Verifying Equivalence Properties of Neural Networks with ReLU Activation Functions

Marko Kleine Büning(✉), Philipp Kern, and Carsten Sinz

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{marko.kleinebuening,carsten.sinz}@kit.edu,
ufedz@student.kit.edu

Abstract. Neural networks have become popular methods for tackling various machine learning tasks and are increasingly applied in safety-critical systems. This necessitates verified statements about their behavior and properties. One of these properties is the equivalence of two neural networks, which is important, e.g., when neural networks shall be reduced to smaller ones that fit space and memory constraints of embedded or mobile systems.

In this paper, we present the encoding of feed-forward neural networks with ReLU activation functions and define novel and relaxed equivalence properties that extend previously proposed notions of equivalence. We define ϵ- and *top-k*-equivalence and employ it in conjunction with restricting the input space by hierarchical clustering. Networks and properties are encoded as mixed integer linear programs (MILP). We evaluate our approach using two existing reduction methods on a neural network for handwritten digit recognition.

1 Introduction

The popularity of neural networks (NNs) for solving machine learning tasks has strongly increased with the availability of high performance computers and large data sets produced by today's society. Nowadays, NNs are considered state of the art solutions for many machine learning tasks, including machine translation [2], image processing [19] or playing games like Go and chess [23]. The complex structure of layers and weights, however, renders them incomprehensible to humans. While this does not have serious consequences when it comes to playing games, it can have a severe impact when neural networks are applied to safety-critical systems like self-driving cars [3]. Validation and verification procedures are thus needed to provide safety guarantees.

The verification of NNs is a relatively young field. Among the first papers published is the work by Pulina and Tacchella [22], where the authors checked bounds on the output of multilayer perceptrons. Most current publications (e.g., [10,16,26]) focus on proving the adversarial robustness of NNs, meaning that a network assigns the label of a known reference input-point to all points in a

H. Simonis (Ed.): CP 2020, LNCS 12333, pp. 868–884, 2020.
https://doi.org/10.1007/978-3-030-58475-7_50

small region around it. To prove this property, the NNs are often encoded as constraint systems solved with SAT, SMT or MILP solvers. Over the last years, the size of NNs used in practical applications grew rapidly, and today's networks often require huge amounts of memory space. Their execution causes high energy consumption, rendering them impractical for use on mobile or embedded devices. As a consequence, methods have been developed to reduce the size of NNs [14,27]. The question then arises whether the reduced NN is suitable as a replacement for the original one, in the sense that it behaves "sufficiently equivalent" on relevant inputs. Narodytska *et al.* [21] consider two feed-forward NNs equivalent if, for all valid[1] inputs of the input domain, the NNs produce the same output labels. They are able to prove this property for the specialized class of binarized NNs, which allows them to produce a SAT formula representing the equivalence of two NNs. Kumar *et al.* [20] denote this equivalence property by local equivalence over a domain, and use the term equivalence for NNs that give the same output for all inputs. Based on these notions they collapse layers of a given NN while guaranteeing the equivalence to the original NN.

Unfortunately, exact equivalence is hard to fulfill for two NNs, even if they have the same structure and have been generated using the same training data, due to the stochastic nature of the training process. In this paper, we study feed-forward NNs with the ReLU activation function, which is the most used activation function in modern NNs [11]. We present a new relaxed equivalence property for NNs and show, how it—along with existing equivalence properties—can be encoded in MILP. Additionally, we show an encoding for the verification of equivalence, as well as maximizing the size of equivalent regions, when the input domain is restricted to radii around a point in input space. We evaluate our approach using the constraint solver Gurobi [13] and a NN trained on the Optical Recognition of Handwritten Digits dataset [7]. The evaluation of our approach marks the first time, that NN compression methods have been examined by verification methods with respect to generating equivalent NNs.

2 Foundations

We give short introductions into NNs and MILP. Afterwards, we present the encoding of NNs into MILP formulae.

Neural Networks. NNs consist of a number of interconnected units, sometimes called neurons. One single neuron j computes its output y_j as a function of input values $x_0, ..., x_n$ according to $y_j = \sigma(\sum_{i=0}^{n} w_{ij}x_i)$, where σ is called the activation function and x_0 is commonly set to 1, such that w_{0j} encodes the bias of the neuron. The weights w_{ij} can be learned from training data. To enable the NN to capture non-linear functions, the activation function also has to be non-linear. While there are many choices like the *tanh* or sigmoid function, we focus on the *rectified linear unit*: $\text{ReLU}(x) = max(0, x)$, which is the most commonly

[1] Validity just ascertains that inputs are suitably bounded, e.g. to a range of $[0, 255]$ for greyscale pixels.

used activation function in modern NNs [11]. A NN is formed by connecting neurons via directed links, such that the outputs y_k of previous neurons serve as inputs x_i of the current neuron. In this paper, we focus on feed-forward NNs, where the graph formed by these connections is acyclic. The neurons in these NNs can be organized in layers, such that each neuron only uses outputs of the neurons in the layer directly preceding it as its inputs. The first layer—called input layer—is just a place holder for the inputs to be fed into the NN, the subsequent layers are called hidden layers, while the last layer—the output layer—holds the function value computed by the NN. In a regression setting, the output value represents the NN's estimate of the respective latent function value for the given input. For a classification task, however, the output y_i of neuron i represents the probability of the NN's input belonging to class i. To ensure that the resulting distribution over the outputs is normalized, each output neuron i uses the softmax activation function

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad .$$

Mixed Integer Linear Programs. A MILP problem is an optimization problem for a linear objective function under additional linear constraints. Some variables are constrained to be integers, while others range over $\mathbb{R}$.

