vehicle [2], face recognition [3], airborne collision avoidance system [11]. However, it is well known that DNNs are vulnerable to slight perturbations, i.e., adding imperceptible perturbations to inputs may cause DNN to make mistakes [17, 18, 8, 5]. Statistical results show that the accident frequency of autonomous vehicles is much higher than that of conventional vehicles [7]. As safety-critical systems require strict safety and reliability guarantees, it raises a new problem of certifying the trustworthiness of the intelligent components, i.e., DNNs.

Formal methods have been proved their effectiveness in certifying DNNs. For rigorousness, formal methods guarantee a DNN satisfies a property if the property is proved to be true, and otherwise counterexamples are computed as witnesses to the violation. In the context of neural network verification, a counterexample is called an adversarial example that causes misclassification to DNN. The literature on formal verification of neural network is booming in the past several years. Details can be referred to the survey [10].

Most of the existing neural network verification approaches suffer from bad scalability issue due to the intrinsic complexity of neural networks. Katz *et al.* [12] showed that the verification problem of even simple fully connected feedforward neural networks taking ReLU activation function is NP-complete. Abstraction is one of the effective approaches to scale up verification algorithms. The basic idea of abstraction is to tune concrete constraints into abstract ones which can be solved more efficiently [9, 15, 8, 1, 6, 13]. Abstraction must preserve soundness, i.e., a property proved in the abstract system implies the concrete system satisfies that property.

One promising abstraction technique for neural network verification is to construct Interval Neural Networks (INNs) [13] to abstract ordinary deep neural networks. Intuitively, an INN takes intervals as inputs, unlike DNN whose inputs are concrete values. An INN can be constructed by merging neurons in the same hidden layer [13]. The decrease of neurons makes it faster to verify an INN than to verify its corresponding DNN. There are two criteria for evaluating an abstraction approach, i.e., the number of neurons that are merged, and the overestimation of output interval. An approach that can merge more neurons with lower overestimation is more preferred than the one that merges fewer neurons with larger overestimation.

In this paper, we propose a novel fine-grained abstraction approach to abstract feedforward neural networks that take ReLU as activation function into INNs for the purpose of improving the efficiency of their formal verification. In our approach, we classify the merging of neurons into four cases according to the signs of weights, and propose the corresponding merging rules for each case. We also devise a strategy for determining the priority of neurons to be merged. Before the abstraction of neural network, we compute an indicator for each pair of neurons and priori-

DOI reference number: 10.18293/SEKE2021-071

tize neurons according to the indicator. We prove that our abstraction approach is sound. We implement the approach into a tool called NNZipper, and evaluate it on the benchmark of neural networks trained on MNIST [16] and ACAS Xu [11]. Experimental results show that our approach can significantly reduce the scale of a neural network with low overestimation induced, compared with the pioneering abstraction approach in [13].

2 Preliminaries

2.1 Deep Neural Network (DNN)

A deep neural network is a model consisting of an input layer, several hidden layers and an output layer. Except for input layer, each layer contains some neurons, which are connected to the neurons in the preceding layer. Each edge has a weight. The neurons in input layer receive input data, and the neurons in the next layer get their values by computing a dot product of the values of preceding layer and edge weights, with the addition of a bias, and then operated by an activation function such as ReLU. After layer-by-layer calculation, the output layer gives the result of DNN.

Definition 1 (DNN). *An n-layer DNN is a triple* $(\{S_i\}_{0\le i\le n}, \{W_i\}_{1\le i\le n}, \{B_i\}_{1\le i\le n})$, *where*

- S_i *is the set of neurons in the i-th layer.* S_0 *denotes the input layer,* S_n *denotes the output layer.*
- W_i *denotes the weight matrix between the* $i-1$*-th layer and i-th layer.*
- B_i *denotes biases vector of the i-th layer.*

2.2 Interval Neural Network (INN)

In an interval neural network (INN), the edge weights and biases are not in value form but interval form.

