ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis Econometrics and Operations Research

Incorporating nonlinear distance metrics in mixed integer linear programming formulations of deep neural networks

Name student: Jia Hui Zhu

Student ID number: 498700

Supervisor: B.T.C. van Rossum

Second assessor: Dr. T.A.B. Dollevoet

Date final version: July 4, 2021

Abstract

Deep Neural Networks (DNNs) are biologically inspired computing systems that are capable of learning and extracting patterns from data without the necessity of explicit programming. DNNs can be misled by *adversarial examples*, which are slightly perturbed inputs that strongly resemble the original input, but are classified differently. Previous literature has created adversarial examples with expensive local search algorithms, which minimize the necessary perturbations measured by some nonlinear distance metric. We explore the feasibility of creating adversarial examples with nonlinear distance metrics through a 0-1 MILP model that represents a DNN in the context of handwritten digit recognition. These distance metric are incorporated in the model with the assistance of piecewise linear approximations and are evaluated based on the classification accuracies and the robustness of the adversarial trained network. We find that this procedure for the creation of adversarial examples is practical with an effective bound tightening technique for smaller sized DNNs. Performing adversarial training with the created adversarial examples furthermore achieves more accurate and more robust neural networks.

Contents

1 Introduction

Deep Neural Networks (DNNs) are computing paradigms that are inspired by the biological neural networks in animal brains. DNNs consist of layers made up of internal units called *neurons*. These neurons process the output of the previous layer in a nonlinear way and return the output to the neurons in the next layer. DNNs learn and extract patterns from data without the need of explicit programming. Applications include image classification, speech recognition, playing games and the processing of languages (Carlini & Wagner, 2017). Given a fixed set of parameters, DNNs can be modeled as a 0-1 Mixed Integer Linear Programming (0-1 MILP) model. Whilst this 0-1 MILP model is unsuited for training, it can process inputs in the same manner as the original neural network and produce identical outputs.

Neural networks are, however, susceptible to attack: Szegedy et al. (2013) found that neural networks can be misled by *adversarial examples*. These are slightly modified inputs that are very similar to the original input, but are classified differently. Adversarial examples are typically created by minimizing the deviation from the original input measured by some distance metric. Each metric aims to measure a different distance, such as the number of changed inputs or the maximum changed input compared to the original input. As a result, adversarial examples created with different distance metrics can have different effects on neural networks. Previous works, including the research done by Szegedy et al. (2013) and Goodfellow et al. (2014), have relied on expensive local search algorithms to tackle the nonlinear problem of creating large quantities of adversarial examples. We will explore the flexibility of the creation of adversarial examples with nonlinear distance metrics in a 0-1 MILP model setting. We will incorporate several nonlinear distance metrics in the 0-1 MILP model by performing linearizations and piecewise linear approximations. Based on the computation times and the quality of the created adversarial examples, conclusions will be drawn on the feasibility of creating adversarial examples through mixed integer linear programming.

This work closely follows the research done by Fischetti and Jo (2018). A DNN is considered for the recognition of handwritten digits. Input figures are obtained from the MNIST data set of handwritten digits by LeCun and Cortes (2010). The DNN is modeled as a 0-1 MILP model, with continuous variables representing the output of each neuron. This model is used for finding input examples that maximize the activation of each neuron and for the creation of adversarial examples. Fischetti and Jo (2018) additionally propose a bound tightening procedure to reduce the running times of the 0-1 MILP model. We will extend the model to fit the L_0, L_2 and L_∞ distances metrics for the creation of adversarial examples. Due to the quadratic nature of the L_2 distance metric, we will approximate this with the λ-formulation discussed by Bisschop (2006). The adversarial examples will be used for adversarial training in line with Szegedy et al. (2013). The newly trained network will be tested for its accuracies on the original MNIST data and on test data containing adversarial examples, and will be evaluated based on the required distances for creating new adversarial examples.

We find that our results are in line with Fischetti and Jo (2018) in the context of feature visualization and the creation of adversarial examples with the L_1 distance metric. The model can be solved in a matter of seconds for smaller DNNs. Solving larger and more realistic DNNs becomes troublesome,

leading to computation times of over an hour for individual instances. The bound tightening procedure succeeds in significantly reducing solution times. We find similar results for the nonlinear distance metrics, being able to smoothly solve small DNNs but encountering troubles sooner than the L_1 distance as the size of the DNN increases. Using the L_0 distance leaves a large portion of instances unsolved and does not suffice in making the DNN more robust. The piecewise linear approximation of the L_2 distance is shown to be of high quality, giving an optimality gap of 1.38% with 17 breakpoints. Overall, the L_∞ distance provides the highest accuracies and the most robust network without excessively increasing the computation times.

Our main contributions include the incorporation of nonlinear distance metrics in a 0-1 MILP model formulation of a DNN through a piecewise linear approximation. We conduct a systematic comparison of these distance metrics in the context of creating adversarial examples for handwritten digits on the basis of the classification accuracies and the robustness of the adversarial trained network. We show that this method of creating adversarial examples is practical for smaller DNNs and that the resulting adversarial examples can succeed in making the network more accurate and robust.

The remainder of the paper is structured as follows. We first provide a literature review in Section 2. Section 3 gives the 0-1 MILP model formulation for the DNN and discusses its applications. Section 4 contains details about the creation of the adversarial examples with the nonlinear distance metrics and the adversarial training procedure. In Section 5 we will present computational results and evaluate the performance of the 0-1 MILP model in the creation of adversarial examples. Finally, Section 6 concludes the paper and discusses possible future research.

2 Literature review

Szegedy et al. (2013) were the first to notice the vulnerability of neural networks to adversarial examples. They create such malicious inputs by maximizing the prediction error of the network, using a box constrained L-BFGS. The adversarial examples are found to be relatively robust and can cause different networks with a different number of layers, different activation functions and different training data to wrongly classify the same inputs. They furthermore introduce the first adversarial training procedure: by inserting adversarial examples with the correct label in the training data, the models were found to become somewhat resistant to adversarial attacks. Goodfellow et al. (2014) extend this research by establishing that the vulnerability of neural networks to adversarial examples is caused by their linear nature. This discovery allows them to develop the *fast gradient sign method*, which linearizes the loss function of the the neural network and uses its gradient to efficiently compute adversarial examples.

Cheng et al. (2017) establish properties about the resilience of Artificial Neural Networks (ANNs) to noisy and manipulated inputs. These properties are defined as the maximum amount of input perturbation that the ANNs tolerate. The calculation of such perturbation bounds reduces to solving Mixed Integer Programming (MIP) problems. They propose specialized encoding heuristics and use parallelization to reduce the running times of MIP solvers.

To counter the effects of adversarial inputs, Katz et al. (2017) propose *Reluplex*, an extension of the

simplex algorithm that is used for solving linear programming problems, to verify properties of DNNs containing *rectified linear units* (ReLUs). Besides allowing variables to temporarily take on values outside their boundaries, Reluplex also permits variables to violate ReLUs. The algorithm iteratively considers such variables and corrects them with pivot and updating procedures.

The paper by Tramèr et al. (2017) explores a fast single-step method that optimizes a linear approximation of the loss function of neural networks. This method is suitable for producing adversarial examples on a large scale. They additionally introduce *Ensemble Adversarial Training* to increase the strength of adversarial training, which augments the adversarial training data with additional adversarial examples obtained from other pre-trained neural networks. Qin et al. (2019) extend this research and find that *Method-Based Ensemble Adversarial Training*, a procedure that leverages adversarial examples generated by different attacking methods, is beneficial in harnessing neural networks against adversarial attacks.