Definition 1. *A* mixed integer linear programming problem *consists of*

1. *a linear* objective function $f(x_1, ..., x_k) = \sum_{i=1}^{k} c_i x_i$ *over decision variables* x_i *that is to be minimized or maximized,*
2. *a set of linear* constraints $\sum_{i=1}^{k} a_{ij} x_i \bowtie b_j$, $\bowtie \in \{\leq, =, \geq\}$, *where* a_{ij} *and* b_j *are constants,*
3. *and an integrality constraint* $x_i \in \mathbb{Z}$ *for some variables.*

Solving MILPs is in general NP-hard. Algorithms for solving them include branch and bound, cutting planes, or methods based on relaxations.

2.1 Encoding of Neural Networks as MILP

To argue over properties of NNs, we encode them in MILP utilizing the big-M encoding presented in [5]. Our encoding is equal under transformation to the ReLU-encodings of [9].

For a NN to be encoded, we first have to encode a **single neuron**. A neuron j applies a non-linear activation function σ to a linear combination $s_j = \sum_{i=0}^{n} w_{ij} x_i$ of its inputs $x_0, ..., x_n$. Given fixed weights w_{ij}, this equation can be directly encoded in a mixed integer linear program. The non-linear ReLU activation function $y_j = max(0, s_j)$ can be encoded using given bounds $m \leq s_j \leq M$, which can be calculated knowing the bounds for the inputs x_i and weights w_{ij} of the NN. The ReLU function can be encoded using a new zero-one variable $\delta \in \{0, 1\}$, with $\delta = 0$ representing the case $(s_j \leq 0 \wedge y_j = 0)$

and $\delta = 1$ representing $(s_j \geq 0 \wedge y_j = s_j)$. The ReLU function is then encoded by the following set of linear inequalities:

$$\begin{aligned} y_j &\geq 0 & s_j &\geq m(1-\delta) \\ y_j &\geq s_j & y_j - s_j &\leq -m(1-\delta) \\ s_j &\leq M\delta & y_j &\leq M\delta\,. \end{aligned}$$

Given tight bounds $m \leq s_j \leq M$, the encoding can be further simplified [5]. If $M \leq 0$, we can directly encode the ReLU function as $y_j = 0$, and if $m > 0$, we encode the output of the activation function as $y_j = s_j$. These reductions in complexity are particularly valuable as they do not use any integer variables, on whom we might have to branch when solving the resultant mixed integer linear program. Therefore, we employ the approach of [5] to generate tighter bounds by means of interval arithmetic and also solve for bounds on intermediate variables by maximizing or minimizing their values using smaller mixed integer linear programs only covering a low number of layers of the NN at a time.

Based on the encoding of a single neuron, we can encode a **whole NN**. Each input of the NN is represented by a variable x_i with associated bounds $l_i \leq x_i \leq u_i$. These l_i and u_i can be set according to physical limitations or might be obtained from the respective training dataset and can be used for the calculation of subsequent bounds in the encoding. The neurons in the first layer are then encoded according to the previous description. The same procedure is applied to the neurons of the next layers with the outputs y_i of the neurons of the previous layer taking the role of the inputs above, until the output layer is reached. In classification NNs, the neurons in the output layer use the softmax activation function. Due to its exponential functions, an exact encoding in MILP is impossible. However, since the softmax function is monotonic, we are able to reason about the order of the outputs by encoding the linear combination of the input values for the neurons of the output layer in a classification NN.

3 Equivalence Properties

Let $\mathcal{R}$ be a reference and $\mathcal{T}$ a test NN computing functions $f_{\mathcal{R}}, f_{\mathcal{T}} : \mathbb{R}^m \to \mathbb{R}^n$. We further assume that the inputs to these NNs come from the same domain X and that the i-th component of the output vector of $f_{\mathcal{R}}(\mathbf{x})$ and $f_{\mathcal{T}}(\mathbf{x})$ has the same meaning in the encoding of the output neurons. Proving exact equivalence of the test NN $\mathcal{T}$ and the reference NN $\mathcal{R}$ would then mean to ascertain that

$$\forall \mathbf{x} \in X : f_{\mathcal{R}}(\mathbf{x}) = f_{\mathcal{T}}(\mathbf{x})\ . \tag{1}$$

However, the training procedure of NNs is highly non-deterministic and training could be on different datasets, thus leading to differences in the learned weights, even if the NNs share the same number of layers and neurons. It is therefore unlikely for two NNs to fulfill the exact equivalence property stated above. Hence, we need to relax it to obtain a more practical notion of equivalence. In general, this can either be achieved by (1) relaxing the exact equality of the function

values in Eq. 1 through a less strict relation $\simeq$, or (2) restricting the domain of the inputs to the NNs to smaller subsets, for which equality is more likely. The first approach is described below, while we discuss the input restriction in Sect. 4.

3.1 Relaxed Equivalence Properties

The definition of exact equivalence in Eq. 1 can be written as a difference: $\forall \mathbf{x} \in X : f_{\mathcal{R}}(\mathbf{x}) - f_{\mathcal{T}}(\mathbf{x}) = 0$. An easy relaxation would be to consider two functions equivalent, if their difference is at least close to zero within some threshold.

Definition 2 (ε-Equivalence). *We consider $\mathcal{R}$ and $\mathcal{T}$ to be* ε-equivalent *with respect to a norm $\|\cdot\|$, if $\|f_{\mathcal{R}}(\mathbf{x}) - f_{\mathcal{T}}(\mathbf{x})\| \leq \varepsilon$ for all $\mathbf{x} \in X$.*

While this is a valid relaxation in the context of regression NNs, the functions $f_{\mathcal{R}}$ and $f_{\mathcal{T}}$ compute class probability distributions when it comes to classification problems. In most of these cases, one is not interested in the full class probability distribution, but only in the classification result—the class assigned the highest probability by the NN. In that case, we can obtain a relaxed equivalence property by comparing only the classification results.

