Improving the robustness of Deep Neural Networks using adversarial examples

Bachelor thesis Econometrie & Operationele Research

Tim Tjhay, 495230

Supervisor: B.T.C. van Rossum

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

July 4, 2021

Erasmus University Rotterdam

Erasmus School of Economics

Abstract

In this thesis several approaches are evaluated that attempt to improve the robustness of Deep Neural Networks(DNNs) using adversarial examples. A comparison is made between three approaches, the first two of which consist of retraining the DNN with these adversarial examples. The third approach attempts to improve the robustness using an input perturbation which is based on the Mixed Integer Linear Programming formulation of a DNN from Fischetti and Jo [2018]. Using this comparison, it was shown that retraining the DNN results in the greatest improvement in robustness as the accuracy on adversarial examples improved quite drastically. However, the input perturbation does improve the robustness to some extent as well, compared to the original DNN without perturbation.

Contents

1 Introduction

Deep Neural Networks(DNNs) are a type of Deep Learning architecture that is becoming increasingly popular. They can be used for many different purposes such as image classification, speech recognition and the detection of spam mails. Because of this increasing popularity there is also a growing interest in methods that find weaknesses in these networks. One of these methods, namely adversarial machine learning, slightly modifies an input, that is correctly classified by a DNN, to find adversarial examples that trick the DNN into misclassifying it. This would, for example, allow senders of spam mails to hide their mails from the spam detection of email platforms. As those email platforms would like to reduce the probability of this happening as much as possible, defenses against these adversarial examples are being studied extensively, as well as methods to prevent them from being created.

Several methods to craft adversarial examples can be found in the existing literature. One of these methods uses the Mixed Integer Linear Programming(MILP) formulation of DNNs from Fischetti and Jo [2018], that is created by linearizing the layers and the architecture of a DNN by using binary variables. One advantage of this approach is that extra constraints can be added to these examples relatively easily due to the flexible nature of the MILP formulation. In the existing literature several similar MILP formulations for DNNs can be found that are all used for different purposes, which highlights this flexibility. The improvement of the defensive strength against adversarial examples can also be achieved using multiple approaches. An approach that can be found in the existing literature adds the adversarial examples to the data set used to train the DNN and retrains it. This approach has been shown to regularize the DNN, meaning the DNN performs better on unseen data after the retraining process. Another approach detects if an input is an adversarial example which can be done using several unique approaches.

As the MILP formulation from Fischetti and Jo [2018] provides a relatively easy method to create adversarial examples, this thesis attempts to use them to improve the defensive strength of the DNN and to reduce the weaknesses found by these adversarial examples. This can be done by using them to retrain the DNN to make it more reliable as is done in the existing literature. It is also evaluated if this retraining reduces any overfitting that might be incurred by the complex nature of DNNs. In this thesis these two factors, the defensive strength and the measure of overfitting, are summed up as the robustness of a DNN which leads to the following research question: "*To what extent can adversarial examples improve the robustness of Deep Neural Networks?*".

Three different approaches are evaluated to answer the research question: The first approach is proposed by Papernot et al. [2016] and Goodfellow et al. [2014] and adds a set of adversarial examples to the training data used to originally train the DNN and completely retrains it. The second approach retrains the DNN to maximize the number of correctly specified adversarial examples using the initial weights and biases as a starting point. This method, unlike the first one, uses only the adversarial examples during the retraining process and not the training data. The third and final approach aims to find a perturbation for the input that reduces the importance of input values(e.g. pixels for images) that could be weaknesses of the DNN, based on the adversarial examples. This approach is especially interesting as it has not been attempted in the existing literature and does not require the DNN to be altered. This could be helpful if, for example, the DNN is provided by an external company that does not allow the user to change the weights and biases. These three approaches are then compared to each other and to a base case, where the initial weights and biases are used on an unperturbed input.

In this thesis, the method that is used to create adversarial examples uses the MILP formulation of DNNs proposed by Fischetti and Jo [2018]. Since these examples are an important aspect of the performed research, the methodology from Fischetti and Jo [2018] is closely followed and the obtained results are replicated. Additionally, the same data set is used, i.e. the MNIST data set from Lecun et al. [1998] which is publicly available, e.g. in the Python library Keras.

After comparing the test set accuracy achieved by these approaches, it became clear that the first approach achieved the greatest improvement in robustness. This approach achieved an accuracy of 90.78% on adversarial examples and 33.5% on images that are incorrectly classified by the original DNN, which could indicate a reduction of the measure of overfitting. Additionally, the second approach appears to perform quite well, with an accuracy of 79.11% on adversarial examples. However, this approach does not seem to reduce overfitting as well as the first one. While the third approach does not perform nearly as well as the other two, it does allow the DNN to classify 51.89% of the adversarial examples correctly which shows that the approach does have potential. Furthermore, all three approaches do not compromise the accuracy on original images that are correctly classified by the original DNN, as the accuracy on these images is over 96% for all approaches.

The rest of this thesis is structured as follows: Section 2 contains a review of the existing literature and Section 3 gives a detailed description of the used methods. Afterwards, the results are presented in Section 4 and finally, a conclusion is given in Section 5 based on the results.

2 Literature review

As the research in this thesis will build upon the work done in Fischetti and Jo [2018], it is also the article this work relates to the most. This is mainly due to the MILP formulation of DNNs that is given in the article. Similar formulations have also been published by Tjeng [2018], Cheng et al. [2017] and Serra et al. [2017]. All three articles do, however, use the model for different purposes. In Tjeng [2018] the model is used as a complete verifier that determines whether a property on the output of a DNN holds for every input in a bounded input domain. Alternatively, Cheng et al. [2017] use it to compute maximum resilience bounds that are defined as the maximum amount of input perturbation that does not change a classification. Finally, Serra et al. [2017] compute how many linear regions a piecewise linear function can have if it is represented by a DNN. While there are some similarities, especially between the first two articles and Fischetti and Jo [2018] that all use adversarial examples in some way, these different approaches show how much can be done using this formulation.

The main use of the MILP formulation in this thesis will be to craft adversarial examples, as is done in Fischetti and Jo [2018]. These adversarial examples are a main aspect of adversarial machine learning and, as is stated in Szegedy et al. [2013], they can be found by slightly altering a correctly classified input in such a way that the DNN misclassifies it. Besides the method using the MILP formulation, there are many alternative approaches to find these adversarial examples. In Papernot et al. [2016] the forward derivative is used to create a saliency map showing which input values, or pixels in the case of images as was used in the article, are potential weaknesses. This saliency map is then used to create adversarial examples. Alternatively, Goodfellow et al. [2014] use the linear behaviour of commonly used activation function to craft adversarial examples using the gradient. Szegedy et al. [2013] use yet another different approach as the problem is formulated as a box-constrained optimization problem which is approximated using the Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm. It should, however, be noted that the approach using the MILP formulation allows the one crafting the adversarial examples to impose extra constraints, such as a maximum number of perturbed input values, relatively easily compared to most alternative approaches.

In Papernot et al. [2016] a method is proposed that measures the robustness of a DNN against adversarial examples. This robustness gives a measure of how difficult it is to create adversarial examples and can also be seen as the defensive strength of the DNN. The improvement of the defensive strength against these adversarial examples is another part of adversarial machine learning that has been studied extensively. Goodfellow et al. [2014] and Papernot et al. [2016] were able to regularize a DNN by adding adversarial examples to the training data which improved the DNN's performance on unseen inputs. A completely different approach is used by Prakash et al. [2018], Meng and Chen [2017] and Gupta et al. [2020] who all attempt to detect if an input is an adversarial example by using, respectively, pixel deflection, a framework called

MagNet and a genetic algorithm.