The research done by Papernot et al. (2016) introduces a technique called *defensive distillation* to reduce the impact of adversarial inputs on DNNs. Defensive distillation makes DNNs more robust by extracting information about training points as a probability vector and repeatedly feeding these back to the DNN during training. Carlini and Wagner (2017) look at the creation of adversarial examples using the L_0, L_2 and L_∞ distance metrics. By introducing three new attack algorithms, they find that defensive distillation does not effectively safeguard neural networks against adversarial examples. In particular, research that addresses defenses against adversarial inputs should be sufficiently robust against the L_2 distance metric.

A variety of works have researched the link between DNNs and their associated MILP representations. For the purpose of verifying neural networks, Tjeng et al. (2017) formulate an MIP model to measure their vulnerability to adversarial examples. Through tight formulations for nonlinearities, they are two to three orders of magnitude faster at evaluating the robustness of neural networks to adversarial distortions than Reluplex. Their methods work on neural networks containing hundreds of thousands of neurons and allows them to determine the exact adversarial accuracy of an MNIST classifier.

Grimstad and Andersson (2019) expand the use of ReLU networks to surrogate models in Mixed-Integer Linear Programming (MILP) problems. The MILP formulation bears resemblance to the formulation proposed by Fischetti and Jo (2018), with binary variables representing the activation of ReLU units and *big-M* constraints. Grimstad and Andersson (2019) further explore bound tightening techniques based on linear relaxations and interval arithmetic.

Strong MIP formulations for high dimensional neural networks containing ReLU units and max pooling functions are presented by Anderson et al. (2020). These formulations correspond to trained neural networks and serve several purposes, such as verifying the robustness of an image classification network to adversarial inputs or solving decision problems in the field of machine learning models. They develop strictly stronger formulations than previous approaches from the literature for the aforementioned nonlinear operators, which substantially improve the computation times of verification tasks for image classification.

For a given neural network and a set of inputs, Dutta et al. (2018) aim to compute an over-

approximation of the set of possible outputs. They present an efficient range estimation algorithm based on the alternation between a global combinatorial search that makes use of MILP formulations for the nonlinear activation functions of neural networks and a local optimization procedure that repeatedly searches for the local optimum of the function represented by the neural network. By effectively removing suboptimal nodes, their approach is able to successfully reduce the size of the combinatorial search over possible combinations of active neurons in the neural network.

Our work closely relates to the research done by Fischetti and Jo (2018). They propose a 0-1 MILP model formulation for DNNs in the context of recognizing handwritten digits. We will focus on the creation of adversarial examples with this model, incorporating the nonlinear distance metrics employed by Carlini and Wagner (2017) through piecewise linear approximations. The generated adversarial examples will be used for adversarial training and the robustness of the resulting model will be evaluated, in the spirit of Szegedy et al. (2013).

3 The 0-1 MILP model

Deep Neural Networks (DNNs) are biologically inspired computing systems that allow computers to learn from observational data. A DNN contains a set of layers, which are made up of internal units called *neurons*. The first and last layer of a DNN correspond to the input and output layer, respectively, and the designs of these layers depend on the problem at hand. The neurons in the remaining layers, called the *hidden* layers, take the output of the neurons in the previous layer, process it with the aid of a nonlinear operator, and return the corresponding *activation* value.

DNNs can be modeled as a 0-1 Mixed Integer Linear Programming (0-1 MILP) model, with binary variables representing the activation of the nonlinear operators. Using a fixed set of parameters, the model can be utilized to produce outputs identical to those the original DNN would grant, given a set of inputs. In Section 3.1 we will introduce the formulation of the 0-1 MILP model. Section 3.2 looks at several manners in which the model can be adapted to suit various applications such as feature visualization and the creation of adversarial examples.

3.1 Model formulation

We will closely follow the methodology proposed by Fischetti and Jo (2018). We assume we have a DNN consisting of $K+1$ layers, labeled from 0 to K. The first layer, layer 0, contains the inputs of the DNN, and the last layer, layer K, returns the corresponding outputs. Each layer $k \in \{0, 1, \ldots, K\}$ contains n_k neurons, labeled from 1 to n_k. Let $UNIT(j, k)$ denote the j-th neuron of layer k, and let $x^k \in \mathbb{R}^{n_k}$ be the output vector of the k-th layer. The output of $UNIT(j, k)$ is then denoted by x_j^k. For the input and output layer, x_j^0 corresponds to the j-th input value and x_j^K corresponds to the j-th output value of the DNN. The general output formula for $UNIT(j, k)$ is as follows:

$$x^k = \sigma(W^{k-1}x^{k-1} + b^{k-1}), \tag{1}$$

where $\sigma(\cdot)$ corresponds to a nonlinear function and where W^{k-1} and b^{k-1} are matrices of weights and biases respectively. Weights can be seen as the strength of the connections. An input neuron with a higher weight has a greater impact on the activation of the neuron it is linked to than an input neuron with a lower weight, and vice versa. Biases shift the output of a neuron. Finding suitable values for these two parameters is typically done through training, in which training data is repeatedly processed by the DNN and the weights and biases are adjusted based on the difference between the received output and the expected output.

We assume that each neuron uses a *rectified linear unit*, or ReLU, to compute its output. The output of layer k can then be calculated as

$$x^k = ReLU(W^{k-1}x^{k-1} + b^{k-1}), \;\; k = 1, \ldots, K, \tag{2}$$

where for a real vector y, $ReLU(y) := \max\{0, y\}$. Whilst the weight and bias matrices can contain negative elements, each output vector x^k is nonnegative. To linearly model the ReLU function, we will look at the general scalar equation

$$x = ReLU(w^T y + b). \tag{3}$$

We use the linear conditions

$$w^T y + b = x - s, \;\; x \geq 0, \;\; s \geq 0, \tag{4}$$

to separate the positive and negative part of the ReLU input. To make sure that at least one of the two variables x and s is equal to 0, we introduce the binary activation variable z with the following constraints:

$$\left.\begin{aligned} z = 1 \rightarrow x \leq 0 \\ z = 0 \rightarrow s \leq 0 \\ z \in \{0, 1\} \end{aligned}\right\} \tag{5}$$

Such indicator constraints are often accepted in this form by modern MILP solvers and are internally converted into *big-M* constraints of the form $x \leq M^+(1-z)$ and $s \leq M^- z$, assuming that M^+, $M^- \in \mathbb{R}_+$ can be computed such that $-M^- \leq w^T y + b \leq M^+$. We define the binary activation variables z_j^k for each $UNIT(j, k)$. The model formulation is then as follows:

$$min \sum_{k=0}^{K} \sum_{j=1}^{n_k} c_j^k x_j^k + \sum_{k=1}^{K} \sum_{j=1}^{n_k} \gamma_j^k z_j^k \tag{6}$$

$$s.t. \sum_{i=1}^{n_{k-1}} w_{ij}^{k-1} x_i^{k-1} + b_j^{k-1} = x_j^k - s_j^k \qquad k = 1, ..., K, \;\; j = 1, ..., n_k \tag{7}$$

$$z_j^k = 1 \rightarrow x_j^k \leq 0 \qquad k = 1, ..., K, \;\; j = 1, ..., n_k \tag{8}$$

$$z_j^k = 0 \rightarrow s_j^k \leq 0 \qquad k = 1, ..., K, \;\; j = 1, ..., n_k \tag{9}$$

$$lb_j^0 \leq x_j^0 \leq ub_j^0 \qquad j = 1, ..., n_0 \tag{10}$$

$$lb_j^k \leq x_j^k \leq ub_j^k \qquad k = 1, ..., K, \;\; j = 1, ..., n_k \tag{11}$$

$$\overline{lb_j^k} \leq s_j^k \leq \overline{ub_j^k} \qquad k = 1, ..., K,\ j = 1, ..., n_k \tag{12}$$

$$x_j^k, s_j^k \geq 0 \qquad k = 1, ..., K,\ j = 1, ..., n_k \tag{13}$$

$$z_j^k \in \{0, 1\} \qquad k = 1, ..., K,\ j = 1, ..., n_k \tag{14}$$

We assume that the cost parameters c_j^k and γ_j^k, as well as all weights w_{ij}^{k-1} and biases b_j^{k-1} are given and fixed. Expression (6) is the objective function that minimizes a general cost function for the neuron activations. Constraints (7)-(9) define the ReLU outputs using the approach described previously. Constraints (10)-(14) describe lower and upper bounds on the decision variables. The bounds for the input layer $k = 0$ in (10) depend on the problem at hand. For the layers $k \geq 1$, we have $lb_j^k = \overline{lb_j^k} = 0$ and $ub_j^k, \overline{ub_j^k} \in \mathbb{R}_+ \cup \{+\infty\}$.