Definition 3 (One-Hot Equivalence) . *We call $\mathcal{R}$ and $\mathcal{T}$* one-hot equivalent*, if $f_{\mathcal{R}}(\mathbf{x}) = \mathbf{r}$ and $f_{\mathcal{T}}(\mathbf{x}) = \mathbf{t}$ satisfy $\arg\max_i r_i = \arg\max_i t_i$ for all $\mathbf{x} \in X$.*

The name stems from the representation of the true label for each input in the training data as a one-hot vector for classification NNs. We note that this definition is closely related to the property of adversarial robustness [26], however we compare the classification results of two NNs instead of comparing the classification result of one NN with the ground-truth.

The notion of one-hot equivalence can be relaxed even further when we consider not only the most likely class, which is the classification result, but instead take the k most likely classes into account (the definition is motivated by a similar idea in [5]).

Definition 4 (Top-k Equivalence). *A test NN $\mathcal{T}$ is equivalent to a reference NN $\mathcal{R}$, if $f_{\mathcal{R}}(\mathbf{x}) = \mathbf{r}$ and $f_{\mathcal{T}}(\mathbf{x}) = \mathbf{t}$ satisfy*

$$\arg\max_i r_i = j \implies \mathrm{pos}(t_j, \mathbf{t}) \leq k \ ,$$

where $\mathrm{pos}(w_j, \mathbf{w})$ returns i, if w_j is the i-th largest value in vector $\mathbf{w}$, and r_j is the unique maximum component of vector $\mathbf{r}$.

Informally, a testing NN $\mathcal{T}$ is top-k equivalent to a reference NN $\mathcal{R}$, if the classification result of $\mathcal{R}$ is amongst the top k largest results of $\mathcal{T}$. This can be interpreted in a way, such that the NN, even if it differs from the classification result of the original NN, at least only makes sensible errors. One-hot equivalence can also be seen as a special case of top-k-equivalence for $k = 1$.

Note that, while exact equality and one-hot-equality are equivalence relations in the mathematical sense, neither ε-equivalence, nor top-k-equivalence for $k > 1$ meet that criterion, as both are not transitive and the latter additionally is not symmetric.

3.2 Encoding of Equivalence Properties in MILP

In the context of adversarial robustness, properties are often encoded as MILP problems [4,18], a formalism we also employ for our equivalence properties. Searching for an input that maximizes the violation of these equivalence constraints has the advantage that we get information about the extent to which the corresponding NNs are not equivalent. With an encoding as a satisfiability problem, we would only get a single and possibly very small violation. In general, we encode equivalence of two NNs $\mathcal{R}$ and $\mathcal{T}$ as the following mixed integer linear program

$$\max \quad d \tag{2}$$

$$s.t.\ \mathbf{r} = \text{enc}_{\mathcal{R}}(\mathbf{i}) \tag{3}$$

$$\mathbf{t} = \text{enc}_{\mathcal{T}}(\mathbf{i}) \tag{4}$$

$$d = f(\mathbf{r}, \mathbf{t}) \tag{5}$$

where Eqs. 3 and 4 encode a reference NN $\mathcal{R}$ and the testing NN $\mathcal{T}$ on the common inputs $\mathbf{i}$ as described in Sect. 2.1, yielding the respective outputs of the NNs $\mathbf{r}$ and $\mathbf{t}$. As we are dealing with MILP, some of these variables are real numbers, while others are restricted to be integers. Below we are going to discuss the encoding of the function f, which calculates the scalar violation score d for a given equivalence property, for top-k and then for ε-equivalence.

Top-k-Equivalence. We can encode the violation score of the top-k-equivalence $\mathcal{R} \simeq \mathcal{T}$ (or $\mathcal{T}$ is equivalent to $\mathcal{R}$) as a simple difference

$$d = \hat{t}_k - t_j \ , \tag{6}$$

where $\arg\max_i r_i = j$. The variable $\hat{t}_k$ denotes the k-th largest component of $\mathbf{t}$. If $d = \hat{t}_k - t_j \leq 0$, then we have $\hat{t}_k \leq t_j$, meaning that the output of $\mathcal{T}$ corresponding to the classification result of $\mathcal{R}$ is larger or equal to the k-th largest output of $\mathcal{T}$. Therefore t_j would be amongst the k largest outputs of $\mathcal{T}$ and thus satisfy top-k-equivalence. The main difficulty in encoding top-k-equivalence lies in encoding the sorting of the outputs of the NNs according to their activation value. We can encode the calculation of the descendingly sorted vector $\hat{\mathbf{x}} = \Pi\mathbf{x}$ of a NN's output values $\mathbf{x}$ by using a permutation matrix $\Pi = (\pi_{ij})_{i,j=1}^{n}$ similar to [17] and then adding the necessary ordering constraints (Constraint 10):

$$\hat{x}_i = \sum_j \pi_{ij} x_j \ \forall i \leq n \tag{7}$$

$$\sum_i \pi_{ij} = 1 \ \forall j \leq n \tag{8}$$

$$\sum_j \pi_{ij} = 1 \ \forall i \leq n \tag{9}$$

$$\hat{x}_i \geq \hat{x}_{i+1} \ \forall i \leq n-1 \tag{10}$$

$$\pi_{ij} \in \{0,1\} \ \forall i,j \leq n \ , \tag{11}$$

where the Constraints 8 and 9 together with the binary constraint on the π_{ij}, ensuring that each column and each row only contain one 1 and only 0 elsewhere, are sufficient to characterize Π as a permutation matrix. While multiplications of two variables are in general non-linear, we can utilize that the π_{ij} are binary variables, to encode the products $\pi_{ij}x_j$ in the above formulation. Binary multiplications $\delta x = y$, where $\delta \in \{0, 1\}$, can be linearized by encoding the implications $\delta = 0 \rightarrow y = 0$ and $\delta = 1 \rightarrow y = x$ as linear inequalities.

