Deep learning and Neural Networks

Assignment 1

Sjoerd Hermes
Anish Kisoentewari
Thomas Lieber

Universiteit Leiden
Dr. W.J. Kowalczyk
10-03-2020

Contents

1 Task 1: Analyze distances between images

One of the first steps in making the simple distance based classifier is finding the number of points n_i, images in this case, per cloud C_i i in $\{0, 1, ..., 9\}$. The number of points per cloud for this data set are:

- C_0 has 319 points
- C_1 has 252 points
- C_2 has 202 points
- C_3 has 131 points
- C_4 has 122 points
- C_5 has 88 points
- C_6 has 151 points
- C_7 has 166 points
- C_8 has 144 points
- C_9 has 132 points

The main part of the classifier consists of finding the center c_d of each cloud C_d d in $\{0, 1, ..., 9\}$. To find the center, which is a 256 dimensional vector of each cloud, all images of each digit were taken as one such cloud. Subsequently, the mean of each column (256 in total) was calculated per cloud, consisting of n_i points, as stated above. This gives 10×256 dimensional vectors, one representing each cloud.

After calculating the center of each cloud, the radius had to be found, which is defined as the biggest distance between the center of C_d and points from C_d. Here, first the distances were calculated between the center of each cloud and all the images belonging to that cloud, using the euclidean distance. Here the output is a 256 dimensional vector, where each column represents the euclidean distance between the center of a cloud and an image. With all the distances calculated, the biggest one for each distance was selected, such that the function returned a 10 x 256 data-frame, where the rows are the digits and the columns the dimensions.

Finally, using the centers calculated before, finding the distances between each cloud is a straightforward implementation whereby can be concluded that, based on the distances between the clouds, images of digit 7 and digit 9 seem to be most difficult to separate due to the lowest value for the distance between the centers of these digits, see Figure 1. Despite this, the overall accuracy is expected to be high seeing that the value for $dist_{ij}(c_i, c_i)$ is either 0, or near 0, whereas the value for $dist_{ij}(c_i, c_j)$, where $i \neq j$, is a lot higher.

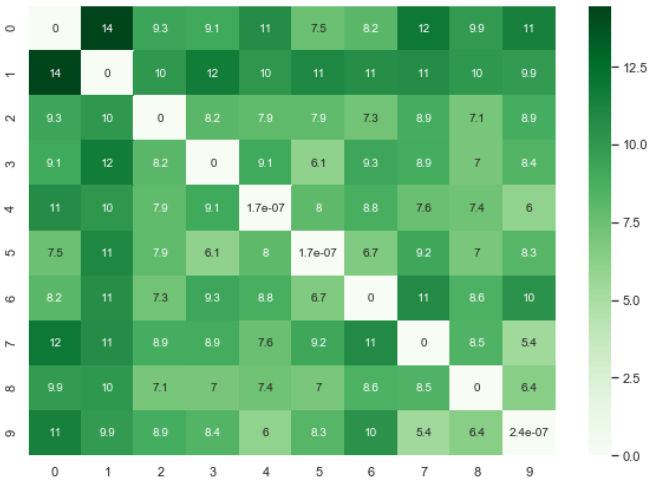

Figure 1: Distance matrix.

2 Task 2: Implement and evaluate the simplest classifier

Using the distance based classifier from task 1, two confusion matrices for the train and test set respectively are presented in order to easily identify difficult to separate classes.

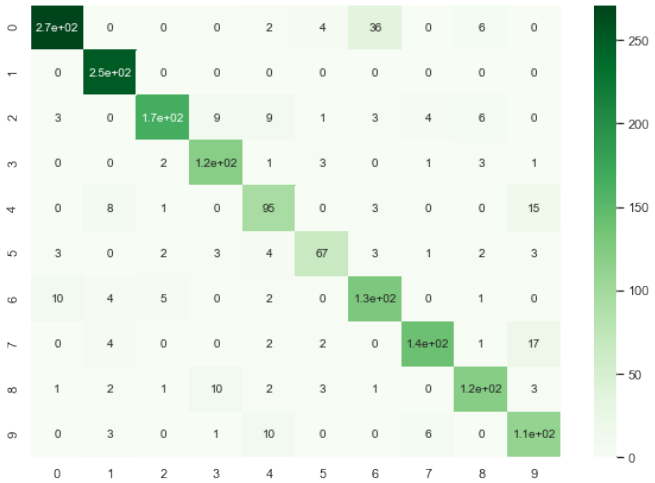

Figure 2: Confusion matrix for the train set.