Belotti et al. (2016) discuss the necessity of bound tightening surrounding MILPs which contain indicator constraints. Reasonable upper bounds are automatically defined by modern MILP solvers for the continuous variables present in the indicator constraints. These upper bounds can be inaccurate, however, leading to weak and inefficient continuous relaxations during the solving of the model. Fischetti and Jo (2018) propose the following method to create bounds for constraints (11) and (12): iterate through each neuron by increasing layers $k = 1, \ldots, K$. For each $UNIT(j, k)$, all constraints and variables relating to all other units in the current and next layers are removed from the model, and the remaining model is optimized twice: once to maximize x_j^k and once to maximize s_j^k. The resulting optimal values or bounds that the MILP returns after a short time limit can be used as tight bounds for x_j^k and s_j^k. The computed bounds in one iteration can be used in each subsequent iteration due to the acyclic nature of the DNN.

3.2 Applications

The previously described DNN and MILP model can be used in the setting of recognizing hand-written digits in a 28×28 input figure, leading to a total of 784 inputs. Each input corresponds to a pixel in the figure that is normalized to a gray level in $[0, 1]$, where 0 means that the pixel is black and 1 means that the pixel is white. Since we assume that all weights and biases are given, the MILP model can not be used for training the DNN. Instead, the weights and biases are obtained by training simple DNNs containing a small number of hidden layers with ReLUs and an output layer with 10 units to classify each digit through Stochastic Gradient Descent. The figures are obtained from the publicly available MNIST data set by LeCun and Cortes (2010).

Fischetti and Jo (2018) use the MILP model for two main applications. The first is *feature visualization*: to visualize what a $UNIT(j, k)$ computes, input examples are found that maximize the objective x_j^k, representing the activation of $UNIT(j, k)$. The second application is the slight modification of DNN inputs to produce misclassified outputs. Such inputs are called *adversarial examples*. Translating this to our setting, let $\tilde{x}^0$ be an input figure that the DNN correctly labels as a digit $\tilde{d}$. The goal is to create a similar input figure x^0 that the DNN wrongly labels as $d \neq \tilde{d}$. We assume that the structure and the parameters of the DNN are known and that each pixel in the input figure can be changed. Fischetti and Jo (2018) furthermore impose that the wrong digit d is obtained by setting $d = (\tilde{d} + 5) \bmod 10$. This is

achieved by adding the constraints

$$x_{d+1}^K \geq 1.2\ x_{j+1}^K,\ \ j \in \{0,\ldots,9\} \setminus \{d\}, \tag{15}$$

to the MILP model, which say that the activation of the wrong digit d is at least 20% larger than every other activation. To minimize the needed modifications to the input figure, the objective to be minimized is then $\sum_{j=1}^{n_0} d_j$, with the continuous variable d_j representing the L_1-norm distance between x_j^0 and $\tilde{x}_j^0$. These variables must satisfy the following constraints to be added to the model:

$$-d_j \leq x_j^0 - \tilde{x}_j^0 \leq d_j,\ d_j \geq 0,\ \ j = 1,\ldots,n_0. \tag{16}$$

4 Distance norms and adversarial training

The adversarial examples in Fischetti and Jo (2018) are generated with the aim of minimizing the L_1-norm distance between the input figure $\tilde{x}^0$ and the perturbed figure x^0. Other research, including the paper by Carlini and Wagner (2017), looks at nonlinear distance norms. The general L_p distance norm is written as $\|x^0 - \tilde{x}_j^0\|_p$, with the p-norm $\|\cdot\|_p$ defined as

$$\|x^0 - \tilde{x}^0\|_p = \left(\sum_{j=1}^{n_0} |x_j^0 - \tilde{x}_j^0|^p\right)^{\frac{1}{p}}.$$

We will explore the nonlinear distance norms L_0, L_2 and L_∞ used by Carlini and Wagner (2017) and incorporate them linearly into the model to evaluate the feasibility of constructing adversarial examples through an MILP model. Each distance metric serves a different purpose in the construction of the adversarial examples. The L_0 distance metric minimizes the number of pixels that change. It is hence likely that less pixels change compared to other distance metrics, but the few pixels that do change will change more drastically. In contrast, the L_∞ distance metric minimizes the maximum change of any pixel. Consequently, a large amount of pixels will presumably change by small amounts. The resulting distances will differ from each other: whilst the L_0 distance metric will likely produce a small integer value representing the number of changed pixels, the maximum change of any pixel in the L_∞ distance metric will presumably be close to zero. The L_2 distance metric bears resemblance to the L_1 distance metric, leaning more towards smaller changes in individual pixels due to its squared terms. We will compare the performances of the MILP model in the construction of each of these types of adversarial examples, and see which ones can be handled and which ones should be constructed through other methods. The quality of the produced adversarial inputs can be assessed by retraining the models with the adversarial inputs and evaluating the robustness of the newly trained models. Due to the different magnitudes of the distance metrics, the absolute values of the distances will not be directly compared to each other. Instead, the objective values of the creation of new adversarial examples with the adversarial trained network will be compared to the objective values of the original network. Sections 4.1-4.3 discuss the incorporation of the L_0, L_2 and L_∞ distance metrics respectively in the 0-1 MILP model for adversarial examples. Section 4.4 elaborates on the adversarial training procedure.

4.1 L_0 distance norm

The L_0 distance metric measures the number of inputs $\tilde{x}_i^0$ such that $\tilde{x}_i^0 \neq x_i^0$. This distance metric therefore looks at the number of pixels that change in the adversarial example compared to the original figure. To incorporate this into the MILP model, we introduce a binary variable u_j for $j = 1, \ldots, n_0$, taking on the value 1 if input j has been perturbed, and 0 otherwise. The objective to be minimized is then $\sum_{j=1}^{n_0} u_j$, and constraints (16) are replaced by

$$-u_j \leq x_j^0 - \tilde{x}_j^0 \leq u_j, \; u_j \in \{0,1\}, \;\; j = 1, \ldots, n_0. \tag{17}$$

Since x_j^0 is normalized to represent the gray level of the input in the interval $[0,1]$, the expression $x_j^0 - \tilde{x}_j^0$ takes on values in the interval $[-1,1]$. In case input j is perturbed, u_j will be equal to 1 and constraints (17) will be satisfied.

4.2 L_2 distance norm

The L_2 distance metric measures the Euclidean distance between x^0 and $\tilde{x}^0$. This distance metric can be implemented in the model given by (6)-(16), using the objective function $\sqrt{\sum_{j=1}^{n_0} d_j^2}$. This is a nonlinear expression, and could therefore be troublesome for commercial MILP solvers. To linearize the objective function, we will use the λ-formulation that includes SOS2-constraints for piecewise linear approximations, described by Bisschop (2006).