Using the above information, we can retrieve sorted vectors $\hat{\mathbf{r}}, \hat{\mathbf{t}} \in \mathbb{R}^n$ of the outputs of two NNs. To find the component of t_j of $\mathbf{t}$ that corresponds to the largest component of $\mathbf{r}$, one can apply the permutation matrix to calculate $\hat{\mathbf{r}}$ to $\mathbf{t}$ and extract its first component. However, we don't need to generate two full permutation matrices. Realizing that we are only interested in the largest value of $\mathbf{r}$ and the k largest values of $\mathbf{t}$, it is sufficient to encode the first row of the permutation matrix for $\mathbf{r}$ and the first k rows of the permutation matrix for $\mathbf{t}$, thus reducing the number of binary variables. When multiple outputs r_i of NN $\mathcal{R}$ share the highest activation value, valid permutations could be obtained, such that in one of them component r_j and in the other $r_{j'}$ is the top component in $\hat{\mathbf{r}}$. Assume that we compare a reference NN $\mathcal{R}$ to a testing NN $\mathcal{T}$, that assigns the highest activation only to t_j, when given the same input as $\mathcal{R}$. Then, we would use this input as a counterexample to their equivalence, since we maximize the violation of the equivalence property and the solver would chose the permutation of $\mathcal{R}$'s outputs, that assigned $r_{j'}$ as the top component. The classification results of $\mathcal{R}$ and $\mathcal{T}$ however could still be the same. Therefore, we require $\mathcal{R}$ to have a unique highest output activation. Since we are not allowed to use strict inequalities in MILP, we use an $\varepsilon > 0$ to ensure a unique greatest output activation. We then arrive at the final encoding of top-k-equivalence. First, we obtain $\mathcal{R}$'s unique top output $\hat{r}_1$:

$$\hat{r}_1 = \sum\nolimits_i \rho_i r_i \tag{12}$$

$$\hat{r}_1 \geq r_i \ \forall i \leq n \tag{13}$$

$$\rho_i = 0 \rightarrow \hat{r}_1 \geq r_i + \varepsilon \ \forall i \leq n \tag{14}$$

$$\sum\nolimits_i \rho_i = 1 \text{ and } \rho_i \in \{0, 1\} \ \forall i \leq n \ , \tag{15}$$

where $\rho = (\rho_1, ..., \rho_n)^T$ is used just as the first row of a permutation matrix. Then, we can solve for $\mathcal{T}$'s activation t_r for the component of $\mathcal{R}$'s largest output, by applying ρ to the output of $\mathcal{T}$.

$$t_r = \sum\nolimits_i \rho_i t_i \tag{16}$$

The k greatest outputs of $\mathcal{T}$ are computed as follows:

$$\hat{t}_i = \sum\nolimits_j \pi_{ij} t_j \ \forall i \leq k \qquad \hat{t}_i \geq \hat{t}_{i+1} \ \forall i \leq k-1$$

$$z_j = \sum\nolimits_i \pi_{ij} \leq 1 \ \forall j \leq n \qquad z_j = 0 \rightarrow t_j \leq \hat{t}_k \ \forall j \leq n$$

$$\sum\nolimits_j \pi_{ij} = 1 \ \forall i \leq k \qquad \pi_{ij} \in \{0, 1\} \forall i \leq k, j \leq n \ ,$$

where the $(\pi_{ij})_{i,j=(1,1)}^{(k,n)}$ form the first k rows of the permutation matrix for $\mathcal{T}$'s outputs and z_i indicates, whether t_i is amongst $\mathcal{T}$'s k largest outputs. Finally, we can compute the violation of the top-k-equivalence property as the difference

$$d = \hat{t}_k - t_r \ , \tag{17}$$

which is then maximised to find the counterexample resulting in the largest possible violation of the equivalence property.

Interval Arithmetic. We assume that lower and upper bounds are given for the input variables and use existing interval extensions for the sum and multiplication to generate bounds on the linear combinations of inputs. The ReLU function is then applied to these bounds to generate bounds on the output of the neuron. This process is repeated throughout the network. Naively applying this kind of interval arithmetic to the equations defining $\hat{r}_1, t_r$ and the $\hat{t}_i$ respectively would produce large overestimates. In Eq. 12 for example, the upper bound on $\hat{r}_1$ would be the sum instead of the maximum of the upper bounds of the r_i (only one entry is equal to 1 in a row of the permutation matrix). Therefore, we use context groups to compute tighter bounds for these variables. Assume, we are choosing a variable x from a set $X = \{x_1, ..., x_n\}$, where $x_i \in [l_i, u_i]$. Let $\hat{\mathbf{l}}$ and $\hat{\mathbf{u}}$ denote the vectors containing the lower, respectively upper bounds sorted in decreasing order. If we choose x to be the k-th largest variable out of X, we can combine x in a *top-k-group* and assign tighter lower and upper bounds for x according to: $x \in [\hat{l}_k, \hat{u}_k]$.

