ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics
Bachelor Thesis Econometrie en Operationele Research

Retraining deep neural networks using mixed integer linear optimization

Abstract

In this paper Deep Neural Networks (DNNs) are considered. The DNNs we train provide us parameters we can use to make a 0-1 Mixed Integer Linear Program (MILP) for this network. By using the flexibilty of this 0-1 MILP we can improve the 0-1 MILP by including bounds, visualize hidden neurons of the DNN and most importantly, create so called adversarial examples. Those adversarial examples are then used to retrain the network to check if this method improves robustness of the network. In our case, the method we use seems not that effective. Moreover, computational results are taken into account in the entire research to discuss on the practical considerations and limits of the 0-1 MILP.

E.C.M. Koot
Student ID number: 453767
Supervisor: B.T.C. Van Rossum
Second assessor: T.A.B. Dollevoet

Date final version: 29 August 2021

The views stated in this thesis are those of the author and not necessarily those of the supervisor, second assessor, Erasmus School of Economics or Erasmus University Rotterdam

1 Introduction

Neural networks are nowadays a popular field of research and machine learning overall seems to be more visible in everyday life. A well-known example of a neural network is pattern recognition, or face recognition to be more specific. The research field of self-driving cars is quite known for using pattern recognition, since nowadays cars already have neural network based assistance with for example road detection. An adaptive real-time road detection system based on neural networks was already studied by Foedisch and Takeuchi back in 2004. Currently, researchers even apply neural networks to let cars drive fully autonomously as described by Bojarski et al. (2016).

Neural networks can be utilized in a wide range of research fields, that are less common and visible for most people, in health care for example. In 2002, Sordo reported how neural networks could help making clinical diagnosis and later in 2009, Temurtas et al. did the same for diagnosing diabetes. The performance of these methods is proper, but questions arise about reliability, since these models are hard to interpret and understand. As Dayhoff and DeLeo (2001) pointed out, the neural networks are often considered to be black boxes. This might invoke suspicion, which is often very undesirable.

In this paper we study neural networks that are modeled as a 0-1 Mixed Integer Linear Program (0-1 MILP). We will utilize the model as introduced by Fischetti and Jo (2018). As indicated earlier, the difficulties with interpreting a neural network can interfere with utilization of neural networks. Therefore Erhan et al. (2009) started to visualize higher-layer features of a Deep Neural Network (DNN), because visualization might play an important role in enhancing interpretability. One of the applications of the 0-1 MILP in this paper is therefore feature visualization.

Furthermore, we use the 0-1 MILP to produce so called “adversarial examples”, as introduced by Szegedy et al. (2013). Adversarial examples can be of great interest to find weaknesses of the proposed DNN model. In this paper, we will also introduce a new application of our constructed adversarial examples. We will retrain the DNN, using a new data set that includes both the original data set and a set of created adversarial examples. We will analyze the new performance of the DNN model with the L_1 norm determining the nearest distance to an adversarial example. In our

applications we will be using the MNIST database on handwritten digits.

In the remainder of this paper we will give relevant background information on familiar studies in Section 2. In Section 3 the methodology is presented after a brief introduction in neural networks. The results are given in Section 4. Finally, we discuss our results and draw a conclusion in Section 5. A small description of the used programming files is given in Appendix A.1

2 Literature Review

Neural networks became a desired field of interest over the last decades, since applications of neural networks such as facial recognition, for example the You Only Look Once (YOLO) system by Redmon et al. (2016), became increasingly visible in everyday life. However, research in this field already had its first substantial breakthrough in 1986 by Rumelhart et al., who introduced back-propagating to train the networks.

Another big milestone in neural network history was the introduction of the AlexNet by Krizhevsky et al. (2012). With the AlexNet, big data sets became remarkably more manageable. Nowadays, computers can dominate the games of chess or Go, as by Silver et al. (2016), with the use of neural networks. The opportunities of these techniques seem limitless, but as mentioned before, in some fields, questions arise.

As Wang et al. (2020) pointed out, Artificial Intelligence (AI) have proved itself useful in health care, but they emphasize the aim to clarify black box models. Wang et al. also want to raise awareness for good pilots and testing before utilization in a clinical environment, and proper monitoring when AI is implemented. Liability issues are not unlikely, since there is a lack of model interpretability.

To improve interpretability of neural networks, visualization of the network features could be useful. Back in 2009, Erhan et al. visualized higher-layer features of a DNN using a geedy ascent method. One problem that may occur using such methods, is getting trapped in a local minimum. Our 0-1 MILP, which is introduced in Section 3 prevents us from getting stuck in a local minimum and is therefore very capable to visualize higher-layer features.