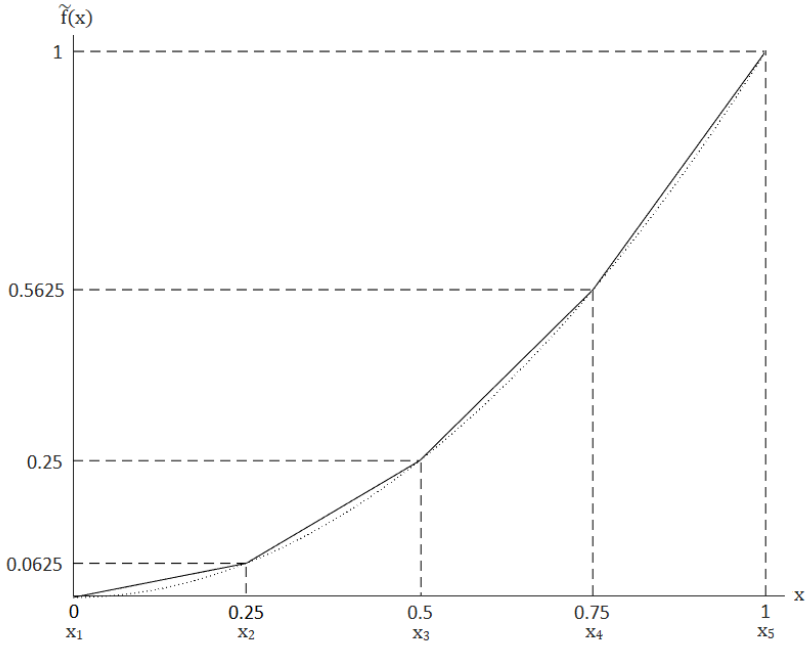

Figure 1: Piecewise linear approximation of $f(x) = x^2$, $0 \leq x \leq 1$, with 5 breakpoints.

An example of this linear approximation is shown in Figure 1. The function $f(x) = x^2$, $0 \leq x \leq 1$, is modeled with the aid of five breakpoints, x_1, x_2, x_3, x_4 and x_5. Let $f(x_1)$, $f(x_2)$, $f(x_3)$, $f(x_4)$ and $f(x_5)$ denote the corresponding function values. The breakpoints in this example are 0, 0.25, 0.5, 0.75 and 1, and

the corresponding function values are 0, 0.0625, 0.25, 0.5625 and 1. Any points between two breakpoints can then be obtained as the weighted sum of two breakpoints. For example, $x = 0.375 = \frac{1}{2} \times 0.25 + \frac{1}{2} \times 0.5$. The approximated function value is then $\tilde{f}(0.375) = 0.140625 = \frac{1}{2} \times 0.0625 + \frac{1}{2} \times 0.25$.

To model this linearization, we denote by $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ and λ_5 five nonnegative weights that sum up to 1. The linear approximation is then as follows:

$$\lambda_1 f(x_1) + \lambda_2 f(x_2) + \lambda_3 f(x_3) + \lambda_4 f(x_4) + \lambda_5 f(x_5) = \tilde{f}(x), \tag{18}$$

$$\lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 + \lambda_4 x_4 + \lambda_5 x_5 = x, \tag{19}$$

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 = 1, \tag{20}$$

with the additional constraint that at most two adjacent λ_i are greater than zero, to ensure that the function values are approximated by their two closest breakpoints. This can be modeled with the additional binary variable y_i, which takes on the value 1 if $\lambda_i \geq 0$, and 0 otherwise. Looking at the example presented in Figure 1, this requires the constraints

$$\lambda_i \leq y_i, \;\; i = 1, ..., 5, \tag{21}$$

$$\sum_{i=1}^{5} y_i \leq 2, \tag{22}$$

$$y_1 + y_3 \leq 1, \tag{23}$$

$$y_1 + y_4 \leq 1, \tag{24}$$

$$y_1 + y_5 \leq 1, \tag{25}$$

$$y_2 + y_4 \leq 1, \tag{26}$$

$$y_2 + y_5 \leq 1, \tag{27}$$

$$y_3 + y_5 \leq 1. \tag{28}$$

Constraints (23)-(28) ensure that for every pair of nonadjacent breakpoints, at most one of the associated y_i, and therefore λ_i, can be strictly greater than 0.

Translating this approach to our setting, x in constraints (18) and (19) is replaced with d_j, and a set of lambdas with size equal to the number of breakpoints is made for each d_j. Constraints of the form (21)-(28) are modeled for each input pixel. The square root in the objective function $\sqrt{\sum_{j=1}^{n_0} d_j^2}$ is a monotonic transformation. It is therefore computationally easier to neglect the square root and to simply minimize the function $\sum_{j=1}^{n_0} d_j^2$. The actual objective value can then be found by taking the square root after optimizing. It is furthermore noteworthy that each individual d_j^2 is a convex function. The objective function $\sum_{j=1}^{n_0} d_j^2$, which is a sum of convex functions, is consequently also convex. As a result, the piecewise linear approximation will provide upper bounds on the actual Euclidean distances. We will evaluate the computational effort needed and the upper bound gaps for several numbers of breakpoints.

4.3 L_∞ distance norm

The L_∞ distance metric measures the maximum change of any of the inputs:

$$\|x^0 - \tilde{x}^0\|_\infty = \max\left(|x_1^0 - \tilde{x}_1^0|, \ldots, |x_{n_0}^0 - \tilde{x}_{n_0}^0|\right).$$

This can be incorporated in the 0-1 MILP model by introducing the variable W, $W \geq 0$, representing the maximum change of any input. The objective to be minimized is then W, and constraints (16) are replaced by the constraints

$$-W \leq x_j^0 - \tilde{x}_j^0 \leq W,\ W \geq 0,\ \ j = 1, \ldots, n_0. \tag{29}$$

4.4 Adversarial training and evaluating robustness

To evaluate the quality of the adversarial examples, we will perform adversarial training on the neural network in a manner similar to Szegedy et al. (2013). Given an input figure $\tilde{x}^0$ with label $\tilde{d}$, an adversarial example x^0 that is correctly labeled as $\tilde{d}$ is added to the training data during the Stochastic Gradient Descent that is used by Fischetti and Jo (2018). The neural network is thus explicitly trained to correctly label adversarial examples. The retrained network will then be tested for its accuracy on test data sets that contain different adversarial examples, and will also be used to create new adversarial examples for all distance metrics. A more robust network should on average require greater distances as measured by the associated distance metric to create new adversarial examples.

5 Computational Experiments

This section evaluates the performance of the state-of-the-art MILP solver IBM ILOG CPLEX 20.1 in feature visualization and the creation of adversarial examples. The previously described bound tightening technique and the creation of adversarial examples with nonlinear distance metrics will furthermore be assessed.

Table 1: The five considered DNNs with the number of neurons present in each hidden layer.

	Hidden layers					
	1	2	3	4	5	6
DNN1	8	8	8			
DNN2	8	8	8	8	8	8
DNN3	20	10	8	8		
DNN4	20	10	8	8	8	
DNN5	20	20	10	10	10	

Table 1 shows the structures of the five DNNs that are addressed. Each DNN additionally contains an input layer consisting of 784 inputs for each pixel of the 28×28 figure and an output layer with 10 units for each possible digit. The DNNs are trained for 50 epochs with Stochastic Gradient Descent and with a categorical cross-entropy loss function, leading to accuracies of 93.19%, 92.83%, 95.54%, 95.52% and 96.10% for the five DNNs, respectively.

The DNNs are coded and trained using the programming language Python in PyCharm version 2021.1.1 with the aid of Tensorflow. The 0-1 MILP model and the bound tightening procedure are

programmed in Java using the IDE Eclipse 2018-12 (version 4.10) and executed on a PC with a 2.8 GHz Intel Core i7-7700HQ processor and 16 GB RAM. The MILP solver IBM ILOG CPLEX 20.1 with default settings is used to solve the model.

Section 5.1 looks at several feature visualization inputs. In Section 5.2 we evaluate the performances of the DNNs in the creation of adversarial examples with the bound tightening procedure described in Section 3.1. Section 5.3 extends this evaluation by incorporating the nonlinear distance metrics in the DNNs. Lastly, in Section 5.4 we will perform adversarial training and compare the created adversarial examples.