The relevance of this thesis mainly lies in the comparison between the retraining approach used in Goodfellow et al. [2014] and Papernot et al. [2016], and the two alternative approaches. Namely, the retraining approach that only uses adversarial examples as training data and the approach that uses the input perturbation. Especially the perturbation approach, that tries to improve the robustness of the DNN using the MILP formulation, could be interesting as this was not attempted in the other articles using the formulation. The research is mainly of scientific relevance in the field of adversarial machine learning as the three approaches to improve defensive strength against adversarial examples will be compared and evaluated.

3 Methodology

In this section the methods used to answer the research question are explained. These methods are then applied to the MNIST data set from Lecun et al. [1998] consisting of 28×28 grayscale images of handwritten digits, as is done in Fischetti and Jo [2018]. Each pixel in these images is represented by a grayscale intensity that is normalized to a value between zero and one with zero and one corresponding to black and white respectively.

The rest of this sections is structured as follows: First, in Section 3.1, a more detailed definition of a DNN is given. In Section 3.2 the MILP formulation of DNNs from Fischetti and Jo [2018], that will be used to replicate the results, is then repeated. Then the methods used to obtain the adversarial examples and the feature visualization in Fischetti and Jo [2018] are explained in Section 3.3. Finally, the proposed robustness improvement methods are presented in Section 3.4.

3.1 Deep Neural Networks

DNNs are organized into several layers that consist of neurons. The number of these layers and neurons are what distinguish DNNs from regular Neural Networks, as DNNs consist of four or more layers. In this thesis the different layers of a DNN with $K + 1$ layers are denoted by $k \in L \cup \{0\}$ where $L = \{1, ..., K\}$ is the set of layers and layer 0 and layer K correspond to the input and output layer of the DNN respectively. For every layer the neurons are denoted by $j \in J_k$ where $J_k = \{1, ..., n_k\}$ is the set of neurons in layer k and n_k is the number of neurons in that layer. The layers between the first and last one are called hidden layers and are the ones that the input is passed through in order to classify the input in the output layer. This classification is made using a transformation of the input by the different neurons in the hidden layers, that each take in the output from all neurons in the previous layer and compute a weighted sum.

Then a bias is added to this sum and a non-linear operator, called the activation function, is applied to the resulting value to get the output of the neuron. These activation functions are typically monotonic and differentiable, which is necessary for gradient-based training methods. In this thesis the Rectified Linear Unit(ReLU) operator is used, which outputs the maximum of its input and zero, but many alternative activation functions are available such as max pooling or the logistic sigmoid function. The output, also called the activation, of all neurons in a layer k is thus given as:

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

In (1) the function $\sigma(\cdot)$ is the activation function, which in this thesis is $ReLU(\cdot)$. The vector x^k consists of the output for the n_k neurons in layer k and W^{k-1} and b^{k-1} are, respectively, a matrix containing the weights and a vector containing the biases of layer $k-1$. These weights and biases for the neurons are obtained through a training process that aims to find the weights and biases that maximize the number of correctly classified inputs, which can be done through various methods. In accordance with Fischetti and Jo [2018] this thesis uses a supervised training process, that is performed for 50 epochs, to train the DNN . In this context a supervised training process means the correct classification of the training data is known to the DNN during training.

To evaluate the model the same five DNNs, that are used in Fischetti and Jo [2018], are used. Table 1 shows the number of neurons the hidden layers of these DNNs consist of.

Table 1: Number of neurons in the hidden layers of the five DNNs used to evaluate the model

	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	

3.2 MILP formulation

Formulating the DNN as an MILP implies that the layers and the architecture of the DNN are linearized using binary variables. The first thing that has to be done to do this is to linearize the activation function which, in this case, is the ReLU operator. To do this, a more general version of (1) is defined in (2) which looks at the output of a single neuron.

$$x = \sigma(w^T y + b) \tag{2}$$

Here x and b are scalars while w and y are vectors and this can then be reformulated as:

$$x - s = w^T y + b \quad ,x \geq 0 \quad ,s \geq 0 \tag{3}$$

In this equation the scalar s absorbs the negative part of the input of the ReLU operator. It is, however, quite clear that the solution to this equation is not unique as any scalar δ can be added to both x and s, which would result in $(x+\delta)-(s+\delta)=x-s$. This can be prevented by ensuring that at least one of the two scalars is equal to zero, which can be done by adding the contraint $xs=0$. To linearize this constraint a binary activation variable z has to be added that equals 1 if $x=0$ and 0 otherwise. This results in the following linear big-M constraints where the parameters M^+ and M^- are finite nonnegative values that bound the input of the ReLU operator such that $-M^- \leq w^T y + b \leq M^+$:

$$x \leq M^+(1-z) \tag{4}$$

$$s \leq M^- z \tag{5}$$

$$z \in \{0,1\} \tag{6}$$

Using these constraints the following MILP formulation for DNNs can be made:

$$\begin{aligned}
\text{maximize} \quad & \sum_{k \in L \cup \{0\}} \sum_{j \in J_k} c_j^k x_j^k + \sum_{k \in L \cup \{0\}} \sum_{j \in J_k} \gamma_j^k z_j^k && && \text{(7a)}\\
\text{subject to} \quad & \sum_{i \in J_{k-1}} w_{ij}^{k-1} x_i^{k-1} + b_j^{k-1} = x_j^k - s_j^k && , k \in L, j \in J_k, && \text{(7b)}\\
& x_j^k \leq M^+(1-z_j^k) && , k \in L, j \in J_k, && \text{(7c)}\\
& s_j^k \leq M^- z_j^k && , k \in L, j \in J_k, && \text{(7d)}\\
& lb_j^k \leq x_j^k \leq ub_j^k && , k \in L \cup \{0\}, j \in J_k, && \text{(7e)}\\
& \overline{lb}_j^k \leq s_j^k \leq \overline{ub}_j^k && , k \in L, j \in J_k, && \text{(7f)}\\
& x_j^k, s_j^k \geq 0 && , k \in L, j \in J_k, && \text{(7g)}\\
& x_j^0 \in \mathbb{R} && , j \in J_0, && \text{(7h)}\\
& z_j^k \in \{0,1\} && , k \in L, j \in J_k && \text{(7i)}
\end{aligned}$$

In this formulation the objective function (7a) represents a general objective function that can be changed depending on the problem the model is used for. The costs c_j^k and γ_j^k are given parameters that are defined based on the problem to be solved. Constraints (7b) - (7d) ensure the output of every neuron is correct, i.e. x_j^k is equal to the output of the ReLU operator. To do this, the previously explained big-M constraints are used. Constraints (7e) and (7f) define bounds for the variables x_j^k and s_j^k. It should be noted that the input layer is included in (7e) as this can be used to bound the input corresponding to their physical meaning, e.g for grayscale images one could bound the input values by zero and one as they represent the intensity of the pixels. For $k \geq 1$ the bounds are defined as $lb_j^k = \overline{lb}_j^k = 0$ and $ub_j^k, \overline{ub}_j^k \in \mathbb{R}_+ \cup \{+\infty\}$ for all neurons $j \in \{1, \ldots n_k\}$. Finally, (7g)-(7i) define the domains of the decision variables.

As big-M constraints can make it hard to solve the model to optimality, Belotti et al. [2016] showed that tightening the bounds of the variables that are used in the big-M constraint can improve how solvable the model is. To this end Fischetti and Jo [2018] propose a bound tightening method as an alternative to the bounds automatically defined by MILP solvers. This method scans the layers of the DNN, $k \in L = \{1, \dots, K\}$, in increasing order. For every neuron $j \in J_k$ in the current layer the following steps are performed. First all constraints and variables related to the other neurons in the same layer and all neurons in any subsequent layers are removed. Then the resulting model is solved twice, optimizing x_j^k and s_j^k respectively. The resulting optimal values can then be used as upper bounds and the model using these bounds is called the improved model. Due to the acyclic nature of the DNN, these bounds can then be used to compute the bounds for the neurons in the following layer.