Definition 2 (INN [13]). *An n-layer INN is a triple* $(\{S_i\}_{0\le i\le n}, \{W_i^l, W_i^u\}_{1\le i\le n}, \{B_i^l, B_i^u\}_{1\le i\le n})$, *where*

- S_i *is the set of neurons in the i-th layer.* S_0 *denotes the input layer,* S_n *denotes the output layer.*
- W_i^l, W_i^u *denote the lower weight matrix and upper weight matrix respectively between the* $i-1$*-th layer and i-th layer, which satisfy* $W_i^l \le W_i^u$ [1].
- B_i^l, B_i^u *denote the lower biases vector and upper biases vector respectively of the i-th layer,* $B_i^l \le B_i^u$.

The input of INN is a set of intervals, as well as the output. The output is computed by solving several maximization and minimization problems built on the lower and upper weights and biases. It is proved that a DNN can be abstracted to an INN with smaller size [13]. For an identical input region, the output of the INN is an over-approximation of the output range of the original DNN.

[1]For matrix A and B of the same size, $A \le B$ means $\forall i, j, a_{i,j} \le b_{i,j}$. Similarly for vector.)

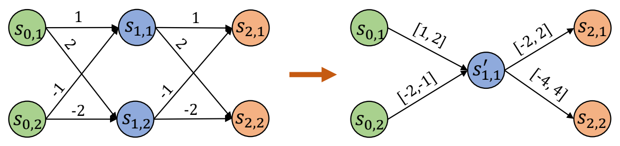


Figure 1: A simple example for abstracting DNN into INN.

2.3 Neural Network Abstraction

Neural network abstraction is a technique for compressing a neural network into a smaller one. Despite the loss of some precision, a smaller network is usually preferred to deploy on edge devices and to formally verify. To abstract a DNN into a smaller INN, several neurons in the same hidden layer are merged with their weights and biases merged into intervals. The state-of-the-art approach takes the convex hull of the original weights as the weight interval for the neuron abstracted from original neurons [13], which is valid and fast but induces considerable imprecision in output range computation.

Figure 1 gives a simple example of abstracting a DNN into an INN by merging neurons and weights. When merging outgoing weights, the convex hull of original weights needs to be multiplied by 2 (equal to the number of neurons merged) to guarantee validity.

3 Fine-Grained Abstraction

In this section, we present our fine-grained abstraction approach to transforming a DNN into an INN, and meanwhile guarantee that a constructed INN is an overapproximation of its original DNN. The property of the abstraction guarantees the soundness of verifying abstracted INNs.

3.1 Abstracting DNN into INN

The complexity of the output range computation of a DNN is strongly related to its size, i.e., the number of all the neurons in the DNN. Our abstraction method aims to decrease the size of the network, and get an overapproximation of the network's output range by computing the abstract network's output range. To accomplish this, several pairs of neurons in the same hidden layer are merged into a single neuron with their weights and biases also merged. The new neuron's weights and bias are not values but intervals, which are calculated based on the weights and biases of original neurons. Note that DNN can be regarded as a special kind of INN whose weights and biases are degenerate intervals, i.e., the lower bound and upper bound of the interval are the same. Hence we will describe the details of the abstraction based on the semantics of INN.

Given an INN $(n, \{S_i\}_{0\le i\le n}, \{W_i^l, W_i^u\}_{1\le i\le n}, \{B_i^l, B_i^u\}_{1\le i\le n})$, let $s_{i,p}$ denote the p-th neuron in layer i, and $w_{i,p}^l, w_{i,p}^u$ denote the lower weight vector and upper weight vector of neuron $s_{i,p}$, and $w_{i,p\lessdot q}^l, w_{i,p\lessdot q}^u$ denote the lower weight and upper

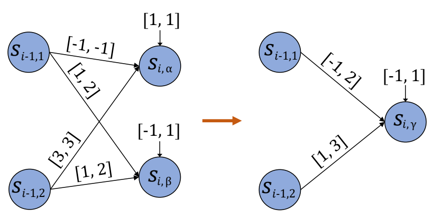


Figure 2: Example of merging biases and incoming edges

weight between the q-th neuron in layer $i-1$ and the p-th neuron in layer i. We use $v_{i,p}$ to denote the valuation interval of the p-th neuron in layer i, and $b^l_{i,p}, b^u_{i,p}$ to denote the lower bias and upper bias of the p-th neuron in layer i. The problem of abstracting a DNN by an INN is merging two neurons $s_{i,\alpha}$ and $s_{i,\beta}$ into a new neuron $s_{i,\gamma}$ ($1 \leq i \leq n-1$).