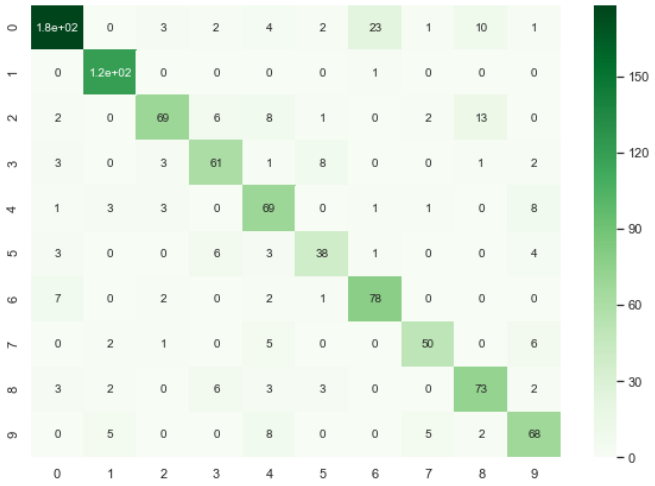

Figure 3: Confusion matrix for the test set.

Based on these two confusion matrices, it seems that classes 0 and 6 are difficult to separate, which is different from the answer given in task 1: 7 and 9.

Where the initial distance based classifier worked with the euclidian distance, using the scikit-learn package, several other distance measures were used in order to see if the error in either the train, test or both data sets would decrease. However, this was not the case. The newer distance measures proved even worse, especially the Manhattan distance.

3 Task 3: Implement a multi-class perceptron algorithm

The weights were initialized by making a matrix $W_{257\times10}$ filled with random numbers. The vector with the predicted output $\hat{y}$ was calculated by taking the product of training input matrix $T_{1707\times257}$ and W. Then, the strongest output node was found by taking the max of each row of TW. The weights were then updated by looping over the predicted vector.
Firstly, the loop checked if an element i in the predicted vector was not equal to the same index for the output vector. If this was the case then all the over-activated nodes of W were updated by subtracting row(s)

$$W_{[:,overactivatedweights]} := W_{[:,overactivatedweights]} - T_{[i,:]}$$

Where $T_{[i,:]}$ is the input of the wrongly identified image and $W_{[:,overactivatedweights]}$ the weights of the nodes that were activated more than the node that it should have been. Further, the weights of the correct node were updated by

$$W_{[:,correctnodeweights]} := W_{[:,correctnodeweights]} + T_{[i,:]}$$

This whole process counted as one iteration of the algorithm.
The method was tested by letting the algorithm converge on the training inputs and see how fast it converged. This was done 100 times, choosing different weights each time. On average, it converged within 391 iterations ($std = 77.97$ and average time was 8.4 seconds. The test accuracy was on average 0.84 ($std = 0.0068$)
The algorithm was also tested using the following equation to update the weights.

$$W_{[:,\hat{y}_i]} := W_{[:,\hat{y}_i]} + \eta(y_i - \hat{y}_i)T_{[i,:]}$$

Where $W_{[:,\hat{y}_i]}$ is the to be updated weight and η is the learning rate, which is 1 in this case. This method converged fairly quickly but had a low accuracy with mean accuracy 0.15. The test accuracy about the same. This could be due to the fact that we did something wrong, or this equation is not suited for this specific problem. Therefore the obvious better choice for updating the weights is the first method described here.

4 Task 4: Linear Separability

Cover's theorem states that a complex pattern-classification problem cast in a high dimensional space non-linearly is more likely to be linearly separable than in a low dimensional space (Cover, 1965)[1]. If the number of points N in a d-dimensional space is smaller than $2 \times d$, then they are almost always linearly separable. However, if the number of points in a d-dimensional space is bigger than $2 \times d$ then they are almost always non linearly separable. In this task $d = 256$, so $2 \times d = 512$. Therefore it becomes possible with the knowledge from task 1 (the number of points n_i, per cloud C_i i in $\{0, 1, ..., 9\}$, to check if they are linearly separable using this theorem. Therefore, all pairs except $(0, 1)$ and $(0, 2)$ are linearly separable, as those pairs have 571 and 521 points respectively, such that for both pairs it holds that $N > 2 \times d$.