It should be noted that, for fixed weights and biases, the computed bounds do not rely on an input but on the lower and upper bounds of the input that are defined by the problem at hand. This means the bounds can be used for different instances of the problem and they do not need to be recalculated for every input. As solving these models might take a long time for larger DNNs, Fischetti and Jo [2018] also propose a weaker version of this bound tightening technique. This version imposes a time limit of one second on every bound computation and uses the best found upper bound at that time. These bounds are then used in the weaker improved model.

3.3 Feature visualization and adversarial examples

The MILP formulation in (7a) - (7i) can be used for feature visualization and to obtain adversarial examples. For feature visualization the objective is to maximize the activation, x_j^k, of one of the neurons. This is done for neurons in the output layer to obtain inputs that maximize the activation of the different classification neurons, as is done in Fischetti and Jo [2018].

To obtain adversarial examples, the same method is used as in Fischetti and Jo [2018]. A correctly classified input image $\tilde{x}^0$ is given with the digit $\tilde{d}$ as its classification and a desired target classification is defined as $d = (\tilde{d} + 5) \bmod 10$. E.g. an input correctly classified as 0 has to be defined as 5 and an input classified as 7 has to be classified as 2. It is assumed that the the structure and parameters of the DNN are known and that every pixel of the input can arbitrarily be changed. To classify the modified input as the target class it is imposed that the activation of the neuron corresponding to the target digit in the output layer is at least 20% larger than the activation of the other digits, by adding the following constraint to (7a) - (7i):

$$x_{d+1}^K \geq 1.2 x_{j+1}^K \qquad , j \in \{0, \dots, 9\} \setminus \{d\} \tag{8}$$

A variable δ_j is added that measures the deviation from the original image, since it is desirable to reach the target classification while modifying the input as little as possible. The objective then becomes to minimize the sum of this deviation which results in the objective function $\sum_{j \in J_0} \delta_j$ that replaces (7a) in the model. In order to define δ_j the following constraint needs to be added:

$$-\delta_j \leq x_j^0 - \tilde{x}_j^0 \leq \delta_j \qquad, \delta_j \geq 0, j \in J_0 \tag{9}$$

The constraint $\delta_j < \Delta$, $j \in J_0$, can also be added to impose a maximum deviation, Δ, from the original image $\tilde{x}^0$.

3.4 Robustness improvement methods

In this subsection three approaches are given that all try to improve the robustness of the DNN. The first two approaches retrain the model using a set of adversarial examples, with a more in depth description of this process being given in Section 3.4.1. The third approach perturbs the input of the DNN in an attempt to reduce the importance of input values that could be weaknesses of the DNN. A more detailed description of this approach is given in Section 3.4.2. Finally, it is explained how the robustness of a DNN is evaluated in Section 3.4.3.

3.4.1 Retraining DNN

One of the methods that can be used to improve the robustness of a DNN is to retrain the DNN using adversarial examples and the first two approaches are based on this method. The first approach adds a set of adversarial example to the initial training data and completely retrains the DNN which is why it is called the Complete Retraining(CR) approach. This approach is also used in Goodfellow et al. [2014] and Papernot et al. [2016] where it was shown that doing this can help regularize the DNN. This approach does, however, require the training data to be available and as this might not always be the case, an alternative approach was constructed.

This alternative approach retrains the DNN on the adversarial examples with the current weights and biases as the starting point. This starting point is used since most methods, used to train DNNs, are gradient descent algorithms, which means they will start by looking for solutions relatively close to the starting point. Thus, this approach could be seen as a gradient descent method with a warm start. In this approach the training data is not used during the retraining which is why it is called the Partial Retraining(PR) approach. It can, however, be evaluated if the original inputs, that the adversarial examples were created from, should be included or not.

3.4.2 Input Perturbation

During the training process certain input values(pixels) can be made more important than they actually are, due to the overfitting that is incurred by the complexity of DNNs. This could cause the classification to change when one of these pixels is even slightly altered. A concrete example of this can be seen in Figure 4 in Section 4.1, where it is shown that the classification can be changed from two to seven by changing a small number of pixels. The third approach proposed in this thesis aims to counteract this and is called the Input Perturbation(IP) approach.

This approach is inspired by the weight mask and the dropout operation proposed by Song et al. [2018] that were used to create adversarial examples by detecting features that are important in the classification and either weighing them or leaving them out. This raised the question if there exists an operation that can act as an inverse to these operations. This operation would attempt to reduce the importance of input values that could be weaknesses of the DNN by perturbing the input. This operation is the foundation of the IP approach as this approach attempts to improve the robustness by finding a perturbation that could act as this operation.

The perturbation consists of a set of weights and a set of disturbances with every pixel in the input having a corresponding weight and disturbance. The intensity value of this pixel is then multiplied by the weight, after which the disturbance is added to it. The resulting value is then used as the input for the DNN. Since it is desirable to find a perturbation that is applicable to as many inputs as possible, the training data used to create the perturbation consists of multiple inputs. During this creation process weights and disturbances are found that maximize the number of correctly specified inputs in the training data. One way to implement this creation process is by modifying the MILP formulation given in Section 3.2.

The first change that needs to be made is to replace x_j^0 with $p_j\bar{x}_j + q_j$, where $p_j \in \mathbb{R}^+$ and $q_j \in \mathbb{R}$ are, respectively, the weight and the disturbance factor of neuron j and $\bar{x}_j$ is the intensity value of the jth pixel of an image that is known. The weight and disturbance are unbounded since it might be necessary to make one of the pixels more important by inflating its value through a weight or disturbance greater than, respectively, one and zero. An index $h \in H$ also needs to be added to every decision variable, where H is the training set containing multiple images. This h indicates which image the input is from and allows the model to be solved for multiple inputs, with d^h being the digit the input is an image of. Two new decision variables are also added to count the number of correctly classified inputs. The variable y^h corresponds to the maximum activation of the neurons in the output layer of the DNN for input h. Additionally, a binary variable t_j^h is added that is equal to 1 if input h is correctly classified as digit $j-1$, i.e. the activation of the jth neuron in the output layer is the largest and $d^h = j-1$, and 0 otherwise. This is imposed by using the parameter τ_j^h that is equal to 1 if $j = d^h + 1$ and 0 otherwise. Finally, a parameter a_j^h, $j \in J_K, h \in H$, is added that is equal to 1 for $j = d^h + 1$ and 1.2 otherwise. This parameter is used to ensure that the activation of the neuron corresponding to the correct classification is at least 20% larger than the activation of the other classifications, as is done when creating the adversarial examples. These modifications and additions result in the following model:

$$
\begin{aligned}
&\text{maximize} && \sum_{j\in J_{h,K}}\sum_{h\in H} t_j^h && && \text{(10a)}\\
&\text{subject to} && x_j^{h,0} = p_j\bar{x}_j^h + q_j && ,h\in H, j\in J_0, && \text{(10b)}\\
& && \sum_{i\in J_{h,k-1}} w_{ij}^{k-1}x_{ij}^{h,k-1} + b_j^{k-1} = x_j^{h,k} - s_j^{h,k} && ,h\in H, k\in L, j\in J_k, && \text{(10c)}\\
& && x_j^{h,k} \le M^+(1-z_j^{h,k}) && ,h\in H, k\in L, j\in J_k, && \text{(10d)}\\
& && s_j^{h,k} \le M^- z_j^{h,k} && ,h\in H, k\in L, j\in J_k, && \text{(10e)}\\
& && lb_j^{h,k} \le x_j^{h,k} \le ub_j^{h,k} && ,h\in H, k\in L\cup\{0\}, j\in J_k, && \text{(10f)}\\
& && \overline{lb}_j^{h,k} \le s_j^{h,k} \le \overline{ub}_j^{h,k} && ,h\in H, k\in L, j\in J_k, && \text{(10g)}\\
& && y^h \ge a_j^h x_j^{h,K} && ,h\in H, j\in J_K, && \text{(10h)}\\
& && y^h - M_y(1-t_j^h) \le x_j^{h,K} && ,h\in H, j\in J_K, && \text{(10i)}\\
& && y^h \ge 0.01 && ,h\in H, && \text{(10j)}\\
& && t_j^h \le \tau_j^h && ,h\in H, j\in J_K, && \text{(10k)}\\
& && x_j^{h,k}, s_j^{h,k}, y^h \ge 0 && ,h\in H, k\in L, j\in J_k, && \text{(10l)}\\
& && x_j^{h,0} \in \mathbb{R} && ,h\in H, j\in J_0, && \text{(10m)}\\
& && z_j^{h,k} \in \{0,1\} && ,h\in H, k\in L, j\in J_k, && \text{(10n)}\\
& && p_j \ge 0 && ,j\in J_0, && \text{(10o)}\\
& && q_j \in \mathbb{R} && ,j\in J_0, && \text{(10p)}\\
& && t_j^h \in \{0,1\} && ,h\in H, j\in J_K && \text{(10q)}
\end{aligned}
$$