Our first step is to merge the biases of $s_{i,\alpha}$ and $s_{i,\beta}$ and the edges between layer $i-1$ and layer i. The requirement of this step is guaranteeing the valuation interval of the new neuron containing the valuation interval of the original neurons, i.e., $v_{i,\alpha} \subseteq v_{i,\gamma}$, $v_{i,\beta} \subseteq v_{i,\gamma}$. To reach this goal, the new neuron's bias interval is obtained by taking the convex hull of the two original neurons' bias intervals. Specifically, the smaller of the original lower bounds will be the new lower bound, and the greater of the original upper bounds will be the new upper bound. The merging of the edge weights is similar. For each new edge between layer $i-1$ and layer i, the new weight interval of the edge is obtained by taking the convex hull of the corresponding original weight intervals. Formally, we have the following equations:

$$\textbf{Bias: } b^l_{i,\gamma} = min(b^l_{i,\alpha}, b^l_{i,\beta}), b^u_{i,\gamma} = max(b^u_{i,\alpha}, b^u_{i,\beta});$$

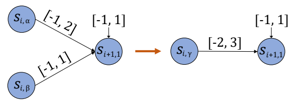
$$\textbf{Weights: } \forall s_{i-1,p} \in S_{i-1}, w^l_{i,\gamma\triangleleft p} = min(w^l_{i,\alpha\triangleleft p}, w^l_{i,\beta\triangleleft p}),$$

$$w^u_{i,\gamma\triangleleft p} = max(w^u_{i,\alpha\triangleleft p}, w^u_{i,\beta\triangleleft p}).$$

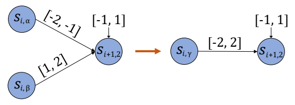
Figure 2 shows a simple example of the first step. After the merging, the new weight interval $[-1, 2]$ is the convex hull of $[-1, -1]$ and $[1, 2]$, and the other new weight interval $[1, 3]$ is similar. The new bias interval $[-1, 1]$ is also the convex hull of the original bias intervals $[1, 1]$ and $[-1, 1]$.

Our second step is to merge the edges between layer i and layer $i+1$. The requirement of this step is guaranteeing the new valuation interval of each neuron in layer $i+1$ containing its original valuation interval, i.e., $\forall s_{i+1,q} \in S_{i+1}, v'_{i+1,q} \supseteq v_{i+1,q}$. According to the sign of the weights of original neurons, the merging in this step will follow different rules. Without loss of generality, we assume that the lower bound of the weight interval for $s_{i,\alpha}$ is not greater than that of $s_{i,\beta}$, i.e., $w^l_{i+1,q\triangleleft\alpha} \leq w^l_{i+1,q\triangleleft\beta}$. Then the merging can be classified into four cases:

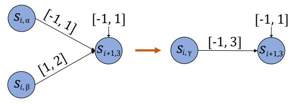
Case 1 We first consider the case when $w^l_{i+1,q\triangleleft\alpha}, w^l_{i+1,q\triangleleft\beta}$ have the same sign, and $w^u_{i+1,q\triangleleft\alpha}, w^u_{i+1,q\triangleleft\beta}$ have the same sign. The lower bound of the new weight interval is obtained by



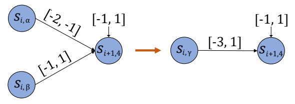
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 3: Examples of merging outgoing edges in four cases

taking the sum of the two original lower bounds. The new upper bound is computed likewise. Formally, we have

$$w^l_{i+1,q\triangleleft\gamma} = w^l_{i+1,q\triangleleft\alpha} + w^l_{i+1,q\triangleleft\beta}, \text{ and}$$
$$w^u_{i+1,q\triangleleft\gamma} = w^u_{i+1,q\triangleleft\alpha} + w^u_{i+1,q\triangleleft\beta}.$$

Figure 3a shows an example of the case. The new lower bound -2 is the sum of original lower bounds -1 and -1, and the new upper bound 3 is the sum of 2 and 1.