Adversarial examples are also a topic of high interest, after Szegedy et al. (2013) elaborated on them. Those examples are capable of tricking the network to produce an improper outcome with only a slight modification of the input. In 2014, Goodfellow et al. introduced a set of methods to generate adversarial examples. In this paper we address adversarial examples generated by our 0-1 MILP model as done by Fischetti and Jo (2018). The adversarial examples can be used to show weaknesses and test robustness of the DNN. Tjeng et al. (2017) also evaluated robustness of neural networks with mixed integer programming, but used another approach. Our method will be further elaborated in Section 3.5 and could be considered to be more hands on.

3 Methodology

3.1 Neural Network

A neural network is a set of neurons in a network communicating in different layers, in which there is an input and an output layer, and one or more hidden layers. Since we address only networks with multiple hidden layers, we speak of Deep Neural Networks, whereas networks with one hidden layer are often referred to as simple hidden layers. The architecture of a DNN is given in Figure 1, this example has three hidden layers. Neural networks are named after the human brain, where we have an input, like seeing or hearing, and an output like thinking or doing. In our brains all neurons are connected in some way between the input and the output layer, but what exactly happens is still quite unclear. Humans for example are very capable to distinguish cats and dogs, although there is a wide variation within those animal species. So, there is no clear set of visible separators between those two.

In our study we have DNNs consisting of $K+1$ layers where layer 0 is the input layer and layer K is the output layer. Consequently, there are $K-1$ hidden layers. Each layer $k \in \{0, 1, ..., K\}$ consists of n_k nodes, or so called neurons. We define each node $j \in \{0, 1, ..., n_k\}$ in layer k as NEURON(j, k). Let x_j^k be the output of NEURON(j, k). As can been seen in Figure 1, all neurons in the hidden layers are connected with all neurons in the preceding and successive layers. This

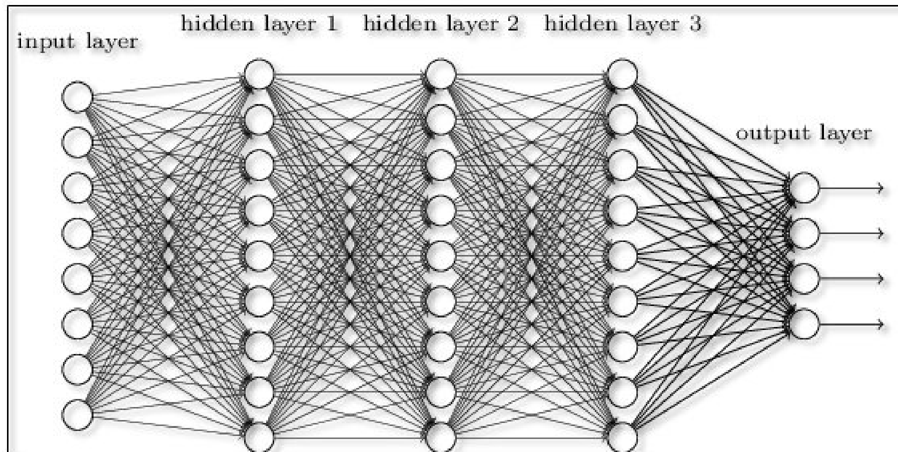


Figure 1: Deep neural network architecture from RSIPVision (2015)

connection is made by an activation function. In this paper we will use the Rectified Linear Unit (ReLU) function, where $ReLU(y) = max\{0, y\}$. With given weight matrix W and a vector of biases b, the following activation is found for the vector of outputs x^k:

$$x^k = ReLU(w^{k-1}x^{k-1} + b^{k-1}), \quad k = 1, ..., K \tag{1}$$

Note that the weights and biases are obtained in the training-phase of the DNN, therefore we can use them as fixed values after the DNN has been trained. We can now write Equation 1 to the following linear conditions:

$$\sum_{j=1}^{n_{k-1}} w_{ij}^{k-1}x_i^{k-1} + b_j^{k-1} = x_j^k - s_j^k, \quad x_j^k \geq 0 \quad s_j^k \geq 0 \quad k = 1...K, j = 1...n_k \tag{2}$$

This way, the positive and negative part of the ReLU are dealt with in one formula, but since there is no unique solution for this equation, i.e. if x_j^k becomes larger, s_j^k can become larger to cancel this out. Therefore we introduce binary variable z_j^k to ensure either x_j^k or s_j^k is equal to zero.

