Deep Learning Assignment 1

Murad Bozik
s2619822

Yuchi Zhang
s2724952

Hansha Leng
s2604825

March 19, 2020

1 Introduction

The objective of the project is to implement several algorithms in order to classify images of handwritten digits. Simplified version of MNIST data set has been used. The data set includes a collection of 2707 digits represented by vectors of 256 numbers. Each vector symbolize images of handwritten digits. MNIST data set is divided into a training set (1707 images) and a test set (1000 images).

2 Task 1: Analyze distances between images

The main purpose of the task 1 is to classify images of hand-written digits by developing distance based classifier. In order to reach the purpose, all images are considered as points in highly dimensional space. Since each image is represented by a 256 numbers, 256-dimensional space covers all points. In this space, all the points which represent the same digit creates a cloud of points. Classifier calculates the distance between the image and each centers of those clouds. The closest distance shows the class of that image.

2.1 Algorithm description

Cloud centers are calculated by getting the average value of every dimension for a particular digit points. Each cloud's radius is the max distance from the center to the points. Distances between each centers calculated and presented as a heat-map. Calculation of distances are performed using Euclidean metric.

```
# We split our data into small dataFrames and finding their centroids
digits = dict(), centers = dict(), radiuses = dict()
for i in range(10):
    digits[i] = df[df['label']==i].iloc[:, :-1]
    centers[i] = digits[i].mean()
    radiuses[i] = digits[i].apply(lambda x:euclidean(x,centers[i]),axis=1).max()
```

Digit	Total Points	Radius	Digit	Total Points	Radius
0	319	15.89	5	88	14.45
1	252	9.48	6	151	14.03
2	202	14.17	7	166	14.91
3	131	14.74	8	144	13.71
4	122	14.53	9	132	16.14

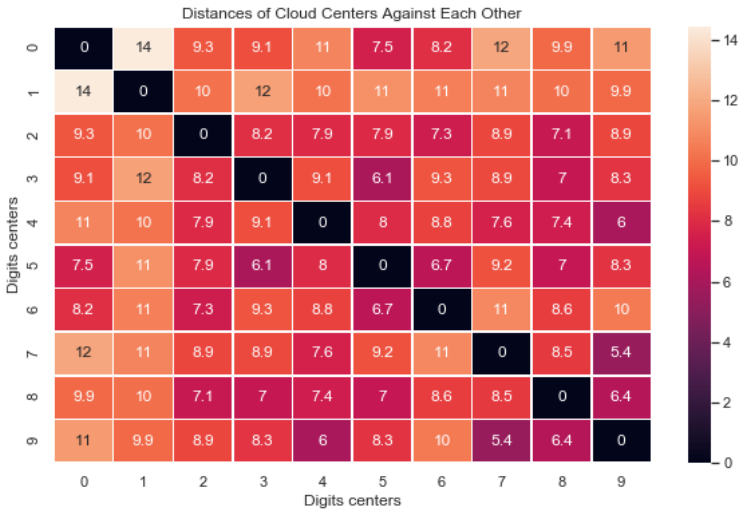


Figure 1: Visualization of center's distances

2.2 Discuss

This measurement method is reasonable, so the accuracy of the test set should be over seventy-five percent. By observing the distance between various centers, the nearest three pairs are (7,9), (4,9) and (8,9). These pairs seem to be most difficult three pairs to separate.

3 Task 2: Implement and evaluate the simplest classifier

3.1 Preliminaries

In this task, calculations of the distances are made by using the metrics which are valid in sklearn and scipy packages. The metrics used are as follows: *braycurtis, canberra, chebyshev, correlation, dice, hamming, jaccard, kulsinski, minkowski, rogerstanimoto, russellrao, seuclidean, sokalmichener, sokalsneath, sqeuclidean, yule, cityblock, cosine, euclidean, l1, l2, manhattan.* Calculating the Confusion Matrix is done with sklearn package.