The objective function (10a) maximizes the number of correctly classified samples. The constraints (10b) impose the previously stated replacement of x_j^0 for all inputs h. Constraints (10c) - (10g) remain unchanged from the original MILP formulation. Constraints (10h) and (10i) impose that y^h is exactly the maximum value of the neurons in the output layer if the input is classified correctly and that it is at least 20% larger than the activation of the other classifications. If this is the case, constraints (10i) also set t_j^h to 1 indicating a correct classification, using the constant M_y that is an upper bound for the value of y^h. (10j) puts a lower bound on the activation of the neuron corresponding to the correct classification of input h and ensures h is actually classified as d^h. Constraints (10k) ensure t_j^h can only be set to 1 if input h is of class j. Finally, (10m) - (10q) are the domain constraints for the decision variables.

These additions do, however, add additional big-M constraints to the formulation, which might have a negative effect on how solvable the model is and its performance. This could be avoided by reformulating the objective function such that big-M constraints are not necessary. However, these big-M constraints are not the only part of the model that makes it more complex than the model presented in Section 3.2. The inclusion of the h index and the added decision variables also add to the complexity, as the total number of decision variables increases. This increase can be reduced by leaving out the x variables corresponding to the input layer and directly performing the substitution from (10b) in (10c). Since the input layer contains the largest number of variables, this could drastically reduce the number of decision variables. While this could make the model easier to solve, it would still be more complex than the base model.

This approach can also be performed in two different ways: The first perturbation is found for all inputs, the General Perturbation(GP). This perturbation aims to find the general weaknesses of the DNN that affect the classification of all digits. However, due to the complexity of the model it might be quite hard to create this perturbation for a large set of training data.

For this reason, the second perturbation approach, the Conditional Perturbation(CP), consists of a perturbation for every possible classification. In this case an input would first be classified regularly, after which the perturbation corresponding to the resulting classification is used to perturb the input. This perturbed input is then classified again to test if the initial classification was correct, as is shown in Figure 2. To find the perturbations for the CP the training data can be separated based on their unperturbed classification. The model in (10a) - (10q) is then solved for all subsets of images to obtain the perturbation. This would allow for more training data to be used, since the data set is split up into subsets which could result in more weaknesses being found.

The process of creating these perturbations is shown in the flowchart in Figure 1. In this flowchart it can be seen that original images as well as adversarial examples are used to create the perturbations which is done in an attempt to maintain the correct classification of original images even when they are perturbed.

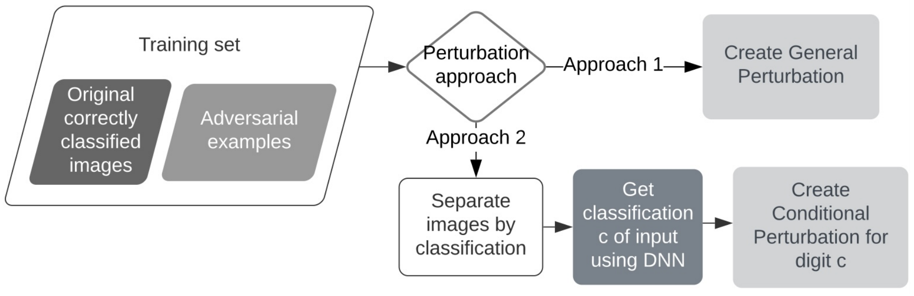

Training set
Original correctly classified images
Adversarial examples
Perturbation approach
Approach 1
Create General Perturbation
Approach 2
Separate images by classification
Get classification c of input using DNN
Create Conditional Perturbation for digit c

Figure 1: Flowchart showing the process used to create the perturbations

3.4.3 Evaluating robustness

In order to evaluate the performance of the three proposed approaches their robustness will be compared. This robustness is made up of two factors: the defensive strength against adversarial examples and the degree of overfitting. This comparison will be made both between the approaches and to a base case, where the initial weights and biases are used on an unperturbed input. Both factors will be evaluated using the accuracy on a set of test data where the percentage of correctly classified inputs will be measured. This data set consists of several adversarial examples as well as several images that are correctly classified by the original DNN. These unaltered images are used to measure if the proposed approaches have a negative effect on the classification of regular inputs. To test the reduction in overfitting, several images that are incorrectly classified by the DNN are also added to the test set, in order to test if the approaches cause these errors to be corrected. The flowchart in Figure 2 illustrates how this test set is used for the two perturbation approaches. Another measure that will be taken into account measures the defensive strength against adversarial examples of a DNN by evaluating the number of pixels that is perturbed to create an adversarial example, the adversarial distance. To some extent this was also done by Papernot et al. [2016], however, they did not specifically use it as a measure of the defense against adversarial examples.

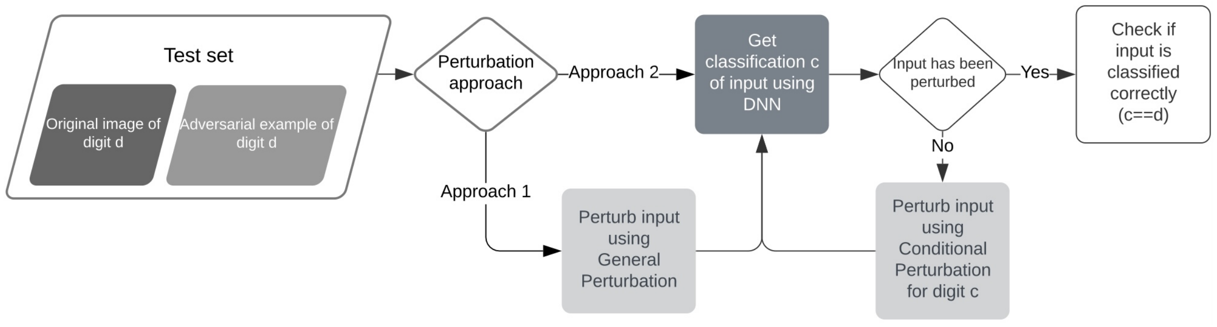

Test set
Original image of digit d
Adversarial example of digit d
Perturbation approach
Approach 2
Approach 1
Get classification c of input using DNN
Input has been perturbed
Yes
No
Check if input is classified correctly (c==d)
Perturb input using General Perturbation
Perturb input using Conditional Perturbation for digit c

Figure 2: Flowchart showing the process used to test the accuracy of the perturbation approaches

4 Results

In this section the results, obtained using the methods described in Section 3, are reported. To evaluate the performance of these methods, they are applied to the five DNNs mentioned in Section 3.1, as is done in Fischetti and Jo [2018]. These DNNs are trained using the Adam algorithm, proposed by Kingma and Ba [2014], from the Python library Keras[Chollet et al., 2015]. For DNN1, DNN2 and DNN4 a test set accuracy between 92% and 95% was achieved, which is similar to the accuracy achieved in Fischetti and Jo [2018]. However, for DNN3 and

DNN5 a lower accuracy of, respectively, 84% and 86% was obtained, which could affect the results. The results are obtained using the MILP solver IBM ILOG CPLEX 20.1.0[Cplex, 2020] on a standard 4-core notebook equipped with a Intel I5 @ 1.6GHz and 8GB RAM.