In this paper, we use five different DNNs. An overview can be found in Table 1. All networks will have in input layer of 784 entries for all pixels and an output layer with 10 neurons to classify all digits. The models are trained through using Stochastic Gradient Descent (SGD), using 50

epochs. All models achieved an accuracy higher than 90% on the validation set. The training set consisted of 60000 images and the validation set had a size of 10000. Note that the notation used above as well as the DNNs we use are strongly correlated to Fischetti and Jo (2018).

Table 1: DNN overview

Name	Neurons per hiddden layer
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 0-1 MILP

The full formulation of the 0-1 MILP is given below.

$$\min \sum_{k=0}^{K} \sum_{j=1}^{n_k} c_j^k x_j^k + \sum_{k=1}^{K} \sum_{j=1}^{n_k} (\gamma_j^k z_j^k + \lambda_j^k s_j^k) \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n_{k-1}} w_{ij}^{k-1} x_i^{k-1} + b_j^{k-1} = x_j^k - s_j^k \qquad k = 1...K, j = 1...n_k \tag{3.2}$$

$$x_j^k, s_j^k \geq 0 \qquad k = 1...K, j = 1...n_k \tag{3.3}$$

$$z_j^k \in \mathbb{B} \qquad k = 1...K, j = 1...n_k \tag{3.4}$$

$$z_j^k = 1 \rightarrow x_j^k \leq 0 \qquad k = 1...K, j = 1...n_k \tag{3.5}$$

$$z_j^k = 0 \rightarrow s_j^k \leq 0 \qquad k = 1...K, j = 1...n_k \tag{3.6}$$

$$lb_j^0 \leq x_j^0 \leq ub_j^0 \qquad j = 1...n_0 \tag{3.7}$$

$$lb_j^k \leq x_j^k \leq ub_j^k \qquad k = 1...K, j = 1...n_k \tag{3.8}$$

$$lb_j^k \leq s_j^k \leq ub_j^k \qquad k = 1...K, j = 1...n_k \tag{3.9}$$

Here, Constraints (3.2) ensure the ReLU connectivity in the network, where Constraints (3.3) set the non-negativity of x_j^k and s_j^k. Constraints (3.4) - (3.6), also known as indicator constraints, regulate the 0-1 MILP to get unique solutions. More specific, these constraints result in the fact that if $z_j^k = 1$ then $x_j^k = 0$ and if $z_j^k = 0$ then $s_j^k = 0$ for $k = 1, ..., K, j = 1, ..., n_k$. This ensures that the x_j^k and s_j^k can not cancel each other out. These indicator constraints tend to have very weak relaxations during the Branch and Bound optimization, which is used while solving for this problem. This may lead to a enormous increase in computing time. Namely, the indicator constraints are converted to Big M constraints: $x_j^k \leq M_x(1-z_j^k)$ and $s_j^k \leq M_s z_j^k$. Defining proper upper bounds to the continuous variables x and s can tighten this formulation and consequently reduce computing time and is therefore of great importance. This was already emphasized by Camm et al. back in 1990.

3.2.1 Bounds

Constraints (3.7) set limits for the input layer. Since we use normalized pixels, those lower bounds (lb) and upper bounds (ub) are set as 0 and 1, but those bounds are only convenient for the input layer. To further improve the usability of the model, upper bounds on all neurons in the hidden layers and output layer are necessary. Therefore we come up with a smart pre-processing phase, typically suitable for our DNN instances. Bounds (3.8) and (3.9) are determined with this pre-processing phase, which works as follows. First we use our objective function (3.1) n_1 times to independently maximize activity of all x_j^k with $k = 1$, so for all x_j^k in the first hidden layer. Since there is no connection between neurons within a single layer, we are allowed to calculate these in parallel. The outcomes of this max. activation are the upper bounds for those x variables. Now we do the same for all s_j^k with $k = 1$, this gives us the upper bounds for those s variables of the first hidden layer. Next we add both the upper bounds for x_j^k and s_j^k to the model and repeat those step for the next layer. Repeat this procedure $K - 1$ times, such that you have created, layer by layer, upper bounds for all hidden layers and the output layer. Note that is procedure is possible since the activation of a neuron is only dependent on its predecessors and there is also no connection

between the neurons within a single layer.