Another way of seeing if pairs of digits are linearly seperable is using the algorithm created in task 3. The algorithm was applied to each pair of digits i, j in $\{0, 1, ..., 9\}$ where $i \neq j$. This was done by first finding the indices of a pair ij. Then, only uses these indices for the training input matrix T. The algorithm converged fairly quickly to 1, taking either 1 to max 115 iterations, with mean test accuracy being 0.96. To check whether images of i could be separated from all remaining images, a similar approach was used as described for the first part of this task. Firstly, finding the indices equal to i then setting i to 1 and all other indices not equal to i 0. Then, applying this to all pairs of images i and image set $\{0, 1, ..., 9\} \setminus \{i\}$ it was discovered that all pairs of images were linearly separable. It took about 694 iterations max and minimum of 108 iterations. The mean test error was 0.84 $std = 0.17$. So it was possible to linearly separate all images of digit i from the set of all images $\{0, 1, ..., 9\} \setminus \{i\}$.

5 Task 5: Implement the XOR network and the Gradient Descent Algorithm

After implementing the MLP with one hidden layer and back-propagation (applied only after each activation of the whole training set), the Mean squared error (MSE) and classification error rate was measured per iteration. In each iteration the XOR network classified the inputs $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$, and used an error function to compare the outputs of the XOR network with the desired output vector $(0,1,1,0)^T$. The following graphs display the MSE and classification error rate for three different activation functions (Sigmoid, Relu and Tanh), three different levels of the learning rate (0.10, 0.50 and 0.90) and two random sets of weight initialization (uniform between 0,1 and standard normal) of the same seed. Each sets of graph displays the monitored metrics for a different activation function.

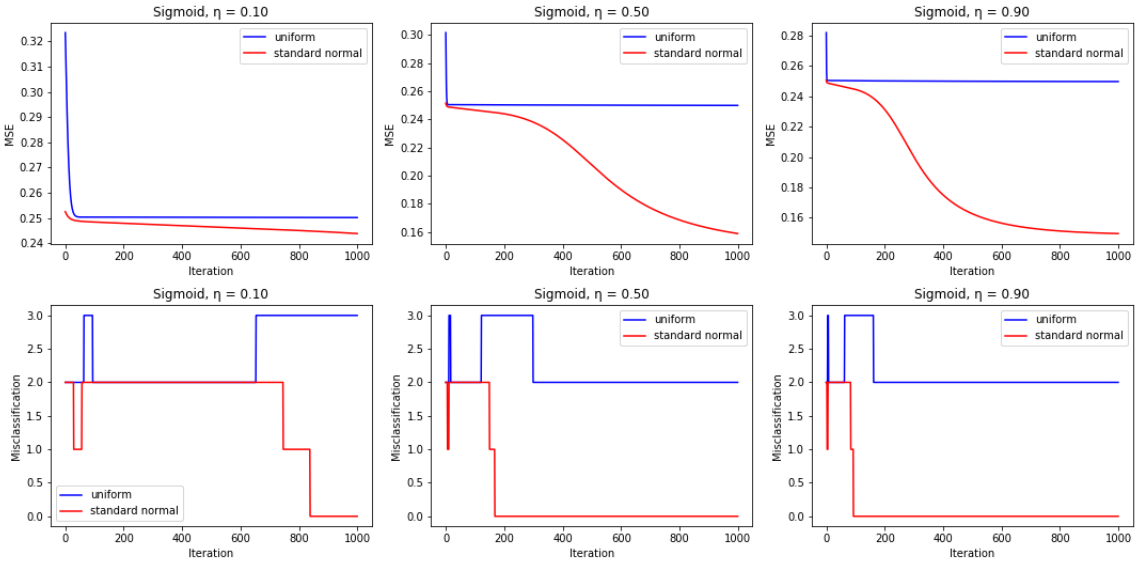

Figure 4: Sigmoid activation for MSE and classification error on learning rates 0.10, 0.50 and 0.90.