5.1 Feature visualization

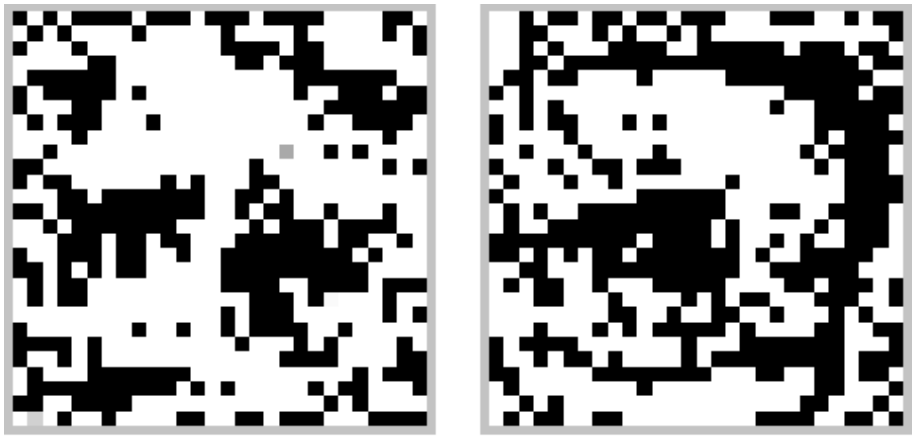

Figure 2: Inputs that maximize the activation of $UNIT(2,2)$ and $UNIT(3,1)$ in DNN1.

Feature visualization inputs are created for each $UNIT(j,k)$ with the model given by (6)-(14) and the objective function x_j^k. Figure 2 shows two example inputs that maximize the activation of $UNIT(2,2)$ and $UNIT(3,1)$ in DNN1. For the basic DNN containing 3 hidden layers of 8 internal units each, every model could be solved in less than a second. No visual patterns can be recognized in Figure 2. Input examples for each label can be found in the Appendix.

5.2 Adversarial examples

For the purpose of creating adversarial examples, the weights and biases that were obtained after training are used to build the model described by (6)-(14), with the addition of the variables d_j and the associated constraints (15)-(16). The upper bounds of all d_j variables are set to positive infinity, meaning that there is no limit to the change of the inputs. The bound tightening procedure described in Section 3.1 is applied to construct upper bounds for all x_j^k and s_j^k variables. For the three largest DNNs, this requires significant preprocessing times. We therefore also look at the performance of weaker bounds, which are computed by imposing a time limit of 1 second on the computation of each bound.

Figure 3 shows adversarial examples for a handwritten 0 and a handwritten 9 that are created using DNN1. The original handwritten digit is shown on the left, along with adversarial examples which are

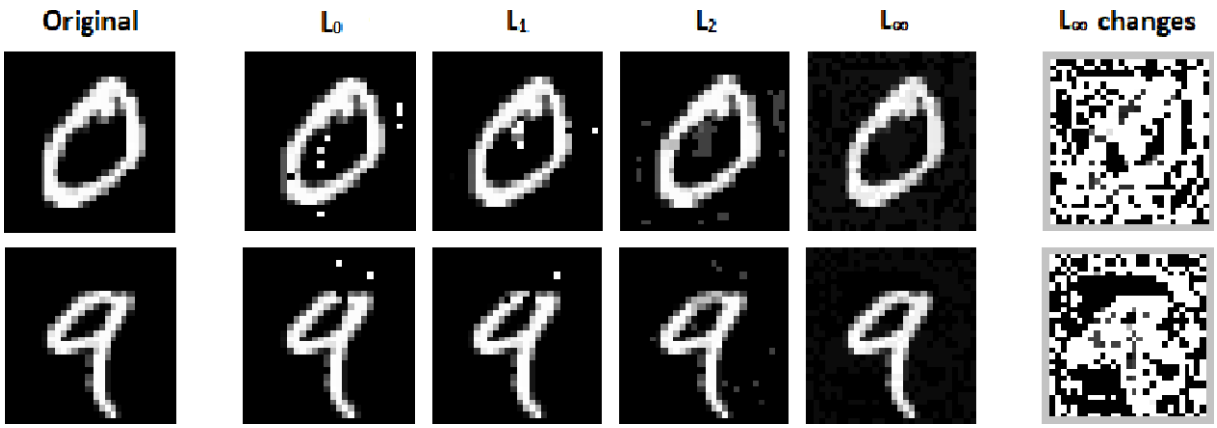

Figure 3: Adversarial examples created with DNN1 for a handwritten 0 and 9, using all four distance metrics. The pixel changes for L_∞ are shown in the far right column and have been rescaled such that white corresponds to unchanged pixels, and black corresponds to the maximum changed pixel.

built using all four distance metrics. Due to the latent changes in the adversarial examples that were created with the L_∞ distance metric, the rescaled pixel changes with white pixels representing unchanged pixels and black pixels representing the maximum changed pixel are displayed on the right. The figure shows that the amount of pixels that changes increases over the distance metrics from left to right, whereas the amount that pixels change by on average decreases. The L_0 distance metric minimizes the amount of changed pixels, resulting in a small number of pixels changing completely from black to white and from white to black. The L_∞ distance metric minimizes the maximum amount a pixel is changed by, leading to numerous pixels changing very slightly. The L_1 and L_2 distance metrics can be seen as compromises between the two extremes, with the L_2 distance metric preferring to change more pixels by small amounts than L_1. Adversarial examples for each digit can be found in the Appendix.

Adversarial examples are created for 100 different instances of the MNIST data set with a time limit of 300 seconds. Table 2 presents some statistics of the 100 runs. Columns 2-5 contain results of the basic model. The column "%solved" reports the percentage of instances that are solved to optimality within the given time limit. The column "%gap" shows the optimality gap of the best obtained solution in percentages, with each optimal solution having an optimality gap of 0. The column "Nodes" presents the average number of branching nodes explored during the computation, and the column "Time (s)" contains the average running times. Columns 6-10 and 11-15 report the results of the improved models with exact and weaker bounds, respectively. The columns "t.pre." display the time spent preprocessing the bounds.

Table 2: Performance of the basic and improved models with exact and weaker bounds, with a time limit of 300 seconds.

	Basic model				Exact bounds					Weaker bounds				
	%solved	%gap	Nodes	Time (s)	t.pre.	%solved	%gap	Nodes	Time (s)	t.pre.	%solved	%gap	Nodes	Time (s)
DNN1	100	0.00	1,994	0.99	6.48	100	0.00	313	0.50	6.48	100	0.00	313	0.50
DNN2	99	0.08	108,412	38.09	20.93	100	0.00	7,133	4.51	20.93	100	0.00	7,133	4.51
DNN3	63	17.32	468,387	165.16	329.54	99	0.27	10,564	19.49	56.27	99	0.29	46,742	20.81
DNN4	48	21.65	613,642	214.48	343.36	99	0.32	63,686	41.93	73.43	98	0.36	63,123	48.80
DNN5	8	66.48	580,653	290.04	6,947.64	80	8.97	74,135	142.65	69.67	70	10.68	188,455	165.58

Table 2 shows that the basic model is able to solve the instances for the first two DNNs with little issues. For the last three DNNs, the basic model fails to terminate within the specified time limit for a large percentage of the instances and provides significant optimality gaps. The improved model with exact bounds outperforms the basic model for all DNNs and only starts to experience difficulties for the largest DNN. The model with exact bounds also explores notably less branching nodes on average and solves the instances in shorter running times.

The column "t.pre" of the exact bounds in Table 2 shows that the preprocessing times of the exact bounds are negligibly small for the first two DNNs, whilst it took 329.54 seconds for DNN3, 343.36 seconds for DNN4 and 6,947.64 seconds for DNN5. We therefore investigate the performance of weaker bounds, which are constructed with a time limit of 1 second for each bound. The column "t.pre" of the weaker bounds shows that this significantly decreases the preprocessing times without substantially reducing the performance of the DNNs. The model with weaker bounds is unable to solve 1 more instance for DNN4 and 10 more instances for DNN5 than the model with exact bounds. Our results show that a short preprocessing phase still provides a large improvement over the basic model.