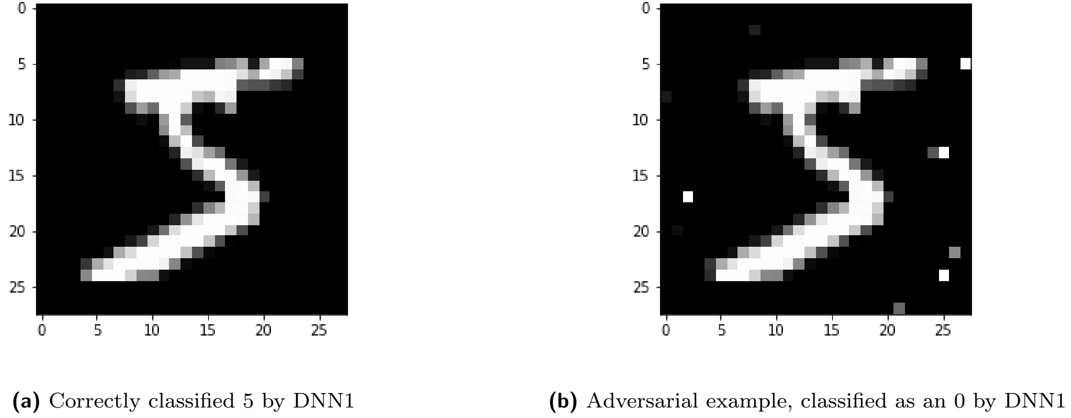
Since this pre-processing phase can be time consuming in bigger models, we introduce a time limit of 1.0 second on the bound of each variable. This way the bounds will be less tight, which most likely results in longer computation times for the 0-1 MILP, but the pre-processing time is acceptable. The importance of these bounds will be investigated while calculating adversarial examples, on which we elaborate later on in Section 3.4. We will compare the differences between the model with or without exact bounds and between the the exact and weaker bounds. Since the bounds will tighten the formulation, feasible solutions are expected to be found earlier and faster within the Branch and Bound procedure, which shall lead to better computation times.

3.3 Feature visualization

The first application of our 0-1 MILP is feature visualization. Since neural networks are often considered to be black boxes, feature visualization can be feasible to get more insight in what a neural network is or does and what the meaning of a certain neuron is. Since we use the MNIST data set of handwritten digits, hidden nodes may represent a part of a digit, such as a horizontal line in a certain section of the image. Besides neurons in hidden layers, also neurons in the output layer can be viewed. By maximizing the activity of a neuron, we generate input that would activate this neuron as much as possible and then investigate whether or not we can identify a pattern.

3.4 Adversarial examples

Now we will discuss adversarial examples and how they are generated by the 0-1 MILP. This application is the core of this research. An example of an adversarial example can be found in Figure 2. Figure 2a is a five that is correctly classified, while in Figure 2b only certain pixels have changed, the DNN classifies this image as a zero. Next, we will explain how we adjust and utilize the 0-1 MILP to produce such examples.



(a) Correctly classified 5 by DNN1

(b) Adversarial example, classified as an 0 by DNN1

Figure 2: Example of adversarial examples

Given a set of correctly classified digits d, we try to modify these pictures as little as possible such that they will be classified wrongly. To do so, we modify each picture in order to make sure that the output will become $\tilde{d} = d + 5 \mod 10$. To make sure the activation of the wrong classification is at least 20% higher than the activation of all other nodes we add the following equations to our 0-1 MILP:

$$x^k_{\tilde{d}} \geq 1.2 x^k_j, \quad j \in \{0, ..., 9\} \setminus \{\tilde{d}\} \tag{4}$$

In order to minimize the number of pixels changed, we define distance variable d_j which must satisfy the following constraints, where $\tilde{x}^0_j$ represents the pixels of the adversarial example and x^0_j the pixels from the original picture:

$$-d_j \leq \tilde{x}^0_j - x^0_j \leq d_j, \quad d_j \geq 0, \quad j = 1, ..., n_0 \tag{5}$$

Next, we introduce a new objective function that minimizes the L_1-norm distance between the original picture and the newly created adversarial example. This new objective becomes:

$$\min \sum_{j=1}^{n_0} d_j \tag{6}$$

Finally, we consider the case were pixels can not be changed by more than 0.2 (on a scale from 0 to 1). This can easily be done by adding the restriction $d_j \leq 0.2$ for $j = 1, ..., n_0$ to the 0-1 MILP.

We use the calculation times and the number of nodes in the Branch and Bound tree of the adversarial examples to investigate the importance of the bounds of our 0-1 MILP. While producing those adversarial examples, we set a time limit of 5 minutes on all runs. Furthermore, we will check the performance when we use weaker bounds on our model. Finally, we investigate the case where we use a 1% optimality gap as a stopping criteria for our 0-1 MILP. This way we ensure to have solutions within 1% range of optimality, so that the quality of the solutions does not suffer a lot, but hopefully the computation time decreases.