The sigmoid activation function is the most suitable function for the XOR network, particularly with a standard normal instigated set of weights and a high learning rate. Note that in the upper row of graphs the y-axes are not the same. The iteration with the learning rate of 0.90 clearly produces the lowest MSE for the network the fastest.

The uniform instigated set of weights do not include negative values for weights initially, while this is desired to correctly classify for the XOR truth table. This may explain why the standard normal consistently outperforms the uniform initialisation on these metrics, as it does include initial negative weights.

Regarding the classification error rate in line with the 0.5 boundary imposed in the assignment, the graphs look very similar. The learning rate again serves to speed up this learned behaviour of classification for the two sets of randomly instigated weights. Keeping in mind the 0.5 boundary rule, the outputs of the XOR network tend to centre towards 0.5, sometimes slightly lower, sometimes slightly higher. While the MSE lowers, the more discrete classification error rate may fluctuate for higher number of iterations due to this behaviour of the network.

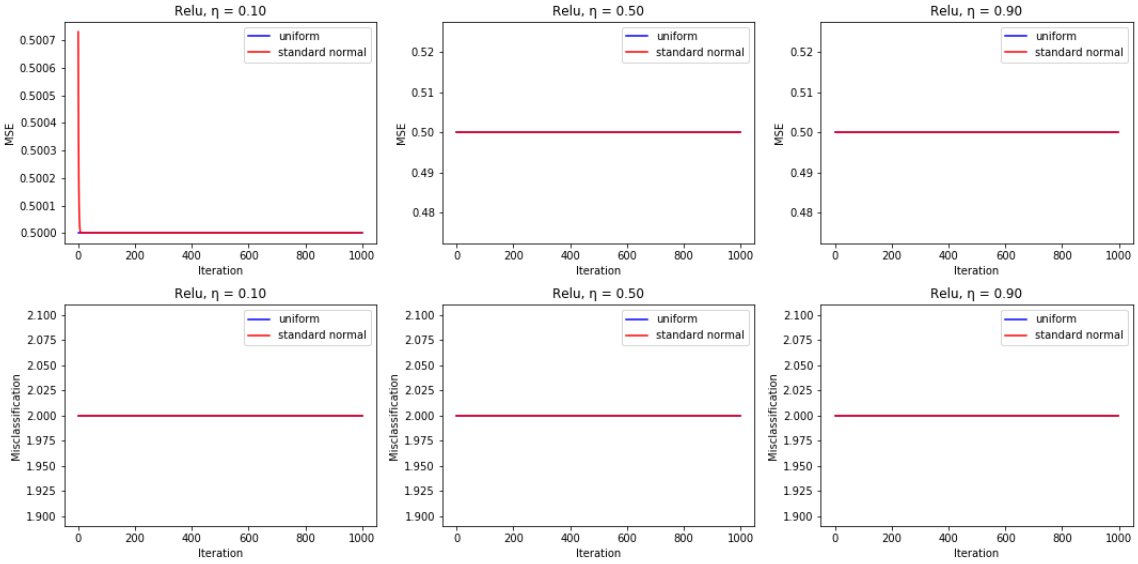

Figure 5: Relu activation for MSE and classification error on learning rates 0.10, 0.50 and 0.90.

The relu activation function performs the worst on the XOR network. It allows for an output of the network greater than 1, while the output of the network should be between 0 and 1. Thus the network does not improve itself on any of the metrics.