Case 2 In the second case, we consider that $w^l_{i+1,q\triangleleft\alpha} < 0$, $w^u_{i+1,q\triangleleft\alpha} < 0$, $w^l_{i+1,q\triangleleft\beta} \geq 0$, and $w^u_{i+1,q\triangleleft\beta} \geq 0$. In this case, the lower bound of the new weight interval is equal to the lower bound of the original weight interval for $s_{i,\alpha}$. The upper bound of the new weight interval is equal to the original upper bound for $s_{i,\beta}$. Formally, they are defined as follows:

$$w^l_{i+1,q\triangleleft\gamma} = w^l_{i+1,q\triangleleft\alpha}, \text{ and}$$
$$w^u_{i+1,q\triangleleft\gamma} = w^u_{i+1,q\triangleleft\beta}$$

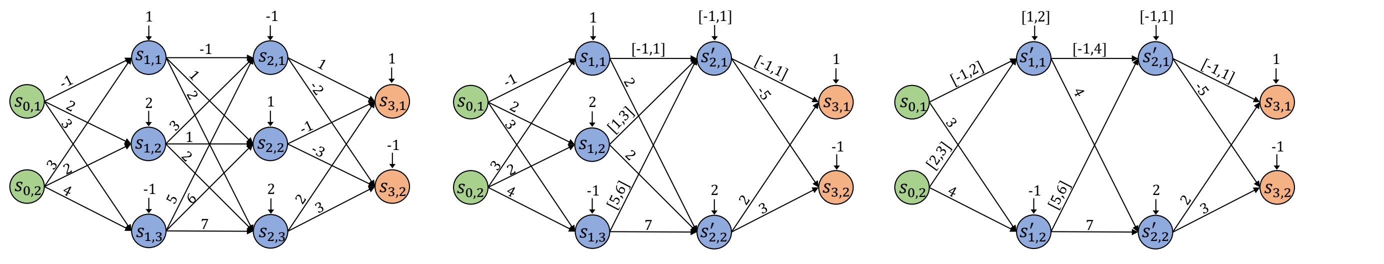
Figure 3b shows an example of this case. The new lower bound -2 is derived from the lower bound of the weight interval [-2,-1], and the new upper bound 2 is derived from the upper bound of the weight interval [1,2].

Case 3 The third case considers $w^l_{i+1,q\triangleleft\alpha} < 0$, $w^u_{i+1,q\triangleleft\alpha} \geq 0$, $w^l_{i+1,q\triangleleft\beta} \geq 0$, and $w^u_{i+1,q\triangleleft\beta} \geq 0$. In this case, the lower bound of the weight interval after merging is equal to the lower bound of the original weight interval for $s_{i,\alpha}$. The upper bound of the new weight interval is set the sum of the two original upper bounds. Formally, we have

$$w^l_{i+1,q\triangleleft\gamma} = w^l_{i+1,q\triangleleft\alpha}, \text{ and}$$
$$w^u_{i+1,q\triangleleft\gamma} = w^u_{i+1,q\triangleleft\alpha} + w^u_{i+1,q\triangleleft\beta}.$$

Figure 3c shows an example of the case. The new lower bound -1 is derived from the lower bound of the weight interval [-1,1], and the new upper bound 3 is the sum of original upper bounds 2 and 1.