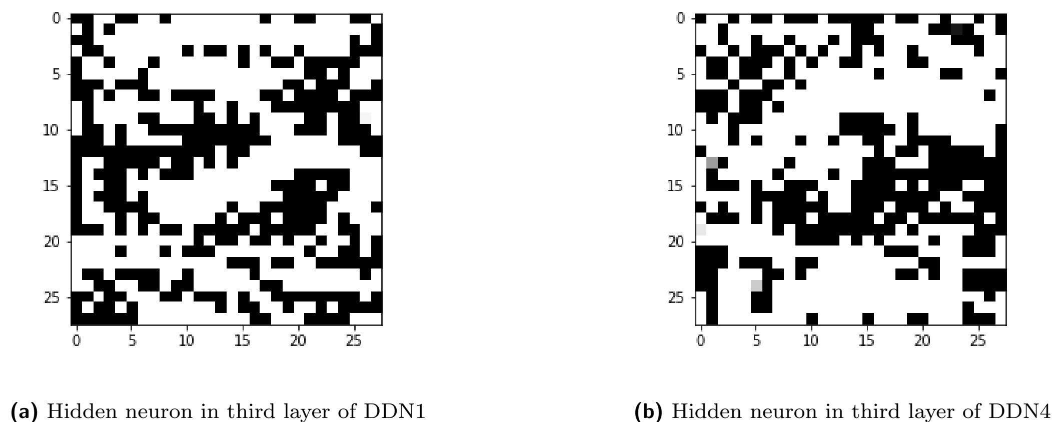
3.5 Improving the DNN using adversarial examples

In this section we will use the adversarial examples to try to improve our DNN. Using the previously described methods to create adversarial examples, we now produce 4 sets of adversarial examples. These sets have a size of 5 up to 20 percent of the size of the original data set. In our case this means we create sets of 3000, 6000, 9000 and 12000 adversarial examples. After doing so, we will add them to the original data set on which the DNN was trained, with the correct output. Now will we train a new network based on the extended data set. Since weak spots are addressed by the adversarial examples, we expect the new DNN to be more robust. The robustness will be measured by the L_1-distance norm to create new adversarial examples using the new DNN. So after we trained our new DNNs with extended data sets, again with an accuracy higher than 90%, we let them produce 100 adversarial example each to calculate the average distance. Due to computational reasons, this is examined with the DNN1 model.

4 Results

4.1 Feature visualisation

After visualizing several neurons in our networks no visual patterns could be found. Two examples of those features can be found in Figure 3. These examples show that visualizing the hidden nodes does not provide us the help we searched for in understanding and interpreting DNNs.



(a) Hidden neuron in third layer of DDN1

(b) Hidden neuron in third layer of DDN4

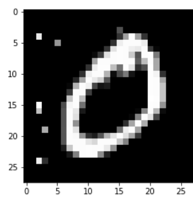
Figure 3: Two examples of feature visualisation

4.2 Adversarial examples

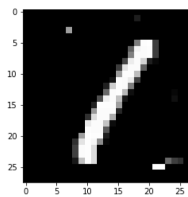
Figure 4 shows us ten adversarial examples made by the 0-1 MILP based on DNN3. As can be seen, in some cases, for example in Figures 4b and 4j, only a few pixels need to be changed to trick the DNN. Where in other cases such as in Figures 4d and 4f, clearly more pixels need to be modified in order to create an adversarial example. Overall the 0-1 MILP is very capable of producing adversarial examples, but the amount of pixels that needs to be changed seems somewhat case dependent.

In Figure 5 the adversarial examples are presented, where eacht pixel could only be modified by 0.2 (on a scale from 0 to 1, where 0 corresponds with a black pixel and 1 with a white pixel). The main difference between Figure 4 and Figure 5 is clearly the fact that in the latter, many more

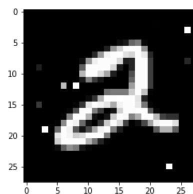
pixels need to be modified, which expected since each pixel was only allowed to be less modified. To further clarify this difference, we can view Figure 6. In this figure the pixels that are modified are shown on. The upper four pictures represent the pixel changes when there are no limitations to the modification of each pixel and the bottom four pictures represent the pixel changes for the same images where each pixel is only allowed to change by 0.2. To improve readability of the bottom four pictures, the pixel changes are rescaled such that if a pixel changes by the maximum of 0.2, it is a black pixel. Figure 6 clearly illustrates that more pixels need to change when adding a restriction on the allowed change per pixel.