In the rest of this section the results are presented starting with the results that are replicated from Fischetti and Jo [2018]. In Section 4.1 the feature visualization and the crafted adversarial examples are discussed, while Section 4.2 contains a comparison of performance of the models discussed in Section 3.2. Afterwards, the performance of the robustness improvement methods, proposed in Section 3.4, are presented in 4.3.

4.1 Feature visualization and adversarial examples

As is done in Fischetti and Jo [2018], the method discussed in 3.3 can be used for feature visualization and to create adversarial examples. Figure 3 shows the input that maximizes the activation of the neuron in the output layer that corresponds to the classification of the digit two for DNN1. It can be seen that no visual pattern resembling the human perception of the digit two can be recognized in the figure, as was the case in Fischetti and Jo [2018].

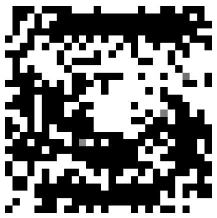
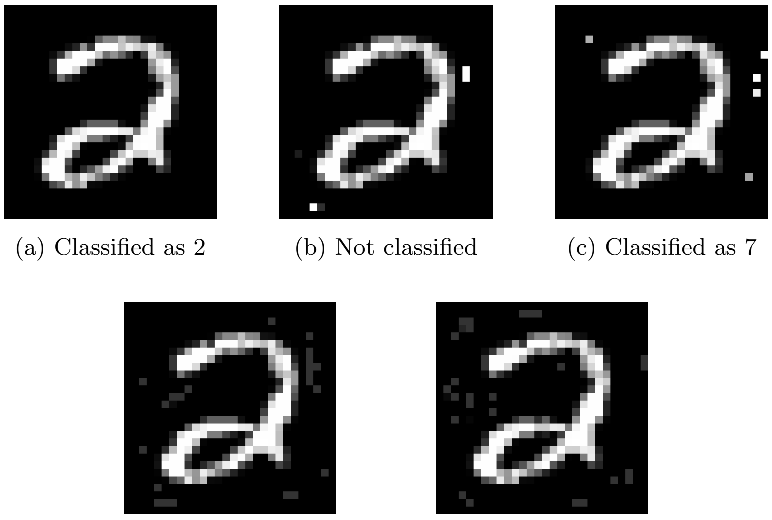


Figure 3: Input that maximizes the activation of the classification neuron corresponding to the digit 2

Several adversarial examples obtained using the methods described in Section 3.3 are shown in Figure 4. Figure 4a shows an unaltered image of the digit two that is correctly classified by DNN1 and Figure 4b shows the adversarial example for this image together with its classification, that is obtained using the MILP formulation. After analyzing the output layer of several adversarial examples it became apparent that it is possible for all the neurons in the output layer to have an output of zero which means the DNN is not able to classify the input. This is feasible since, as is mentioned in Section 3.3, the only constraint that is added to impose the target class is the constraint in (8), which does not ensure that the output of the neuron corresponding to the target digit is strictly greater than zero.

To prevent this from occurring, a lower bound could be set for the output of the neuron corresponding to the target digit. For the adversarial example in Figure 4c this lower bound is set to 0.01 which results in the DNN classifying this example as a seven, as opposed to the example in Figure 4b that is not classified by the DNN. It is visible that the adversarial example with the lower bound requires more deviation than the one without this restriction. This shows that, in some cases, it is apparently cheaper to set all outputs in the output layer to zero than it is to impose the classification of the target class. Additionally, Figure 4d shows an adversarial example with an upper bound of 0.2 on the deviation, which was mentioned in Section 3.3 as well. For the creation of this figure no lower bound was used, however, for the adversarial example in Figure 4e this lower bound was imposed.


(a) Classified as 2 (b) Not classified (c) Classified as 7

(d) Not classified($\Delta = 0.2$) (e) Classified as 7($\Delta = 0.2$)

Figure 4: Adversarial examples of handwritten digits with their classification

4.2 Performance

The performance of creating these adversarial examples using the MILP formulation is evaluated as well. In order to do this, for each of the five DNNs mentioned in Section 3.1, an adversarial example is created for 100 unique inputs that are correctly classified by the corresponding DNN. An average is then taken of these runs to obtain the statistics that are reported in this section. This is done for the base model and the two models with the tightened bounds described in Section 3.2. A time limit of 300 seconds is imposed for the creation of each individual adversarial example, as is done in Fischetti and Jo [2018].

Several performance measures for these three models are reported in Table 2. The "Solved" column shows the percentage of times the model could be optimally solved within the time limit. The "Gap" column represents the optimality gap between the best found feasible solution and the highest lower bound. The "Nodes" column corresponds to the number of branching nodes that were needed to solve the model. Additionally, the "Presolve" and the "Time" column represent the time it took to compute the tightened bounds and solve the model respectively. For the "Time" column the instances that exceeded the time limit are included in the average and the "Presolve" column is omitted for the base model as no tightened bounds are used.

Table 2: Performance of the three models for the five DNNs

	Base model				Improved model					Weaker improved model				
	Solved (%)	Gap (%)	Nodes	Time (s)	Solved (%)	Gap (%)	Nodes	Presolve (s)	Time (s)	Solved (%)	Gap (%)	Nodes	Presolve (s)	Time (s)
DNN1	100	0.00	1 868	0.75	100	0.00	895	6.00	0.62	100	0.00	895	5.94	0.66
DNN2	100	0.00	15 392	7.87	100	0.00	5 626	12.93	3.97	100	0.00	5 626	12.91	3.93
DNN3	95	0.99	33 501	68.16	100	0.00	9 036	129.32	14.28	100	0.00	9 059	49.03	15.84
DNN4	46	27.55	90 766	209.06	96	0.85	30 195	325.93	67.16	99	0.35	28 644	71.11	65.71
DNN5	15	66.36	185 604	280.19	55	19.39	97 401	543.04	192.69	58	17.84	104 993	81.60	186.38

It can be seen in the table that the MILP can be solved for the two smaller DNNs quite efficiently, as the model could be solved to optimality for all adversarial examples for all models. For the three largest DNNs the improved model can be solved to optimality within the time limit in more cases than the base model, which is in line with the results from Fischetti and Jo [2018]. This is especially clear for DNN4 and DNN5, as the percentage of optimally solved instances increases by 20% and 40% respectively. Additionally, it can be seen that the running time decreases quite substantially for all five DNNs when comparing the improved and weaker improved model to the base model. Besides the running time, the average optimality gap and the number of nodes also decrease drastically. It can even be seen that these values are more than halved when using the improved model opposed to the base model.

Additionally, the weaker improved model appears to perform just as well, if not better, than the improved model. In Table 2 it can be seen that the weaker improved model even slightly outperforms the improved model for the two largest DNNs, as the percentage of optimally solved instances increases slightly. Average running time, optimality gap and number of nodes all decrease to some degree as well, even though the bounds are not as tight. This slight increase in performance could, however, be caused by the randomness in the branching decisions of CPLEX as the difference between the two models is not very large.

As is done in Fischetti and Jo [2018], this test is also performed with a tolerated optimality gap of 1%, which means the algorithm is terminated when this gap is reached. The same performance measures that are shown in Table 2, are presented for the case with this tolerated gap in Table 3. It can again be seen that the performance is increased quite significantly when using the improved or weaker improved model. It is also interesting to see that the improved model outperforms the weaker improved model for DNN4 in this case, in contrary to the results in Table 2. This supports the earlier stated assumption that the difference between these model might be caused by randomness. Another interesting observation that can be made is that the number of optimally solved instances is higher without the optimality gap for the weaker improved model for DNN4 and the improved model for DNN5. A possible explanation for this could be the randomness in the running time, as it might be possible that there are several instances that are solved very close to the time limit of 300 seconds.