***ε*-Equivalence.** We encode ε-equivalence and exact equivalence as maximizing

$$d = \|\mathbf{r} - \mathbf{t}\|. \tag{18}$$

The equivalence property is satisfied, if $\max d \leq \varepsilon$ for ε-equivalence. The value of ε has be chosen according to the dataset. The "optimal" value can be determined by incrementally looking at counterexamples for the equivalence and deciding if, from the user-perspective, the outputs are equivalent. For exact equivalence $\varepsilon = 0$ is required. In order to use Eq. (18) in MILP, we need to encode the non-linear $\|\cdot\|$ operator. We restricted ourselves to the Manhattan $\|\cdot\|_1$ and the Chebyshev norm $\|\cdot\|_\infty$ defined as

$$\text{Manhattan: } \|\mathbf{x}\|_1 = \sum\nolimits_i |x_i|, \quad \text{Chebyshev: } \|\mathbf{x}\|_\infty = \max_i |x_i| \ , \tag{19}$$

because they are piecewise linear functions and can thus be encoded in MILP.

Just as we have done earlier, $y = |x|$ can also be expressed as cases $x \leq 0 \wedge y = -x$ and $x \geq 0 \wedge y = x$, that can be encoded as linear inequalities by introducing a binary variable. If the bounds $l_x \leq x \leq u_x$ indicate, that the domain of x contains only positive ($l_x \geq 0$) or only negative ($u_x \leq 0$) values, we can just set $y = x$ or $y = -x$, respectively.

In case of the Manhattan norm, we just sum over the absolute values of the components. The maximum operator used in the Chebyshev norm can be

represented in the same way, as we have done to obtain the output with the highest activation for a NN in Eqs. (12)–(15) in the previous section.

However a unique largest value is not required in this case, so Eq. (14) is not needed in this encoding. We can also use the *top-k-group* we introduced in Sect. refsssec:ia above with a value of $k = 1$ to allow for the calculation of tighter bounds on the result of this maximum operator.

4 Input Restriction

As mentioned in Sect. 3, exact equivalence can also be relaxed by restricting the input domain, for which the equivalence property has to hold. In practice, it is especially useful to restrict the input domain to values, that are covered by the training dataset of the respective NNs. Differences in the output of NNs in the neighborhood of their training samples are far more meaningful than differences in regions, where they would not have been applied anyway. Furthermore, restricting the input values allows for the calculation of tighter bounds.

Below, we give a quick overview of the hierarchical clustering approach of [12], we used for restricting the input space to regions within a radius around cluster-centers of training data. Subsequently, we show MILP encodings for proving equivalence of two NNs for the restricted input regions. We also show, how this process can be modified for maximizing the radius around a point, such that the violation of a chosen equivalence property is smaller than a specified threshold.

4.1 Hierarchical Clustering

The hierarchical clustering method of [12] starts by clustering a set of labelled data-points $\{(x_i, y_i)\}_{i=1}^{n}$, with k distinct labels into k clusters. If a cluster contains input points of different labels, the method is recursively applied to that cluster, until all clusters only contain inputs of a common label. Every cluster is then characterized by its cluster center and its radius, which denotes the maximum distance from the cluster center to its input points. The underlying assumption for this clustering is, that all points, not just the training data-points, in a dense cluster should be assigned the same label. As points close to a cluster boundary might lie on a real decision boundary between two classes, [12] set the radius r_c of a cluster to the average distance of the input-points to the cluster center. Note, that the above assumption only holds for clusters of high density n/r_c, where n is the number of training data-points in the respective cluster.

4.2 Encoding of Clusters

As each cluster is characterized by its center $\mathbf{c}$ and radius r_c, one can place a norm restriction $\|\mathbf{i} - \mathbf{c}\| \leq r_c$ on the vector of inputs $\mathbf{i}$, to reduce the domain of the verification procedure to only inputs from that cluster. Thus, we can encode the input restriction by extending the encoding of NN equivalence given in Sect. 3.2 through adding the norm restriction to Eqs. (3)–(5).

Again, we restrict ourselves to encoding the Manhattan and Chebyshev norms (Eq. 19). The encoding of the Chebyshev norm for input restriction is less complicated than its encoding needed for ε-equivalence, as we do not need to actually calculate the value of the norm, but just ensure, that all input values are within a set distance of the cluster center. Which leads to a box constraint on the input variables $\mathbf{i}$. Therefore the lower and upper bounds l_j and u_j of variable i_j can be updated to $l'_j = \max(c_j - r_c, l_j)$ and $u'_j = \min(c_j + r_c, u_j)$ respectively. The Manhattan norm $\|\cdot\|_1$ however has to be encoded just as in Sect. 3.2. Nonetheless, we can use the fact that $\|\mathbf{x}\|_1 \geq \|\mathbf{x}\|_\infty \ \forall \mathbf{x}$, to achieve faster tightening of the variable bounds by adding the bounds calculated in the encoding of the Chebyshev norm.