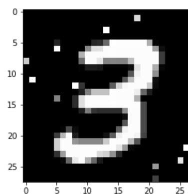
(a) Classified as 5



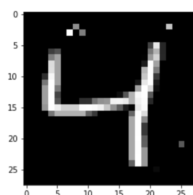
(b) Classified as 6



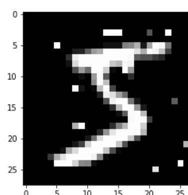
(c) Classified as 7



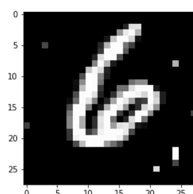
(d) Classified as 8



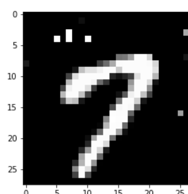
(e) Classified as 9



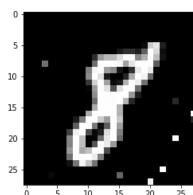
(f) Classified as 0



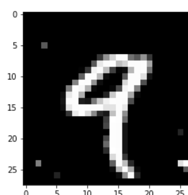
(g) Classified as 1



(h) Classified as 2

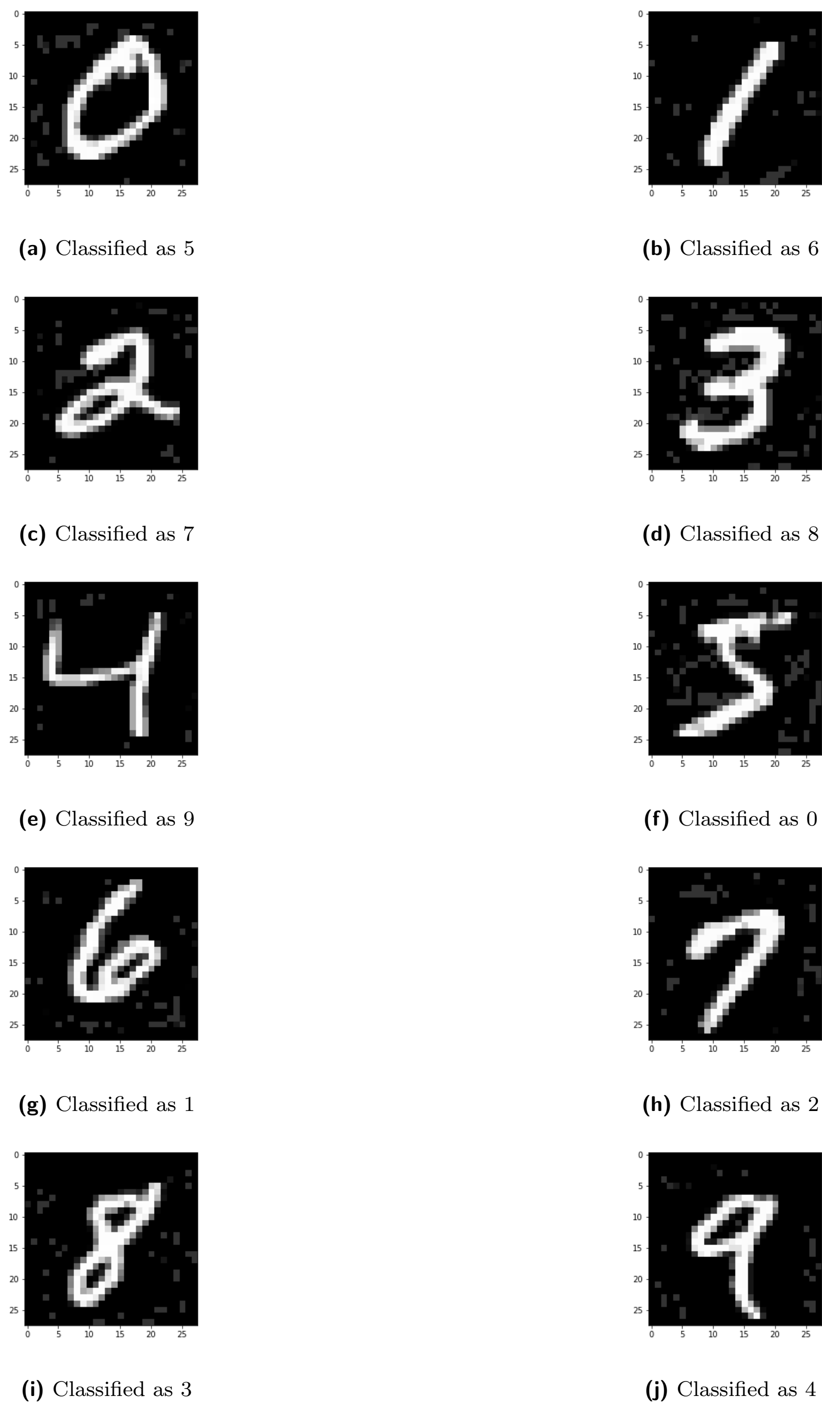


(i) Classified as 3



(j) Classified as 4

Figure 4: Adversarial examples from DNN3



(a) Classified as 5

(b) Classified as 6

(c) Classified as 7

(d) Classified as 8

(e) Classified as 9

(f) Classified as 0

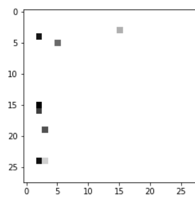
(g) Classified as 1

(h) Classified as 2

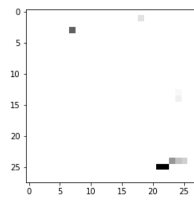
(i) Classified as 3

(j) Classified as 4

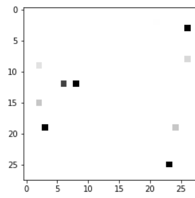
Figure 5: Adversarial examples from DNN3 with $d_j \leq 0.2$



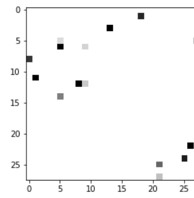
(a) Classified as 5



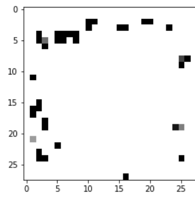
(b) Classified as 6



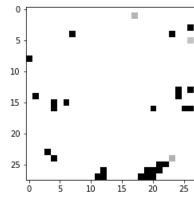
(c) Classified as 7



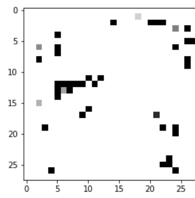
(d) Classified as 8



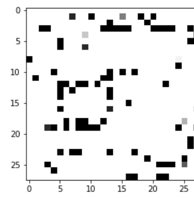
(e) Classified as 5



(f) Classified as 6



(g) Classified as 7



(h) Classified as 8

Figure 6: Scaled pixel changes without (a-d) and with $d_j \leq 0.2$ (e-h)

4.3 Importance of bounds

The results of the DNNs producing adversarial examples with and without exact bounds can be found in Table 2. We look at the percentage of solved cases (%solved), the average gap to optimality (%gap), the average number of Nodes in the Branch and Bound tree en the average Time in seconds needed to calculate the adversarial examples. The importance of bounds can be seen in all models. The number of nodes and the computing time of the examples clearly shrink.

Table 2: Results on DNN performances with and without exact bounds with 5 minute time limit