Case 4 The last case is that $w^l_{i+1,q\triangleleft\alpha} < 0$, $w^u_{i+1,q\triangleleft\alpha} < 0$, $w^l_{i+1,q\triangleleft\beta} < 0$, and $w^u_{i+1,q\triangleleft\beta} \geq 0$. In this case, the lower bound of the new weight interval takes the sum of the two original lower bounds. And the upper bound of the new weight in-



(a) Original DNN (b) The INN after merging $s_{2,1}$ and $s_{2,2}$ (c) Final INN

Figure 4: The process of merging two pairs of neurons for a small neural network

terval is equal to the original upper bound for $s_{i,\beta}$. Formally, they are defined by the following equations:

$$w^l_{i+1,q\triangleleft\gamma} = w^l_{i+1,q\triangleleft\alpha} + w^l_{i+1,q\triangleleft\beta}$$
$$w^u_{i+1,q\triangleleft\gamma} = w^u_{i+1,q\triangleleft\beta}$$

Figure 3d shows an example of the last case. The new lower bound is -3, *i.e.,* the sum of original lower bounds -2 and -1. The new upper bound is 1, which is derived from the upper bound of the weight interval [-1,1] by the definition.

Example 1. *Let us consider the neural network in Figure 4a. We want to merge $s_{2,1}$ and $s_{2,2}$, and then $s_{1,1}$ and $s_{1,2}$. For both mergings, the new weight intervals of the preceding edges are obtained by taking the convex hull of the corresponding original weights. We focus on the merging of succeeding edges. In the merging of $s_{2,1}$ and $s_{2,2}$, because $1 > 0, -1 < 0$, the weight interval $[-1, 1]$ between $s'_{2,1}$ and $s_{3,1}$ is obtained by applying the rules of case 2. And the weight -5 between $s'_{2,1}$ and $s_{3,2}$ is obtained by applying the rules of case 1 because -2 and -3 are both negative. Then in the merging of $s_{1,1}$ and $s_{1,2}$, the weight interval $[-1, 4]$ is obtained by applying the rules of case 3, and the weight 4 by applying the rules of case 1.*

Our method can abstract a DNN into an INN with arbitrary size. However, merging too many neurons will lead to an excessive output range. We need to make a trade-off between the abstraction scale and output overestimation.

3.2 Neuron Prioritization Strategy

The abstraction method above can merge any pair of neurons in hidden layers. The inaccuracy induced by a merging operation depends on the differences of the weights and biases of the original neurons. To get an overapproximation with lower inaccuracy, we present a heuristic strategy for prioritizing the pairs of neurons to merge.

Algorithm 1 sketches the overall process. For each pair of neurons in hidden layers, we compute a value m, which takes the sum of the absolute values of the differences of corresponding incoming weights, with the addition of the absolute value of the difference between the two biases. Then we take m as the indicator and construct a min priority queue Q to guide the abstraction. When performing the abstraction, We repeatedly pop the priority queue and merge the corresponding pair of neurons in the network.

Algorithm 1 Neuron Prioritization

```
Require: a DNN $D$
Ensure: a min priority queue $Q$
1: $Q \leftarrow \perp$
2: for every pair of hidden neurons $s_{i,\alpha}$,$s_{i,\beta}$ do
3:     $m \leftarrow |b_{i,\alpha} - b_{i,\beta}|$
4:     for every neuron $s_{i-1,p}$ do
5:         $m+ = |w_{i,\alpha\triangleleft p} - w_{i,\beta\triangleleft p}|$
6:     end for
7:     Add ($m$, $s_{i,\alpha}$,$s_{i,\beta}$) to Q
8: end for
```

3.3 Overapproximation

We show an INN abstracted in our approach is an overapproximation of its original DNN. It implies the soundness of verifying the DNN by verifying the INN instead.

Definition 3 (Overapproxiamtion)**.** *Given an INN A and a DNN D, A is an overappximation of D if and only if for any input interval I, there is $D(I, \ell) \subseteq A(I, \ell)$ for any label ℓ.*

According to the definition, it is apparent that overapproximation is transitive for the transitivity of $\subseteq$.