Table 3: Performance of the three models for the five DNNs with a tolerable optimality gap of 1%

	Base model				Improved model					Weaker improved model				
	Solved (%)	Gap (%)	Nodes	Time (s)	Solved (%)	Gap (%)	Nodes	Presolve (s)	Time (s)	Solved (%)	Gap (%)	Nodes	Presolve (s)	Time (s)
DNN1	100	0.55	1 859	0.76	100	0.53	900	6.25	0.67	100	0.53	900	5.97	0.65
DNN2	100	0.80	16 551	7.68	100	0.82	5 554	12.85	3.82	100	0.82	5 554	13.08	3.81
DNN3	96	1.78	41 190	67.70	100	0.83	7 376	129.87	14.31	100	0.82	7 715	48.95	14.82
DNN4	46	28.92	115 874	211.17	97	1.63	29 033	323.36	66.45	96	2.31	33 910	71.36	67.12
DNN5	16	65.25	198 940	274.39	54	18.93	101 210	544.96	189.19	59	16.34	100 052	83.10	186.73

4.3 Robustness improvement

In the following subsection the results of the robustness improvement methods, proposed in Section 3.4, are presented. However, due to time constraints these results are only presented for DNN1. For the training of these methods a training set of 450 adversarial examples is used consisting of all possible source-target pairs as well as the 50 original, correctly classified images that these examples were created from. In order to test the accuracy, a test set of 900 adversarial images, 900 original, correctly classified images and 200 original, incorrectly classified images is used. First these data sets are used to evaluate the retraining approaches in Section 4.3.1. Afterwards, a similar evaluation is presented for the input perturbation method in Section 4.3.2. Finally, in Section 4.3.3 a comparison is made between the three methods.

4.3.1 Retraining approaches

In Table 4 the test set accuracy that is obtained when applying the retraining methods is presented. For the Complete Retraining(CR) approach only the 450 adversarial examples from the training set are added to the training set that was used to originally train the DNN, since it already contains original images. However, as was mentioned in Section 3.4.1, the Partial Retraining(PR) approach is performed using a training set of either only adversarial examples or both adversarial examples and original images. For the CR approach 50 epochs are used, as is done during the original training process. However, for the PR approach 10 epochs are used since the training set is smaller for this approach. In the table the accuracy is split up into the accuracy that is obtained when testing on the adversarial examples in the test set and the accuracy of tests on the original, correctly and incorrectly classified images.

Table 4: Test set accuracy of retraining approaches

	Adv. exmpls. accuracy (%)	Original image accuracy (%)	Original incorrect accuracy (%)	Overall accuracy (%)
Complete Retraining	90.78	96.67	33.50	87.70
Partial Retraining:				
- Only adv. exmpls.	77.33	97.78	10.00	79.80
- Adv. exmpls. + original	79.11	98.11	10.00	80.75

In the table it can be seen that both approaches perform quite well on the adversarial examples. The table also shows that the accuracy on original images is not affected significantly, as both approaches have an accuracy above 96% on these images. It can also be seen that several images that were incorrectly classified by the original DNN are now correctly classified, which suggests that the retraining process might have slightly reduced overfitting. The CR approach does appear to perform the best among the three methods, but it is interesting to see that the PR approach results in a slightly higher accuracy on the original images, even when the DNN is only retrained on adversarial examples. Among the two versions of the PR approach, the approach with the original images seems to be slightly better, although the difference is small.

4.3.2 Input perturbation

While implementing the Input Perturbation(IP) approach it became apparent that the model could only be solved for a maximum of around 30 images, as the running time would increase quite drastically when using a larger set of training data. Because of this the General Perturbation(GP) was performed using 20 adversarial examples and 10 original images. While choosing these images it was ensured at least one image of every classification was added. For the Conditional Perturbation(CP), 18 adversarial examples and 5 original images were used as training data for the creation of the perturbation of each individual digit $c = 0, \ldots, 9$. The adversarial examples were selected such that c was the target classification and every other digit was the source for two of the examples. Additionally, to reduce running time the bound tightening method from the improved model is used, however, since the input can also be negative after perturbation the interval of the input values was defined as [-1,1]. Several test runs were also performed using wider intervals, however, this resulted in significantly longer running times which is why this interval was chosen.

Table 5 shows the accuracy on the test set when the two IP approaches are applied on the original DNN. This accuracy is presented in the same way as it is done in Table 4, with the running time needed to create the perturbation being added. For the CP this running time is the total time needed to create all 10 perturbations corresponding to the different classifications. Additionally, a variation on the perturbation approach is added that attempts to minimize the distance between the original and the perturbed input while still maximizing the number of correctly classified images. In order to do this, the absolute sum of the distance between the weights and one ($\sum_{j\in J_0}|p_j-1|$) and the disturbance and zero ($\sum_{j\in J_0}|q_j|$) is subtracted from the objective. This sum is first multiplied by 0.001 to ensure that the correct classification of images is still prioritised. These sums are shown in the table as well, with the sums for the CP being the average over the 10 perturbations.

Table 5: Test set accuracy of perturbation approaches

	Adv. exmpls. accuracy (%)	Original image accuracy (%)	Original incorrect accuracy (%)	Overall accuracy (%)	Time (s)	$\sum_{j\in J_0}\|p_j-1\|$	$\sum_{j\in J_0}\|q_j\|$
General Perturbation	19.44	21.33	13.50	19.70	273.92	8.54×10^{10}	1.68×10^{12}
- Min. distance	51.89	98.78	8.50	68.65	14 152.32	2.66	0.36
Conditional Perturbation	7.89	27.56	8.00	16.75	11 123.02	1.08×10^{6}	1.68×10^{12}

The table shows that, while the perturbations help in correctly classifying several adversarial examples, the accuracy on original images decreases quite drastically when applying both IP approaches. It appears that the GP performs better than the CP as the overall accuracy is higher and the running time is lower in this case. Due to time constraints the minimum distance variation was therefore only applied to the GP. It can be seen that the absolute sum of the distance is reduced drastically when applying this variation and that it results in a significant improvement, as the accuracy on all sets, except on the original, incorrectly classified images, more than doubles. This could be an indication that the perturbation is overfit on the training data if the distance is not minimized, which is why the test set accuracy is a lot lower. This is supported by the fact that the sum of the distance is also relatively high for the CP, which achieves a low accuracy as well. However, the increase in accuracy when imposing the minimum distance does come with a significant increase in running time, as creating the minimum distance perturbation takes more than 50 times longer than creating the regular GP. It is also interesting to see that the running time for the creation of the 10 perturbations for the CP is significantly higher than 10 times the running time for the GP. A possible reason for this could be that the CP has to analyze weaknesses more in depth, since more adversarial examples of each specific digit are used that can each show different weaknesses.

4.3.3 Comparison

In Table 6 a comparison is made between the best variations of the three proposed approaches and the original DNN without input perturbation. For the PR approach the variant with the original images in the training set is used and for the IP approach the minimum distance variation of the GP is used. The same accuracy measures are presented as in the previous tables, however, an extra measure is added, namely the adversarial distance. This distance is obtained by creating an adversarial example of 100 original images that are correctly classified by all approaches. An average is then taken of the distance that is needed to create these examples. More details on the creation of these examples can be found in Appendix B.