4.3 Searching for a Maximal Radius

In order to find the largest radius around a center $\mathbf{c}$ in input-space, where NNs $\mathcal{R}$ and $\mathcal{T}$ are equivalent, it is not possible to just use the equivalence encoding adding the presented norm and set r_c to be maximized. Since the solver finds an assignment to the input variables $\mathbf{i}$, such that the objective, in that case the radius, is maximized, the NNs are still equivalent and $\|\mathbf{i} - \mathbf{c}\| \leq r_c$. In that situation the solver could choose $\mathbf{i} = \mathbf{c}$. Therefore, the equivalence constraint would be met, if the NNs are equivalent on the center, and the maximum of the radius would be arbitrarily large. Hence, we search for the smallest radius r_v, for which a counterexample to the equivalence of $\mathcal{T}$ and $\mathcal{R}$ can be found. This optimization problem is similar to the one proposed in [25] for finding adversarial examples for a single NN close to training inputs, which they approximately solve using gradient based methods. Our MILP formulation reads:

$$\begin{aligned}
\min \quad & r_v && (20)\\
s.t. \ & \mathbf{r} = \mathrm{enc}_{\mathcal{R}}(\mathbf{i}) && (21)\\
& \mathbf{t} = \mathrm{enc}_{\mathcal{T}}(\mathbf{i}) && (22)\\
& f(\mathbf{r}, \mathbf{t}) \geq \varepsilon_v && (23)\\
& \|\mathbf{i} - \mathbf{c}\| \leq r_v \, . && (24)
\end{aligned}$$

Note that we used a small threshold value of $\varepsilon_v > 0$ for the violation.

If r^* is the optimal solution for the above minimization problem, the two NNs are not equivalent for radii $r' \geq r*$, as the solver could generate a counterexample for r^*. But we cannot guarantee that the NNs are equivalent for $r' < r^*$ because of the use of the threshold value. For small values of ε_v, the NNs are likely to be equivalent for radii $r' \leq r^* - \varepsilon_r$ for small ε_r. This can then be verified using the methods for fixed radii described in Sect. 4.2. If verification tasks for fixed radii have been carried out beforehand, the largest (smallest) radius, for which the NNs were (not) equivalent can be used as a lower (upper) bound on r_v in the radius-minimization problem.

5 Application: Neural Network Compression

The huge number of parameters in modern NNs lead to large amounts of memory – AlexNet [19], for example uses 200 MB of disk space. Hence, it is desirable to reduce the number of parameters of a NN, without compromising its performance on the task it is designed to solve. Our approach for verifying equivalence properties of NNs in combination with the presented input restrictions could be used to verify the equivalence, or at least quantify the similarity, of the original NN and the smaller NN, which is the result of the reduction in parameters. This reduction is usually done by pruning unimportant weights - setting their value to zero - essentially removing insignificant connections between neurons. In the context of *magnitude based pruning*, weights of small absolute value are considered negligible [24]. The NN may be retrained after pruning, to correct for the missing connections [24]. During this step and the following iterations of pruning and retraining, the weights of the pruned connections are fixed at zero. Another way to reduce the number of parameters, applicable only to classification tasks, is to directly train a smaller NN on the outputs of a well performing large NN, which is called *student-teacher training* [1,15].

6 Evaluation

We have implemented our approach in Python 3 and are able to automatically generate MILP encodings together with input restrictions. Our program is able to read in NNs exported by Keras [6] and uses version 8.1.1 of Gurobi [13] to solve the generated instances. We used this implementation to analyze the equivalence between compressed and original NNs, as well as between compressed NNs.

Neural Networks. Our original NN consists of an input layer, hidden layers of 32 and 16 ReLU units and an output layer of size 10 (denoted: 32-16-10). It was trained using the *Optical Recognition of Handwritten Digits Dataset* [7].

The dataset consists of 8×8 pixel labeled images of handwritten digits, giving us 64 input variables, whose values are in the closed interval $[0, 16]$, which can be used as naive bounds on the input variables. We implemented bounds tightening and interval arithmetic to increase scalability, yet the applicability for larger networks as well as further optimizations or combinations with approximated approaches are part of future work.

Reduced size NNs were obtained by pruning and retraining the original NN in 10% increments. Additionally, NNs with less ReLU units were learned using student-teacher training. All NNs were trained using the Keras machine learning framework [6]. The achieved accuracy values on the training and testing datasets are shown in Fig. 1 for the pruned NNs, as well as for different structures of student NNs. After the training process, we removed the softmax activation function in the output layer of the NNs to allow for their encoding in MILP.

Experiments. Verification tasks for top-k-equivalence were conducted with and without input restricted around the cluster centers shown in Fig. 2. These clusters were the five most dense clusters obtained by hierarchical clustering, when

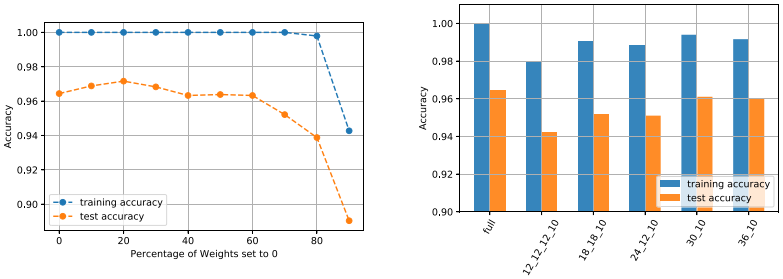


Fig. 1. Accuracy values for original NN, low magnitude weight pruning NNs (left) NNs trained by student-teacher training (right).

applied to the *Optical Recognition of Handwritten Digits* dataset using Manhattan distance. Each cluster contains between 66 and 91 training images.

Experiments were conducted for $k \in \{1, 2, 3\}$ without input restriction and for fixed radii, while the experiments for searching maximal equivalence radii were conducted for $k \in \{1, 2\}$. Due to space limitations, we present the experimental results for searching maximal radii. The tool and instructions how to produce the results can be found under https://github.com/phK3/NNEquivalence.