Lemma 1 (One-step overapproximation)**.** *Given an INN A, let $\hat{A}$ be the INN abstracted from A by merging a pair of neurons. $\hat{A}$ is an overapproximation of A.*

Proof. Consider merging $s_{i,1}$ and $s_{i,2}$ into $\hat{s}_{i,1}$. Let V_i denote the valuation vector of layer i, $v_{i,q}$ denote the valuation interval of the q-th neuron in layer i, $v^l_{i,q}$ denote the lower bound and $v^u_{i,q}$ denote the upper bound.

First we prove the correctness of the first step of merging. Initially, we have $v^l_{i,1} = ReLU(w^l_{i,1}V_{i-1} + b^l_{i,1})$, $v^l_{i,2} = ReLU(w^l_{i,2}V_{i-1}+b^l_{i,2})$. After merging, $\hat{v}^l_{i,1} = ReLU(\hat{w}^l_{i,1}V_{i-1}+\hat{b}^l_{i,1})$, $\hat{w}^l_{i,1} = min(w^l_{i,1}, w^l_{i,2})$, $\hat{b}^l_{i,1} = min(b^l_{i,1}, b^l_{i,2})$. Because V_{i-1} is non-negative and ReLU is monotonic, there are $\hat{v}^l_{i,1} \leq v^l_{i,1}$ and $\hat{v}^l_{i,1} \leq v^l_{i,2}$. Likewise, we have $\hat{v}^u_{i,1} \geq v^u_{i,1}, \hat{v}^u_{i,1} \geq v^u_{i,2}$. Consequently, $v_{i,1} \subseteq \hat{v}_{i,1}, v_{i,2} \subseteq \hat{v}_{i,1}$.

Then we prove the correctness of the second step. We first consider Case 1, for an arbitrary neuron $s_{i+1,q}$, we use $c_{i+1,q}$ and $\hat{c}_{i+1,q}$ to denote the merged neurons' contribution to its valuation interval, *i.e.*, $c^l_{i+1,q} = ReLU(w^l_{i,q\triangleleft 1}v^l_{i,1} + w^l_{i,q\triangleleft 2}v^l_{i,2} + b^l_{i+1,q})$. After merging, $\hat{c}^l_{i+1,q} = ReLU((w^l_{i,q\triangleleft 1} + w^l_{i,q\triangleleft 2})\hat{v}^l_{i,1} + b^l_{i+1,q})$. Because $\hat{v}^l_{i,1} \leq v^l_{i,1}, \hat{v}^l_{i,1} \leq v^l_{i,2}$ and the monotonicity of ReLU, we have $\hat{c}^l_{i+1,q} \leq c^l_{i+1,q}$. Similarly, $\hat{c}^u_{i+1,q} \geq c^u_{i+1,q}$. Thus, $c_{i+1,q} \subseteq \hat{c}_{i+1,q}$. Because other neurons connected to $s_{i+1,q}$ are not altered, we have $v_{i+1,q} \subseteq \hat{v}_{i+1,q}$.

We can prove that $v_{i+1,q} \subseteq \hat{v}_{i+1,q}$ holds in other three cases likewise. Consequently, we have $A(I,\ell) \subseteq \hat{A}(I,\ell)$ for any label ℓ of A. Thus, $\hat{A}$ is an overapproximation of A. $\square$

For the transitivity of overapproximation, it is straightforward to obtain the following theorem from Lemma 1.

Theorem 1 (Overapproximation)**.** *Given a DNN D, let A be the INN abstracted from D in our approach. A is an overapproximation of D.*

Theorem 1 can be proved directly using Lemma 1 based on the fact that the abstraction is a finite-step process. We omit the proof due to space limitation.