Table 6: Comparison of the three proposed approaches and the original DNN

	Adv. exmpls. accuracy (%)	Original image accuracy (%)	Original incorrect accuracy (%)	Overall accuracy (%)	Adv. distance
Original DNN	0.00	100.00	0.00	45.00	2.04
Complete Retraining	90.78	96.67	33.50	87.70	2.01
Partial Retraining	79.11	98.11	10.00	80.75	2.65
Input Perturbation	51.89	98.78	8.50	68.65	2.17

In the table it can be seen that all three approaches improve the overall accuracy compared to the original DNN. Additionally, the table shows that the CR approach achieves the best accuracy, however, the PR approach is not much worse. While the IP approach shows an improvement over the original DNN it still does not perform quite as well as the two retraining methods. It also seems that the three approaches reduce the overfitting to some extent as they all succeed in classifying several originally incorrectly classified images, with the CR approach being especially successful in this aspect. An interesting observation that can be made is that the adversarial distance decreases when the CR approach is applied, while this distance increases for the two other approaches. This could be an indication that this retraining process creates new weaknesses for the DNN that can be exploited while fixing the old weaknesses.

5 Conclusion

In this thesis three approaches were evaluated that attempt to improve the defensive strength against adversarial examples and reduce the overfitting of a DNN. A comparison between these three approaches was made to answer the following research question: "*To what extent can adversarial examples improve the robustness of Deep Neural Networks?*". Two of these approaches involve retraining the DNN using a set of adversarial examples, while the third approach creates an input perturbation using the MILP formulation from Fischetti and Jo [2018]. The approaches are applied to a DNN that classifies images from the public MNIST data set.

The results showed that the Complete Retraining(CR) approach, that was also used in existing literature, achieves the best improvement in robustness, as it achieves the highest overall accuracy at 87.7% as well as the highest accuracy on originally incorrectly images at 33.5%. This could be an indication that this method reduces the measure of overfitting the most effectively among the three approaches. This approach also achieves an accuracy of 90.78% on adversarial examples. It could, however, be the case that this approach creates new weaknesses in the DNN while repairing the old weaknesses, as it did not become harder to create adversarial examples. It should also be noted that the Partial Retraining(PR) approach is quite close to the CR approach in terms of overall accuracy and accuracy on adversarial examples. This suggests that, even if the original training set is not available, retraining the DNN on adversarial examples can improve the robustness of the DNN. This approach also makes creating new adversarial examples the hardest, as more changes to the input need to be made to create new adversarial examples.

While the Input Perturbation(IP) approach does not perform as well as the retraining methods, it does provide an improvement over the original DNN without perturbation, with 51.89% of the adversarial examples being correctly classified when the perturbation is applied. This shows that the approach has potential and that it can definitely be a viable option if, for some reason, the DNN can not be altered. Furthermore, all three approaches do not seem to compromise the accuracy on original images, as more than 96% of these images are still correctly classified after applying the approach. Finally, while the PR and IP approaches are able to correctly classify a number of originally incorrectly classified images, this number is a lot lower than for the CR approach, which implies they might not help as much in reducing the measure of overfitting.

This thesis shows that the robustness of a DNN can be improved using adversarial examples even if the original training data is not available or the DNN can not be altered. These results are in line with the existing literature where it has been stated that the CR approach can regularize a DNN. A new approach, namely the IP approach is also introduced in this thesis that seems to be able to improve the robustness to some extent as well.

The results on the robustness improvement were, however, only evaluated for the smallest DNN which raises the question if the conclusion would be the same if larger DNNs were evaluated. The training set for the retraining approaches was also quite small at 450 adversarial examples compared to the existing literature, e.g. Papernot et al. [2016] where 18,000 adversarial examples were used. Additionally, it was not evaluated if applying the bound tightening method for the input perturbation had any effect on the feasibility and the correctness of the formulation.

This thesis could be a nice starting point for a more in depth analysis of the IP approach as much more can be done with it. A good next step would be reducing the running time needed to create the perturbations. This would make it possible to create perturbations with a larger set of training data which might improve the performance of this approach. The trade-off between perturbation and robustness could also be evaluated more efficiently, which could be interesting since the perturbation with minimum distance showed significant improvement over the regular perturbation. To this end, it might be interesting to analyze perturbations where the weights and disturbances are bounded. Finally, it could also be interesting to evaluate the Conditional Perturbation with minimum distance to see if a similar improvement could be achieved.

References

Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-Gómez, and Domenico Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65, 12 2016. doi: 10.1007/s10589-016-9847-8.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. 04 2017.

François Chollet et al. Keras. `https://keras.io`, 2015.

IBM ILOG Cplex. V20.1.0: User's manual for cplex. *International Business Machines Corporation*, 2020.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23, 07 2018. doi: 10.1007/s10601-018-9285-6.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv 1412.6572*, 12 2014.

Kishor Datta Gupta, Dipankar Dasgupta, and Zahid Akhtar. Determining sequence of image processing technique (ipt) to detect adversarial attacks. 07 2020.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998. doi: 10.1109/5.726791.

Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. 05 2017.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. pages 372–387, 03 2016. doi: 10.1109/EuroSP.2016.36.

Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. Deflecting adversarial attacks with pixel deflection. 01 2018.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. 11 2017.

Yibing Song, Chao Ma, Xiaohe Wu, Lijun Gong, Linchao Bao, Wangmeng Zuo, Chunhua Shen, Rynson Lau, and Ming-Hsuan Yang. Vital: Visual tracking via adversarial learning. pages 8990–8999, 06 2018. doi: 10.1109/CVPR.2018.00937.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 12 2013.

Vincent Tjeng. *Evaluating robustness of neural networks with mixed integer programming*. PhD thesis, 01 2018.

Appendix A Partial perturbation

Several other variations on the input perturbation were attempted. However, due to time constraints these variations could again only be applied to one of the two perturbation approaches, as was the case for the minimum distance variation. Since the General Perturbation(GP) appeared to be the best this approach was chosen. In Table 7 the accuracy and running time for these variations are compared to the regular GP. First it was evaluated if it is possible to perturb the input using either only the weights or only the disturbances. However, performing the perturbation using only disturbances was apparently quite difficult, as no feasible solution could be found by CPLEX after 10 hours. Because of this, the perturbation using only disturbances was created with only 10 adversarial examples and no original images. However, CPLEX was still not able to solve this to optimality within 12 hours with 8 of the 10 adversarial examples being correctly classified after these 12 hours. Additionally, the minimum distance variation is also included in Table 7 as well as the absolute sums of the weights and disturbances for all variations.

Table 7: Test set accuracy of variations of perturbation approaches

	Adv. exmpls. accuracy (%)	Original image accuracy (%)	Original incorrect accuracy (%)	Overall accuracy (%)	Time (s)	$\sum_{j \in J_0} \lvert p_j - 1 \rvert$	$\sum_{j \in J_0} \lvert q_j \rvert$
General Perturbation	19.44	21.33	13.50	19.70	273.92	8.54×10^{10}	1.68×10^{12}
Only weights(p_j)	17.11	21.67	11.50	18.60	40.52	1.30×10^{11}	0.00
Only disturbances(q_j)	0.00	0.00	0.00	0.00	43 202.67	0.00	1.68×10^{12}
Min. distance	51.89	98.78	8.50	68.65	14 152.32	2.66	0.36

It can be seen in the table that the perturbation using only the weights can be created in only 40 seconds which is an 85.21% decrease compared to the running time of Perturbation 1. Despite this drastic decrease in running time, the accuracy does not seem to decrease as severely which suggests it might be worthwhile to use only the weights. This claim is further supported by the distance measures of the perturbation with the minimum distance as the sum of the absolute disturbance is quite close to zero. This also explains why it was more challenging to create a perturbation using only the disturbances as they appear to be less helpful. Additionally, the table shows that the perturbation with only disturbances is unable to classify a single image correctly which further supports this claim.

Appendix B Adversarial examples after improvement

To test if applying the accuracy improvement approaches would make it harder to create new adversarial examples, a similar performance test to the one in Section 4.2 was performed. A set of 100 images was used, that were correctly classified by all approaches and were all unaltered(no adversarial examples). The same method to determine the target digit was used as in Section 4.2. The bounds were tightened for these tests as well with the input interval for the run of the IP approach being set to [0, 1.5] as the perturbation made it possible to have an input value greater than one.