	Basic Model				Improved Model			
	%solved	%gap	Nodes	Time (s)	%solved	%gap	Nodes	Time (s)
DNN1	100	0.0	228	0.3	100	0.0	6	0.3
DNN2	100	0.0	30211	20.8	100	0.0	1214	0.9
DNN3	97	0.5	153144	94.8	100	0.0	16961	10.2
DNN4	69	11.2	413533	182.4	100	0.0	72046	39.2
DNN5	0	87.2	405668	294.1	45	18.5	567199	75.0

In Table 3 the time of pre-processing (t.pre.) is given for all models, this is the time to calculate the bounds. We see that the weak bounds clearly are faster to determine, especially when the models become more comprehensive. In all cases the weaker bounds perform quite good, but in DNN5 we see that exact bounds prove their value, since including those to the model raises the percentage of solved cases by 16 percentage points.

Table 3: Results of weak bounds compared to exact bounds, both with 5 minute time limit

	Improved model									
	Exact bounds					Weaker bounds				
	t.pre. (s)	%sol.	%gap	Nodes	Time (s)	t.pre. (s)	%sol.	%gap	Nodes	Time (s)
DNN1	30	100	0.0	6	0.3	30	100	0.0	3	0.3
DNN2	169	100	0.0	1214	0.9	106	100	0.0	1274	0.9
DNN3	765	100	0.0	16961	10.2	91	100	0.0	22755	13.7
DNN4	1845	100	0.0	72046	39.2	92	100	0.0	85107	35.8
DNN5	10096	45	18.5	567199	75.0	154	29	26.3	561593	257.4

In Table 4 the results of the DNN performances in creating adversarial examples using a 1% optimality gap can be found. The %sol. column is in this table replaced by the number of times the time limit is reached (#timlim), since we do not actually solve the instances when using gaps. Once more, the importance of bounds is proven by this table, both the number of times the time limit is reached as well as the computation time support this. Furthermore if we compare, we can notice some minor differences between the performances when we solve to optimality or with a 1% gap, but only one big difference stands out. In DDN5, 33 more cases of the problem were solved within a 1% gap than when solving to optimality. Hence, gaps might be helpful for large instances.