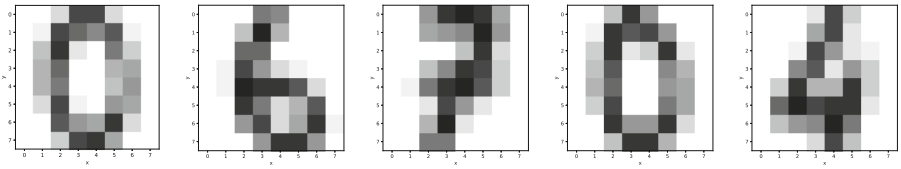


Fig. 2. The cluster-centers of the five most dense clusters obtained from hierarchical clustering of the *Optical Recognition of Handwritten Digits* dataset.

Before the problem encoding was passed to Gurobi, bounds tightening was performed using interval arithmetic and optimization of two-layer subproblems for each linear combination of ReLU inputs. Each subproblem solution-process was stopped after a maximum of 20 s. All experiments were conducted on a computer with an Intel Core i5-3317U 1.70 GHz processor, which has 2 physical and 4 logical cores, and 8 GB of RAM running an x64 version of Windows 10.

6.1 Equivalent Neural Networks

We want to verify the equivalence of compressed NNs, by calculating the maximal equivalence radii for the chosen input clusters. Figure 3 shows the development of maximal radii for top-1 equivalence for both compression methods. The individual radius depends on training data and is reflected by the total number for the maximal radius for all reduction methods.

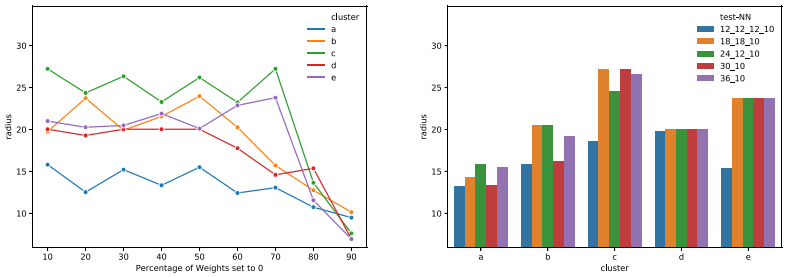


Fig. 3. Maximal equivalence radii for top-1-equivalence to the full NN for weight pruned NNs (left) and student-teacher trained NNs (right)

When pruning a larger percentage of parameters, the equivalence radius fluctuates around a constant level for each cluster up until the 50% reduced NN. If too many parameters were set to zero, the pruned NNs are no longer equivalent to the original NN and the equivalence radius deteriorates, as can be seen for the 80% and 90% pruned NNs. For the 60% and 70% pruned NNs however, one notices, that the equivalence radii for clusters b, d drop as expected. Radii for the clusters a, c and e, on the other hand, either stay on the same level or even increase. An explanation for the observed behavior could be, that about 50% of the original NN's parameters are sufficient to capture the underlying knowledge in the data for the tested clusters. If more parameters are pruned, the reduced NNs focus on the obviously classifiable clusters to still achieve a low training error. When the NNs are pruned even further, their capacity is clearly too low.

Examining the student-teacher trained NNs, we notice, that the equivalence radii not only depend on the number of ReLU nodes, but also the structure of the NNs. While the 12-12-12-10 student has more ReLU units than the 30-10 student and the same number as all other student NNs, it exhibits sometimes significantly smaller equivalence radii on all clusters. Among the student NNs, it also exhibited the lowest accuracy on the training and testing datasets, indicating that 12 neurons per layer are not best suited for this classification task. The 18-18-10 student and the 36-10 student show however, that good accuracy and large equivalence radii can be obtained for this number of ReLU nodes.

Comparing the different compression algorithms for top-1-equivalence, we notice, that most student NNs achieve similar radii as the up to 50% pruned NNs on clusters a, b, c and d and radii as large as that of the 70% pruned NN on cluster e. For the 12-12-12-10 student on clusters b and c and additionally the 30-10 student on cluster b, significantly smaller radii indicate a lack of capacity for the student NNs, although this effect is less severe than for the 70%, respectively 80% pruned NNs.

Figure 4 represents the same data for the top-2 equivalence. The verification of top-2 equivalence is harder, thus our approach only returns upper (dotted lines) and lower (normal lines) for the given timeout. In general, the maximal equivalence radii are, as expected, larger then for the top-1 equivalence. This

indicates that the NNs still assign large probabilities to the correct classification result for the cluster regions. It is also possible to observe, that for example the 12-12-12-10 student network does not lag as far behind the other student NNs as before, indicating, that it at least captured a rough understanding of the data.

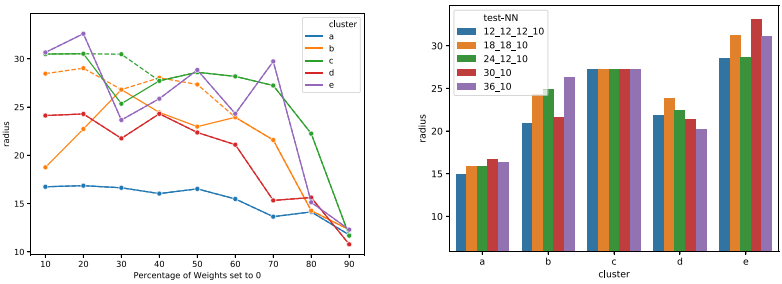


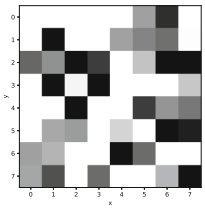
Fig. 4. Maximal equivalence radii for top-2-equivalence to the full NN for weight pruned NNs (left) and student-teacher trained NNs (right)

6.2 Remarks

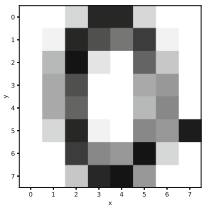
The runtime of the verification procedure depends on the complexity of the MILP encoding, where the number of integer variables seemed to have the largest effect. For our experiments runtime fluctuated between a minute and 30 min for top-1 equivalence. For top-2 equivalence, we set a timeout of 3 h. Verification of equivalence for student-teacher trained NNs with fewer ReLU nodes was in most cases faster, than for the pruned NNs, as fewer integer variables had to be introduced. Only considering pruned NNs, sparser NNs proved to be verified faster than their less sparse counterparts. Overall, verification of equivalence for small fixed radii is faster, than for larger radii, as tighter bounds for all variables in the encoding can be obtained via bounds tightening. For very large radii, however, some NNs seem to be obviously not equivalent and large counterexamples are quickly found by the solver. In the extreme case without input restrictions, counterexamples to equivalence were all found within a minute.