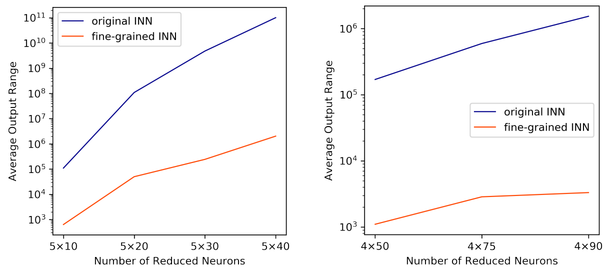
4 Implementation and Evaluation

We implement our framework in Python and use a state-of-the-art MILP solver Gurobi to solve the minimization and maximization problems.

Evaluation Datasets. We consider two benchmarks, ACAS Xu networks [11] which consists of 6 hidden layers with 50 neurons in each layer and a 5×100 network trained on MNIST in [16]. We generate a set of random valid input regions as the evaluation dataset for the ACAS Xu network. For the MNIST network, we choose the first 100 images in the MNIST test set. Each input image can be perturbed in an l_∞ norm form ball with a bound ϵ.

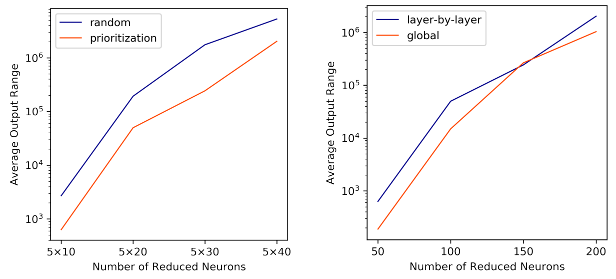
Experimental Setup. All the experiments were conducted on a workstation with a 32-core 3.7 GHz AMD Ryzen Threadripper 3970X CPU and 128GB RAM. We set a timeout one hour for the verification of each input region.

Evaluation Result. Firstly, we compare the performance of our fine-grained INN abstraction to the original INN abstraction. Neuron prioritization strategy is applied to both methods to ensure they merge the same neurons. The abstractions are parameterized by $n \times k$, where k is the number of reduced neurons in each layer and n is the number of hidden layers except the first hidden layer. We show the result of one of the ACAS Xu networks in Figure 5a and the result of the MNIST network in Figure 5b. The results of other benchmarks are similar. Our fine-grained INN has a great improvement in the precision of computation. The average output range computed by fine-grained INN is over



(a) ACAS Xu network (6x50) (b) MNIST network (5x100)

Figure 5: Comparison with the abstraction approach in [13]



(a) Prioritization versus random (b) Global versus layer-by-layer

Figure 6: Comparison of different strategies

two orders of magnitude smaller than the output range computed by original INN. With the increase of reduced neurons, the growth of the output range of fine-grained INN is much lower than that of original INN. In the cases with fewer reduced neurons, e.g., the 4×50 case for MNIST network, the average running time of fine-grained INN (77s) is several times longer than that of original INN (30s). However, with the increase of reduced neurons, due to the sharp fall in computation complexity, the average running time of the both become almost the same.

To show the effectiveness of neuron prioritization strategy, we compare our strategy with a random one. Both of them are based on fine-grained INN abstraction. Figure 6a shows the result. The output range of prioritization strategy is several times smaller than that of random strategy, which demonstrates our neuron prioritization strategy is very effective to improve the performance of INN abstraction.

When using neuron prioritization strategy, we have two sub-strategies to estimate the neurons. One is *layer-by-layer*, where we give a fixed number k as the number of reduced neurons for each layer, and select the best k pairs of neurons in each layer to be merged. The other is *global*, where we give a number j as the total number of reduced neurons, then select the top j pairs from all hidden layers. Figure 6b depicts the comparison between layer-by-layer strategy and global strategy, which are based on fine-grained INN abstraction for ACAS Xu networks. In our experiments, the global strategy usually performs better than layer-by-layer strategy. We find that in these cases, the global strategy mainly merges the neurons in layer 6, 5 and 4. In few cases where layer-by-layer strategy performs better, we find the global strategy merges more neurons in layer 2 and 3 than layer-by-layer strategy. Thus we deduce that merging neurons in the front layers has more influence on the output range computation than merging neurons in the latter layers. That is because the over-estimation induced by merging is amplified layer by layer.