The performances of the basic model and the improved model with weaker bounds are further compared in Table 3 with a time limit of 3600 seconds. The weaker bounds are again constructed with a time limit of 1 second for all bound computations. Each run is terminated as soon as an optimality gap of 1% is reached. Columns 2-5 show the results of the basic model, with the column "#timlim" indicating the number of times the time limit is reached. The average optimality gap is also shown in the column "%gap", along with the average number of branching nodes explored in the column "Nodes" and the average computing time in the column "Time (s)". Columns 6-9 display the results for the improved model with weaker bounds.

Table 3: Performance of the basic and improved models with weaker bounds, with a time limit of 3600 seconds. The runs are terminated when an optimality gap of less than 1% is reached.

	Basic model				Weaker bounds			
	#timlim	%gap	Nodes	Time (s)	#timlim	%gap	Nodes	Time (s)
DNN1	0	0.44	1,352	0.78	0	0.30	158	0.37
DNN2	0	0.86	149,986	49.07	0	0.77	7,113	4.74
DNN3	4	2.16	1,859,162	528.27	0	0.85	46,959	21.73
DNN4	7	2.86	2,819,661	809.09	0	0.94	84,694	50.23
DNN5	68	35.49	6,169,553	2802.87	0	0.99	367,044	261.68

Table 3 shows that the improved model with weaker bounds is able to solve each instance to optimality within the given time limit. The basic model hits the time limit 68 times for DNN5, explores a great number of branching nodes and provides significant optimality gaps compared to the improved model. Even with weaker bounds, the improved model is therefore able to outperform the basic model for all DNNs. This is in line with the results found in Table 2.

5.3 Distance metrics

The same setup is maintained for the creation of adversarial examples for all four distance metrics. Adversarial examples are created for the same 100 instances of the MNIST data set that were used previously to replicate the results of Fischetti and Jo (2018). Each instance has a time limit of 300 seconds, and the exact bounds that are obtained through the bound tightening procedure described in Section 3.1 are utilized.

Table 4: Performance of the L_0 and L_1 distances in the improved models using the exact bounds, with a time limit of 300 seconds for each instance.

	L_0					L_1				
	Obj.	%solved	%gap	Nodes	Time (s)	Obj.	%solved	%gap	Nodes	Time (s)
DNN1	8.50	100	0.00	1349	3.17	7.70	100	0.00	313	0.50
DNN2	7.53	97	1.07	30,908	53,38	6.71	100	0.00	7,133	4.51
DNN3	8.23	74	11.79	34,827	138.16	7.35	99	0.27	10,564	19.49
DNN4	7.41	44	35.02	61,866	223.19	6.53	99	0.32	63,686	41.93

Table 4 shows statistics of the L_0 and L_1 distances. The columns "Obj." contain the average objective values over the 100 runs, and every other column maintains the same definition as before. It is apparent that L_0 gets into trouble for DNN3 and DNN4, being unable to solve a large proportion of the instances within the time limit and leaving significant optimality gaps compared to L_1. This can be explained by the fact that L_0 replaces the continuous distance constraints with binary variable constraints, hampering the linear programming approach taken by modern MILP solvers during the solving of the model. It is furthermore notable that with the exception of DNN3, the more complex the DNN, the lower the average objective values are. Lower objective values entail adversarial examples with more resemblance to the original image in terms of the distance metric which is minimized. DNN3 may therefore be redundant, as the additional computational complexity compared to DNN2 does not lead to adversarial examples which are closer to the original image.

Table 5: Performance of the L_2 distance with 5 breakpoints and the L_∞ distance in the improved models using the exact bounds, with a time limit of 300 seconds for each instance.

	L_2						L_∞				
	Obj.	%bound	%solved	%gap	Nodes	Time (s)	Obj.	%solved	%gap	Nodes	Time (s)
DNN1	11.12	6.73	100	0.00	514	1.66	0.07	100	0.00	348	2.37
DNN2	9.54	3.13	100	0.00	9,089	14.58	0.06	100	0.00	6,034	16.82
DNN3	10.29	3.33	95	1.91	34,920	58.15	0.06	96	1.78	24,068	62.72
DNN4	8.97	4.49	92	2.97	60,391	100.62	0.05	93	2.34	37,881	122.41

To implement the L_2 distance metric, the linearization in Section 4.2 is employed with 5 uniformly distributed breakpoints. Table 5 shows several statistics of the L_2 and L_∞ distance metrics. The column

"%bound" displays the average gap between the upper bound created by the linearization and the actual obtained distance over the 100 runs. Comparing the results in Table 5 to the L_1 distance in Table 4, it can be seen that the computational complexity of these two nonlinear distance metrics does not impact the performance of the models too heavily. For L_2, 5 and 8 instances are not solved to optimality, whilst L_∞ leaves 4 and 7 instances unsolved in DNN3 and DNN4, respectively. The instances also require longer computation times compared to L_1, but on average still terminate well under the imposed time limit.

Looking at the quality of the linearization of L_2, Table 5 shows that the smallest gap of 3.13% is obtained for DNN2. We further explore the addition of breakpoints to increase the accuracy of the linearization.

Table 6: Performances of 9, 11, 17, 21, 33 and 41 breakpoints in the linearization of L_2, using the exact bounds and a time limit of 300 seconds for each instance.

	DNN1					
#breakpoints	Obj.	%bound	%solved	%gap	Nodes	Time (s)
9	13.76	4.22	100	0.00	488	2.17
11	14.57	3.62	100	0.00	476	2.61
17	15.96	2.66	100	0.00	453	4.32
21	17.07	4.50	100	0.00	460	4.73
33	16.94	1.51	100	0.00	437	7.76
41	17.46	2.34	100	0.00	485	11.25
	DNN2					
#breakpoints	Obj.	%bound	%solved	%gap	Nodes	Time (s)
9	11.73	2.02	100	0.00	7,395	17.20
11	12.48	1.74	100	0.00	7,352	17.37
17	13.96	1.38	100	0.00	7,446	28.41
21	14.78	2.08	100	0.00	7,180	34.89
33	15.10	0.89	98	1.50	6,630	61.39
41	15.47	1.22	94	3.76	7,356	97.47
	DNN3					
#breakpoints	Obj.	%bound	%solved	%gap	Nodes	Time (s)
9	12.73	2.20	94	3.11	32,135	62.47
11	13.49	1.89	95	2.33	31,260	75.81
17	15.09	1.39	91	5.18	22,291	99.01
21	15.85	1.84	84	8.78	22,583	128.49
33	16.19	0.83	62	20.40	18,083	191.66
41	16.42	0.97	68	20.68	17,175	189.44

Table 6 compares the performance of using 9, 11, 17, 21, 33 and 41 uniformly distributed breakpoints. Adding more breakpoints to the linearization of L_2 decreases the gap between the created upper bounds and the actual distances compared to the original 5 breakpoints for each DNN. Using 21 breakpoints results in larger bound gaps, however, compared to using 17 breakpoints. The same holds when the number of breakpoints is increased from 33 to 41. Each number of breakpoints is able to produce adversarial examples in a matter of seconds for the first DNN. Every instance is solved within the specified time limit for the first four numbers of breakpoints in DNN2, but leaves a small number of instances

unsolved for 33 and 41 breakpoints. The additional computational effort required for this linearization becomes apparent in DNN3, with large portions of instances being unsolved. It is furthermore notable that adding breakpoints increases the average obtained objective value. Whilst adding breakpoints increases the quality of the linear approximation in terms of the bound gaps, it thus produces adversarial examples that differ more from the original image. Using 17 and 33 breakpoints produces the smallest bound gaps, respectively leaving gaps of 2.66% and 1.51% for DNN1, and gaps of 1.38% and 0.89% for DNN2.