The presented approach is able to search for a maximal radius for which NNs are equivalent and returns an input at the edge of the radius for which the networks are not. We denote this input as a counterexample, which can be analyzed by a potential user. He then has to decide, whether the counterexample should be classified as an valid input. If it is valid, the maximal radius is too small and the NNs are not equivalent, otherwise the NNs are equivalent w.r.t. the cluster. Three kinds of counterexample are shown in Fig. 5. The leftmost picture shows an input picture for unrestricted input. This kind of counterexample is negligible in practice and should not be seen as valid input. It demonstrates the necessity for input restrictions when verifying NNs. The counterexample in

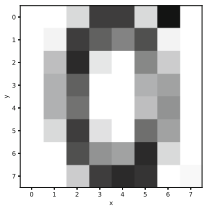
the middle shows a picture of a (in our opinion) zero which is misclassified by the 20% pruned NN. Such a result could indicate that the pruned NN does not fit the wanted equivalence criterion. The left picture on the other hand, shows a digit that is misclassified by the original NN, which could indicate that the original NN should be retrained with the given counterexample.



full NN: 5
50% pruned NN: 6



full NN: 0
20% pruned NN: 6



full NN: 5
50% pruned NN: 0

Fig. 5. Counterexamples to one hot equivalence: Without input restriction (left), input within a 13.6 (middle) and 27.2 (right) radius in Manhattan norm around the center of cluster (a).

7 Conclusion

With top-k-equivalence, we presented a novel relaxed equivalence property for NNs and showed, how it, as well as pre-existing notions of equivalence can be encoded in MILP for NNs using the ReLU activation function. Despite the relaxation, NNs rarely meet these equivalence properties, when the whole input space is considered, as their training only encourages them to agree on areas close to training data. Therefore we used the restriction of inputs to regions around clusters of training data, as proposed in [12]. We then developed MILP formulations, of equivalence for inputs within a fixed radius around obtained cluster-centers, as well as maximizing that radius, such that the NNs are still equivalent. Experiments with a NN trained on the *Optical Recognition of Handwritten Digits Dataset* [7] and its downsized counterparts obtained by student-teacher training or weight pruning showed the validity of our approach. As compression algorithms for NNs are typically only evaluated empirically by measuring the performance of the resultant NNs on a test dataset, this marks the first verification based examination of such methods. The notion of verified equivalence in a given cluster radius can be used to give guarantees for smaller networks. Furthermore, it can be utilized for finding meaningful counterexamples for the pruned and original network which can than be used for further training.

Our approach could also be applied, when numerous verification tasks have to be carried out for a large NN. In this case, a smaller NN could be obtained by compression algorithms. We could then prove its equivalence to the large NN

within the input space of interest and subsequently perform the initial verification tasks on the smaller NN, requiring less computation time. For this scenario further improvements in scalability are needed. A first step could be using dedicated solvers for piecewise linear NNs like *Reluplex* [18] or the assistance of approximate methods [8].

References

1. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 27, pp. 2654–2662. Curran Associates, Inc. (2014). http://papers.nips.cc/paper/5484-do-deep-nets-really-need-to-be-deep.pdf
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate (2014)
3. Bojarski, M., et al.: End to end learning for self-driving cars (2016)
4. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: A unified view of piecewise linear neural network verification (2017)
5. Cheng, C.H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) Automated Technology for Verification and Analysis ATVA 2017. LNCS, vol. 10482, pp. 251–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_18
6. Chollet, F., et al.: Keras (2015). https://keras.io
7. Dua, D., Graff, C.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
8. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: UAI (2018)
9. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) Automated Technology for Verification and Analysis, pp. 269–286. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
10. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints **23**(3), 296–309 (2018). https://doi.org/10.1007/s10601-018-9285-6
11. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
12. Gopinath, D., Katz, G., Pasareanu, C.S., Barrett, C.: DeepSafe: a data-driven approach for checking adversarial robustness in neural networks (2017)
13. Gurobi Optimization LLC: Gurobi. https://www.gurobi.com/
14. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding (2015)
15. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS Deep Learning and Representation Learning Workshop (2015). http://arxiv.org/abs/1503.02531
16. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification, pp. 3–29. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
17. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **28**(8), 1200–1214 (2006)

18. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386
20. Kumar, A., Serra, T., Ramalingam, S.: Equivalent and approximate transformations of deep neural networks (2019)
21. Narodytska, N., Kasiviswanathan, S., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
22. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification, pp. 243–257. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_24
23. Silver, D., et al.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362(6419), 1140–1144 (2018). https://science.sciencemag.org/content/362/6419/1140
24. Ström, N.: Phoneme probability estimation with dynamic sparsely connected artificial neural networks. Free Speech J. **5**, 1–41 (1997)
25. Szegedy, C., et al.: Intriguing properties of neural networks (2013)
26. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming (2017)
27. Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning (2016)