5 Related Work

Our work is inspired by many pioneering neural network abstraction approaches. Our approach is in line with but outperforms the abstraction approach in [13] in terms of the induced overestimation and the size of reduced neurons. Katz *et al.* [6] proposed an abstraction technique for merging neurons in neural networks to accelerate the verification of ACAS Xu networks. The difference is that in their approach the weights after merging are still values, unlike intervals in our approach. Another approach merges the neurons with similar behaviors, i.e., the neurons' values are always similar for a given set of inputs [1]. This approach relies on concrete inputs of the neural networks to abstract, while our approach is independent of inputs.

Another class of abstraction-based neural network verification approaches rely on the abstraction of the constraints transformed from original neural networks, but not the abstraction of neural networks. Representative approaches include abstraction interpretation [9, 14] and linear relaxation and overapproximation [4, 19]. After abstraction, they resort to efficient linear programming solvers such as SMT and MILP solvers to check the satisfiability of abstracted constraints. Like our approach, all these approaches are sound, but refinement is needed to achieve completeness.

6 Conclusion and Future Work

We have presented a fine-grained approach to abstract neural networks for efficient formal verification. We identified four cases of merging neurons in neural networks and defined corresponding merging rules. We also introduced a neuron prioritization strategy to reduce the overestimation induced by the abstraction. Compared with the pioneering merging approach in the work [13], our approach can significantly reduce the scale of original neural networks while cause a relatively low output overestimation.

As for future work, we are planning to apply our approach to the formal verification of real-world large-scale neural networks. Further, we consider extending it to other network architectures and non-ReLU activation functions.

Acknowledgments

This work is supported by National Key Research and Development Program (2020AAA0107800), Joint Funding and AI Project (20DZ1100300) of Shanghai Science and Technology Committee, NSFC general projects (61872146, 61872144), and Open Project Fund from Shenzhen Institute of Artificial Intelligence and Robotics for Society. Weikai Miao and Min Zhang are the corresponding authors.

References

[1] Pranav Ashok, Vahid Hashemi, Jan Kretínský, and Stefanie Mohr. Deepabstract: Neural network abstraction for accelerating verification. In *ATVA 2020*, volume 12302, pages 92–107, 2020.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, et al. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[3] Naser Damer, Yaza Wainakh, Olaf Henniger, Christian Croll, Benoit Berthe, et al. Deep learning-based face recognition and the robustness to perspective distortion. In *24th ICPR*, pages 3445–3450, 2018.

[4] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138, 2018.

[5] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *ACL 2018*, pages 31–36, 2018.

[6] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In *CAV 2020*, pages 43–65, 2020.

[7] Francesca M. Favarò, Nazanin Nader, Sky O. Eurich, Michelle Tripp, and Naresh Varadaraju. Examining accident reports involving autonomous vehicles in california. *Plos One*, 12(9):e0184952, 2017.

[8] Samuel G. Finlayson, Isaac S. Kohane, and Andrew L. Beam. Adversarial attacks against medical deep learning systems. *CoRR*, abs/1804.05296, 2018.

[9] 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 *(S&P'18)*, pages 3–18. IEEE, 2018.

[10] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Comput. Sci. Rev.*, 37:100270, 2020.

[11] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *IEEE/AIAA 35th DASC*, pages 1–10, 2016.

[12] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV 2017*, pages 97–117, 2017.

[13] Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. In *NeurIPS'19*, 2019.

[14] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. Fast and effective robustness certification. In *NeurIPS 2018*, pages 10825–10836, 2018.

[15] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *POPL'19*, 3:1–30, 2019.

[16] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *ICLR'19*, 2019.

[17] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd ICLR*, 2014.

[18] Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. A game-based approximate verification of deep neural networks with provable guarantees. *Theoretical Computer Science*, 807:298–329, 2020.

[19] Yiting Wu and Min Zhang. Tightening robustness verification of convolutional neural networks with fine-grained linear approximation. In *AAAI'21*, 2021.