Table 8: Performance of adversarial example creation after applying the three improvement approaches

	Solved (%)	Gap (%)	Nodes	Time (s)	Adv. distance
Original DNN	100	0.00	888	0.66	2.04
Complete Retraining	100	0.00	753	0.56	2.01
Partial Retraining	100	0.00	855	0.67	2.65
Input Perturbation	100	0.00	841	0.73	2.17

Surprisingly the original DNN required the most branching nodes, however, this could be due to the randomness in the branching decisions. It is interesting to see the running time is in line with the adversarial distance as the approach with lowest running time has the lowest distance and the approach with the highest time has the highest distance. This suggests that there might be a correlation between the two which could be interesting to do more research on. It is again interesting to see that the creation of adversarial examples on the CR approach is the easiest in all aspects. As this method did achieve the best improvement in robustness.

Appendix C   Visualization of minimum distance perturbation

In Figure 5 the resulting image after applying the weights of the minimum distance perturbation on an input of only ones is shown. The pixel intensities are on an interval between 0.52 and 1 but for this visualization they have been normalized between 0 and 1. In this figure white corresponds to weights with the value 1 and black pixels correspond to weights with the value 0.52. This means the darker pixels are weighed less in the perturbation and they can be seen as potential weaknesses for the DNN.

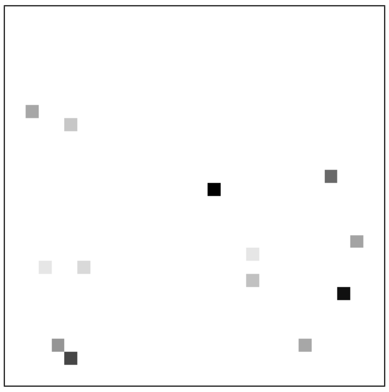


Figure 5: Image of input of ones after applying weights normalized between 0 and 1

In Figure 6 the resulting image after applying the disturbances of the minimum distance perturbation on an input of only zeros is shown. The pixel intensities are on an interval between 0 and 0.12 but they have again been normalized. This visualization is harder to interpret but it is clear to see that very little disturbance is applied in this perturbation.

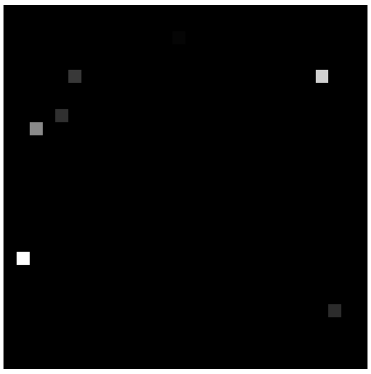


Figure 6: Image of input of zeros after adding disturbances normalized between 0 and 1

In Figure 7 the resulting image after applying the complete minimum distance perturbation on an input of only ones is shown. The pixel intensities are on an interval between 0.52 and 1.12 but they have yet again been normalized between 0 and 1. Here it can be seen that most pixels remain untouched by the perturbation. There are however several lighter pixels that are given more importance since they might make classification easier for the DNN. Additionally, like in

Figure 5, there are a number of darker pixels that can be interpreted as potential weaknesses of the DNN.

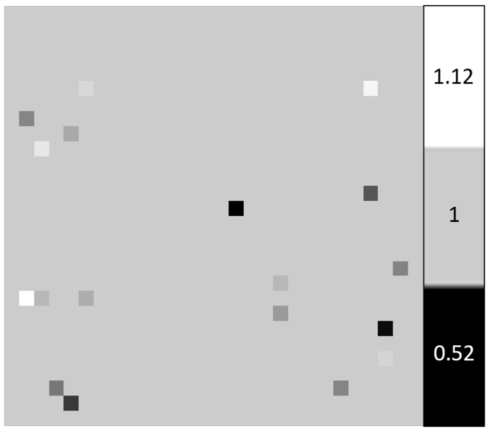
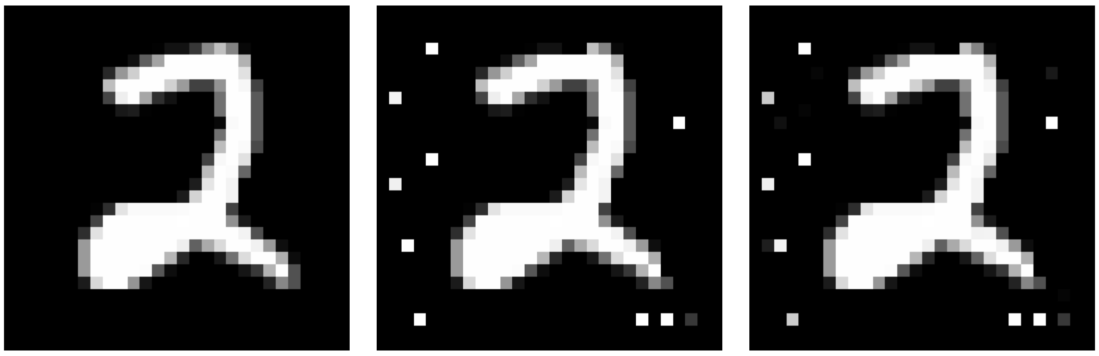


Figure 7: Image of input of ones after perturbation normalized between 0 and 1

Figure 8 shows an adversarial example that is correctly classified after applying the perturbation. The figure shows an image of the digit two, an adversarial example of this image that is classified as a one and the perturbed adversarial example that is correctly classified as a two. It can be seen that the perturbation does not revert the changes that are made to create the adversarial example and instead slightly alters several other pixels that apparently counteract the adversarial changes. As a result, the perturbation does not seem to work as an inverse to these adversarial changes, but rather as a completely independent operation. This could, however, be due to the distance minimization that is imposed as it might be less expensive to slightly change several pixels that counteract the changes than to completely invert them.



(a) Original image of the digit 2

(b) Adversarial example classified as 1

(c) Perturbed adversarial example classified as 2

Figure 8: Example of correctly classified perturbed adversarial example

Appendix D File description for code

The following java files can be found in the code:

- `DNN.java`: Class used to model a Deep Neural Network(DNN) containing all layers
- `Layer.java`: Class used to model a layer of the DNN containing the weights and biases
- `Main.java`: Class used to run the performance tests on the five DNNs
- `MainadvAfterImpr.java`: Class used to run the performance test after applying the improvement approaches
- `MainAdvExmplSet.java`: Class used to create the training and test set of adversarial examples
- `MainPerturbation.java`: Class used to create the perturbation
- `MainVisualize.java`: Class used to create the feature visualization
- `MILPAdvAfterImpr.java`: Class containing CPLEX implementation of MILP formulation used to create adversarial examples after applying the improvement approaches
- `MILPAdversarial.java`: Class containing CPLEX implementation of MILP formulation used to create adversarial examples
- `MILPBounds.java`: Class containing CPLEX implementation of MILP formulation used to tighten bounds
- `MILPPerturbation.java`: Class containing CPLEX implementation of MILP formulation used to create perturbation
- `MILPVisualize.java`: Class containing CPLEX implementation of MILP formulation used for feature visualization

The following python files can be found in the code:

- `CreateImage.py`: Script used to turn pixel values from .csv file into .png images
- `DNN.py`: Class used to train the DNN and write weights and data
- `DNNPretrained.py`: Class that can load pre-trained weights and biases and test its accuracy and perform Partial Retraining
- `main.py`: Script used to train the five DNNs and write the weights and data

- `mainRetrain.py`: Script used to test the accuracy of the retraining approaches
- `mainTestPerturb.py`: Script used to test the accuracy of the perturbations
- `writeData.py`: Script used to write training and test data