5.4 Adversarial training

For the purpose of adversarial training, 2000 adversarial examples are created for each distance metric using DNN1 with exact bounds and with a time limit of 300 seconds for each instance. Out of these examples, 1000 are added to the MNIST training data set. The remaining 1000 are used as test data to evaluate the performance of the DNN. Table 6 in Section 5.3 showed that using 17 and 33 breakpoints provides the most accurate approximations. Since the usage of 33 breakpoints substantially increases the computation times and distances from the original images, 17 breakpoints are used for the adversarial examples that are created with the L_2 distance.

Table 7: Accuracies of the adversarial trained DNN1, evaluated on both the MNIST test data set and test data sets containing different adversarial examples created with each distance metric.

	Accuracy				
Training data	L_0	L_1	L_2	L_∞	MNIST data
L_0	0.123	0.107	0.091	0.086	0.923
L_1	0.105	0.125	0.092	0.094	0.918
L_2	0.108	0.104	0.109	0.104	0.916
L_∞	0.094	0.102	0.104	0.105	0.928
MNIST data	0.081	0.081	0.083	0.084	0.932

Table 7 displays the accuracies of the adversarial trained networks. Each rows corresponds to a network that has either been trained with adversarial examples created with the aid of the distance metric that is indicated in the column "Training data", or with the MNIST data without any adversarial examples. The columns "L_i" show the accuracies evaluated on a test data set containing adversarial examples created with distance metric L_i, $i \in \{0, 1, 2, \infty\}$. Furthermore, the column "MNIST data" contains the accuracies with respect to the MNIST test data set. For instance, the DNN that has been trained on adversarial examples that were created with the L_1 distance metric has an accuracy of 0.092, or 9.2%, when evaluated on a test data set containing adversarial examples that were created with the L_2 distance metric, and an accuracy of 0.918, or 91.8% on the MNIST test data set.

As Table 7 shows, adversarial training slightly decreases the accuracy that is obtained for the MNIST test data set, but increases the accuracies of the adversarial example test sets by 2.1% on average. Excluding the last row and column, the cross entries in the table are the maximum of each row, indicating that a network which is trained on a particular set of adversarial examples achieves the highest accuracy

when evaluated on a test set that consists of adversarial examples created with the same distance metric. All accuracies are between 8% and 13%, which shows that there is no distance metric which substantially improves the accuracy over any other one. Using a network that is trained on adversarial examples which were created with the L_2 and L_∞ distance to evaluate L_0 and L_1 distance adversarial examples seems to perform slightly better, however, than using a network that is trained on L_0 and L_1 distance adversarial examples to evaluate adversarial examples which were created with the L_2 and L_∞ distances.

To further explore the robustness of the network after adversarial training, the weights and biases obtained from adversarial training are used in DNN1 to create new adversarial examples for each distance metric, using the same 100 instances as before. The columns "Objective" in Table 8 contain the average objective values obtained by using the DNN that has been retrained for distance metric L_i in the column "Training data", to create adversarial examples for distance metric L_j below "Objective", $i, j \in \{0, 1, 2, \infty\}$. The columns "%difference" display the percentage difference between the objective values in the columns "Objective" and the objective values obtained for DNN1 in Table 4, 5 and 6. For instance, the DNN that is retrained for L_∞ has an average objective value of 8.91 when it is used to create L_0 adversarial examples, which is 4.82% higher than the objective value of 8.50 in Table 4.

Table 8: The obtained objective values from creating adversarial examples with the adversarial trained models and the percentage differences between these objective values and the objective values in Table 4, 5 and 6.

	Objective				%difference			
Training data	L_0	L_1	L_2	L_∞	L_0	L_1	L_2	L_∞
L_0	6.85	6.43	13.16	0.06	-19.41	-16.49	-17.54	-14.29
L_1	7.67	6.93	14.53	0.06	-9.76	-10.00	-8.96	-14.29
L_2	8.28	7.53	15.04	0.07	-2.59	-2.21	-5.76	0.00
L_∞	8.91	8.13	16.44	0.07	4.82	5.58	3.01	0.00

Table 8 shows that the percentage differences for L_0 and L_1 adversarial trained networks are negative. This means that training the network on L_0 and L_1 distance adversarial examples does not make the networks more robust, measured in terms of the minimal distance required to create new adversarial examples. The percentage differences for L_2 adversarial examples are furthermore nonpositive, and the percentage differences for L_∞ are nonnegative. This indicates that L_∞ succeeds in making the networks more robust, requiring 4.82% more pixel changes for the creation of L_0 adversarial examples, 5.58% for L_1 and 3.01% for L_2.

A possible explanation for these findings could be the properties of the adversarial examples that are created with the different distance metrics. Since the L_0 and L_1 adversarial examples drastically change several pixels, training the neural network on these images results in the network being trained to recognize the specific changed pixels in the adversarial examples rather than the general patterns of each digit. Overfitting on such adversarial examples therefore aids the network in identifying these critical pixels, facilitating the generation of new adversarial examples. The L_∞ distance metric, however, minimizes the maximum change of any pixel. It produces adversarial examples in which the perturbations

are spread over the entire image, but the digits themselves remain close to the original input. Training the network on such images makes the network identify the distinctive features of each digit instead of pinpointing specific pixels, allowing the adversarial training procedure to succeed in making the network more robust to the creation of more adversarial examples.

6 Conclusions

We have explored the 0-1 MILP model formulation for DNNs proposed by Fischetti and Jo (2018). This model is utilized for feature visualization and the creation of adversarial examples in the context of recognizing handwritten digits from the MNIST data set by LeCun and Cortes (2010). We have extended this framework by linearly introducing nonlinear distance metrics in the model through piecewise linear approximations. The qualities of these adversarial examples are evaluated based on the accuracies and robustness of the adversarial trained models.

We find that the model can be solved in a matter of seconds for smaller DNNs. The bound tightening procedure has proven to be effective in reducing the computation times, which significantly increases the solvability of larger DNNs. Whilst the L_0 distance is unable to solve a large portion of the instances for the larger DNNs, the model is able to handle L_2 and L_∞ distances without excessively hindering the computation times. The λ-formulation of Bisschop (2006) provides an accurate approximation of the L_2 distance metric, with an upper bound gap of 1.38% in the case of 17 breakpoints. Training the neural network on L_∞ adversarial examples provides the highest accuracies in correctly labeling adversarial examples and makes the model most robust to subsequent adversarial attacks.

Future research could look at reducing the computational effort needed to find exact solutions for larger and more realistic DNNs. This could be done by developing heuristics, possibly based on simplified versions of the model. Additionally, newer adversarial training techniques and robustness metrics could be implemented to more accurately measure the quality of the generated adversarial examples.

References

Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., and Vielma, J. P. (2020). Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, pages 1–37.

Belotti, P., Bonami, P., Fischetti, M., Lodi, A., Monaci, M., Nogales-Gómez, A., and Salvagnin, D. (2016). On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3):545–566.

Bisschop, J. (2006). *AIMMS Optimization Modeling*. Lulu.com.

Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE.

Cheng, C.-H., Nührenberg, G., and Ruess, H. (2017). Maximum resilience of artificial neural networks.

In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer.

Dutta, S., Jha, S., Sankaranarayanan, S., and Tiwari, A. (2018). Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer.

Fischetti, M. and Jo, J. (2018). Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Grimstad, B. and Andersson, H. (2019). Relu networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580.

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

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE.

Qin, Y., Hunt, R., and Yue, C. (2019). On improving the effectiveness of adversarial training. In *Proceedings of the ACM International Workshop on Security and Privacy Analytics*, pages 5–13.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tjeng, V., Xiao, K., and Tedrake, R. (2017). Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2017). Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*.