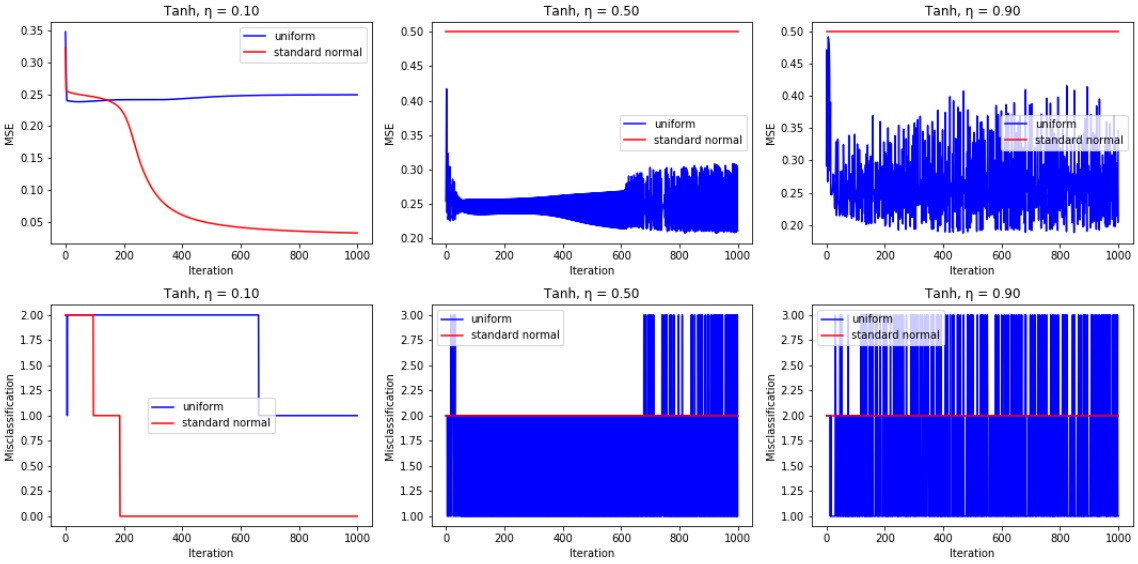

Figure 6: Tanh activation for MSE and classification error on learning rates 0.10, 0.50 and 0.90.

The tanh activation function performs well on a low learning rate and standard normal instigated weights, attaining a significantly lower MSE than even the sigmoid activation. However, as the learning rate increases, the network does not improve on the standard normal instigated set of weights, and oscillating behaviour starts to show after only a few iterations for the uniform set of weights. The oscillating behaviour can be explained by a local minimum in the error function found around 0.25. A large value for the learning rule does not allow the network to hone in on this minimum, as the step size is too big. The same can be seen in the classification error rate.

Lazy Approach

The lazy approach depends heavily on the distribution from which is being sampled for the weights. Random samples from uniform distribution never converge to a set of correctly classifying weights, whilst random samples from a normal distribution with mean 0 and standard deviation 1 (standard normal) often converge after only a handful randomization's, as can be seen in the histogram in figure 7. Sometimes correct classification is even reached at the first set of weights for the standard normal.

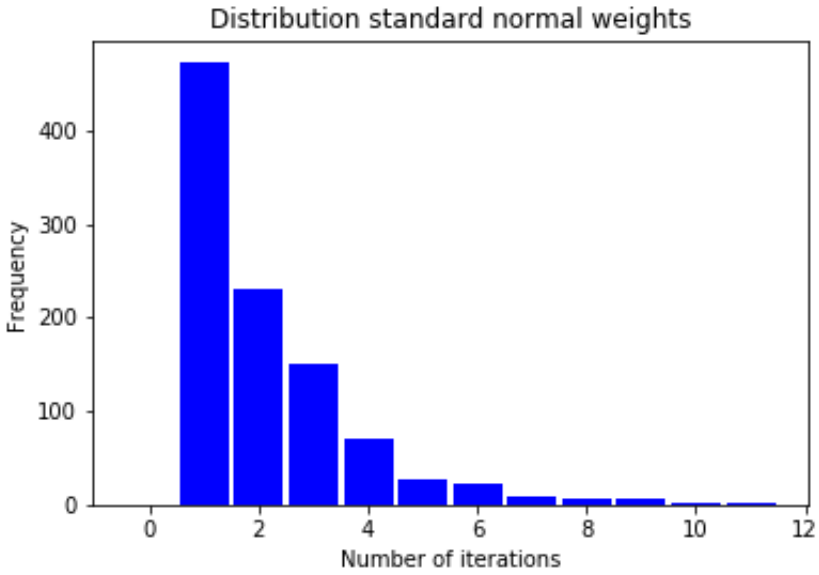

Figure 7: Lazy approach distribution of number of iterations for standard normal weights until correct classification.

References

[1] T.M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Computers*, 14, 1965.