Table 4: DNN performances with 1% gap on optimal solution with 5 minute time limit, weaker bounds are used on DNN4 and DNN5

	Basic Model				Improved (weaker bounds)			
	#timlim	Time (s)	Nodes	%gap	#timlim	Time (s)	Nodes	%gap
DNN1	0	0.3	228	0.1	0	0.3	6	0.0
DNN2	0	20.0	30311	0.9	0	0.9	1208	0.6
DNN3	0	55.5	155108	1.3	0	10.6	17326	0.8
DNN4	30	173.2	426505	10.7	0	35.2	83481	0.9
DNN5	100	296.3	268495	90.8	22	269.8	411978	31.0

4.4 Improving the DNN using adversarial examples

In Table 5 the results of adding adversarial examples to DNN1 can be found. No significant differences or trends can be found by adding adversarial examples to the data set. This might be the result of using the most simple DNN we have, namely DNN1. The fact that all cases with added adversarial examples seem to slightly perform worse concerning robustness than the original model might suggest that the added adversarial examples mainly focus on the same limited set of pixels. If this is truly the case, adding adversarial examples repeatingly might solve for this issue.

Table 5: Results of adding adversarial examples to the data set on the DNN-norm distance

	Original dataset	Size added			
		5%	10%	15%	20%
Mean(distance)	6.11513645	5.067735	4.706737	5.667457	4.477385
St.Dev(distance)	3.758042328	3.365881	1.771677	3.097223	2.055303

5 Conclusion

In this paper we investigated a 0-1 MILP model based on DNNs. We have seen that for smaller DNNs these MILPs can be solved to optimality, but computational problems occur when larger networks are considered. Even though the importance of adding bounds to the 0-1 MILP is undisputed, bounds can not always help out with computational limit in bigger networks. Furthermore, we noticed that feature visualisation is not really helpful in our case, perhaps in other data sets or in larger models a better pattern can be recognized.

Creating adversarial examples with the 0-1 MILP performs well for smaller networks but has some computational difficulties with larger instances. The adversarial examples can be useful to point out weaknesses of the DNN, but adding them to a data set to create a new network did not improve the robustness of the network. This might be because of over fitting on those models or might possibly work with bigger models. Also adding and retraining iteratively could be considered, as this might address different important pixels each iteration.

Further research might also address the computational problems that occur by using other bigger networks. Furthermore, since retraining was not that effective on the robustness of our DNNs, using adversarial examples where pixels can only be changed with a certain percentage gives better results, since it is likely that more pixels will have to change in that case. Experimenting with different gap sizes for calculating bounds as well for solving bigger models might be very insightful since computational issues still seems to be the greatest challenge for the 0-1 MILP.

References

M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

J. D. Camm, A. S. Raturi, and S. Tsubakitani. Cutting big m down to size. *Interfaces*, 20(5): 61–66, 1990.

J. E. Dayhoff and J. M. DeLeo. Artificial neural networks: opening the black box. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 91(S8):1615–1635, 2001.

D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

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

M. Foedisch and A. Takeuchi. Adaptive real-time road detection using neural networks. In *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No. 04TH8749)*, pages 167–172. IEEE, 2004.

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

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

RSIPVision. Exploring deep learning and cnns deep learning and convolutional neural networks: Rsip vision blogs. `https://rsipvision.com/exploring-deep-learning/`, 2015. Accessed: 2021-08-27.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

M. Sordo. Introduction to neural networks in healthcare. *Open Clinical: Knowledge Management for Medical Care*, 2002.

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

H. Temurtas, N. Yumusak, and F. Temurtas. A comparative study on diabetes disease diagnosis using neural networks. *Expert Systems with applications*, 36(4):8610–8615, 2009.

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

F. Wang, R. Kaushal, and D. Khullar. Should health care demand interpretable artificial intelligence or accept "black box" medicine?, 2020.

A Appendix

A.1 Code explanation

Writing this thesis, 5 programming files are used. First there are two python files, 'create_dnn.py' is used to load the MNIST data set, create the DNN, export the weights and biases and export a list of correctly classified pictures that can be used later on for generating adversarial examples. This file also contains a method which is used later on to repeat the process, but then with the extended data set. The second python file is 'plot_mnist_from_cplex.py'. This file is used to plot neurons and adversarial examples after the 0-1 MILP returns a list of pixels. The three java programs are very similar and all contain the 0-1 MILP solver but used for different purposes, to prevent an overload of parameters each time we use the 0-1 MILP differently, 3 copies were made. First of all, 'MILP.java' is used to calculate the bounds per layer for either the x or s variables, and export those. Next, 'MILPadvex.java' is used to solve the model for a hundred adversarial examples and export the time, gap, and number of nodes. 'MILPadvexprinter' is used to export a large set of adversarial examples, used to retrain the model.