A Appendix

A.1 Feature visualization and adversarial examples

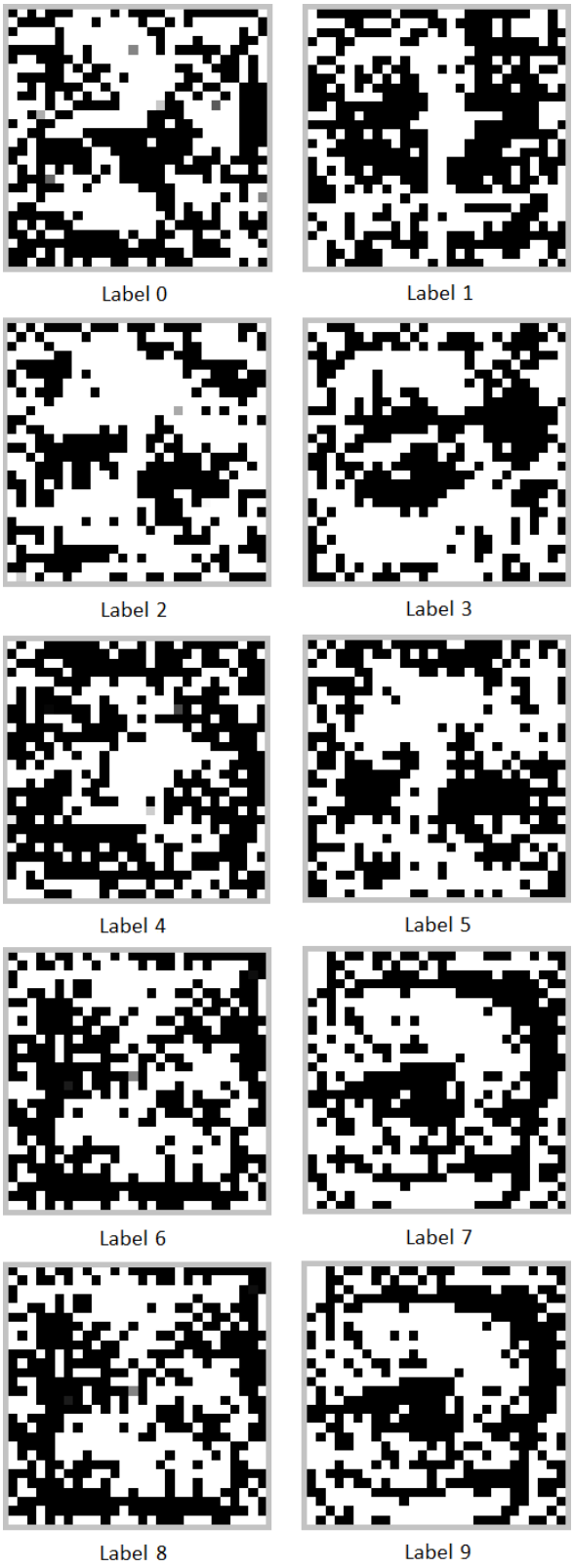

Figure 4: Inputs that maximize the activation of hidden units in DNN1 for each digit.

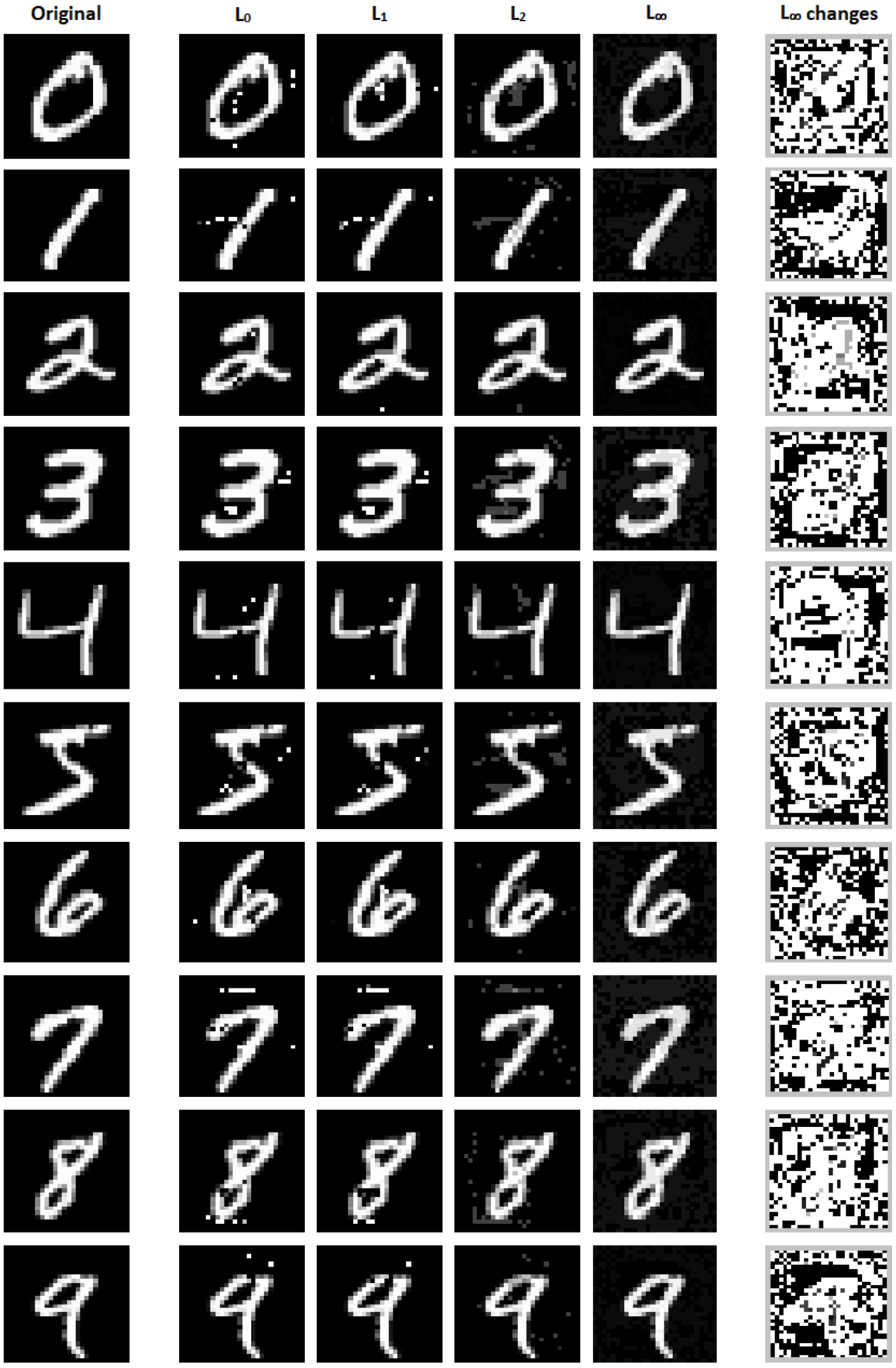

Figure 5: Adversarial examples for each digit, using all four distance metrics. The pixel changes for L_∞ are shown in the far right column and have been rescaled such that white corresponds to unchanged pixels, and black corresponds to the maximum changed pixel.

A.2 Programming code

The following classes are used for this paper:

- *Thesis*, which contains the 0-1 MILP model for the DNNs programmed in Java. Feature visualization, the creation of adversarial examples and the bound tightening procedure are all performed in this class. Several similar classes for the five different DNNs, the different distance metrics and different numbers of breakpoints are also created.
- *main*, which contains the code for the DNNs programmed in Python. This class is used to obtain the weights and biases of the DNNs and to perform adversarial training.
- *images*, which accesses the MNIST data set and exports the images to text files for the 0-1 MILP model. This class is also programmed in Python.