3.2 Implementation

Implementation is achieved by creating *EuclideanClassifier* and *PairwiseClassifier* classes. During the initialization of these classes centroids of clouds are calculated. Prediction of a digit is performed by finding the closest centroids which represents the class of this image. Classifier implementation is applied on both train and test sets. The difference between two classifiers is just the metrics has been used. In *PairwiseClassifier*, metric you want to use can be passed as parameter of class.

3.3 Experiments with Euclidean metric

Euclidean Classifier experiment results are as follows:

	Total Images	Correctly Classified Images	Performance Rate
Training Set	1707	1474	% 86
Test Set	1000	804	% 80

Table 1: Performances of EuclideanClassifier on train and test sets

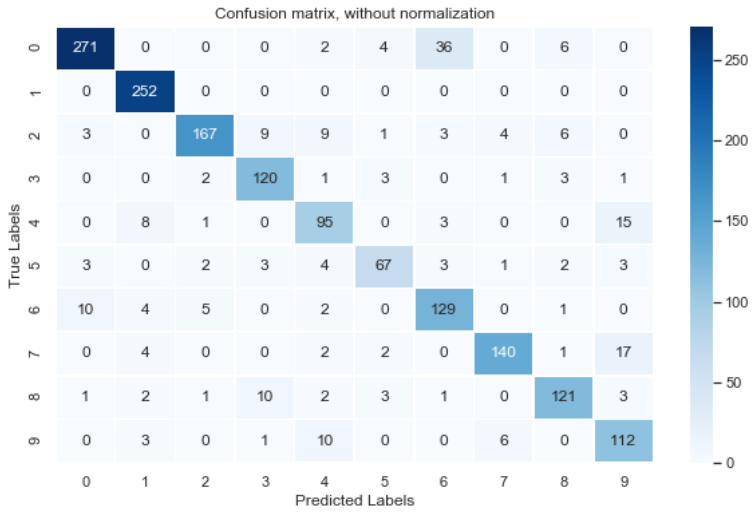


Figure 2: Confusion matrix on train set

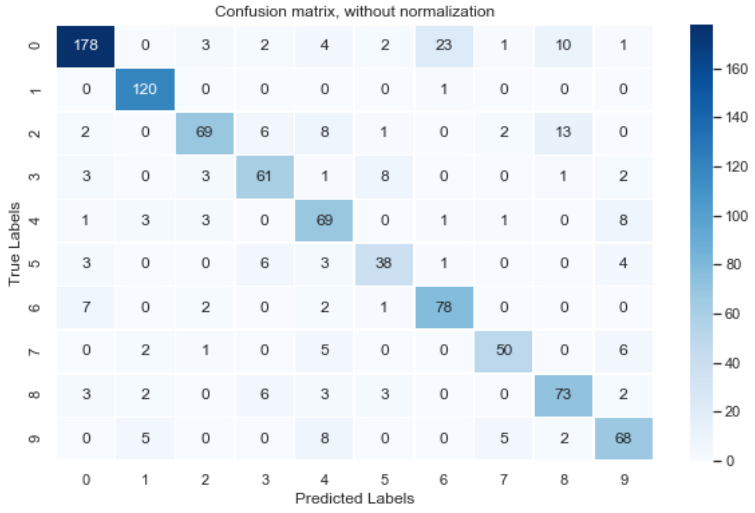


Figure 3: Confusion matrix on test set

3.4 Analysis of Results

The elements on the diagonal of the confusion matrix represent the number of each digit was classified correctly, and the rest represent the misclassification. Through confusion matrix on train set, 0 was classified to 6 was the most,the next is 7 was classified 9, then 4 was classified 9. 0 and 9 digit pairs are most difficult to classify correctly. Similar results on test set. The results compare to the observations in 2.4, 7 and 9 digits pairs centers are most close, and they were classified incorrectly quit a lot but not the most. The reason for this is the simple distance-based classified for the measurement of 0 and 6 center distance is not suitable. For all that, the accuracy rate on train set reached 86 percent and on test set reach 80 percent.

3.5 Alternative Distance Metrics Experiment

In this experiment 22 different distance metrics has been used. These metrics applied with sklearn and scipy libraries. Experiment is performed only on the test set. Figure 4; shows the performance

percentages for different metrics.

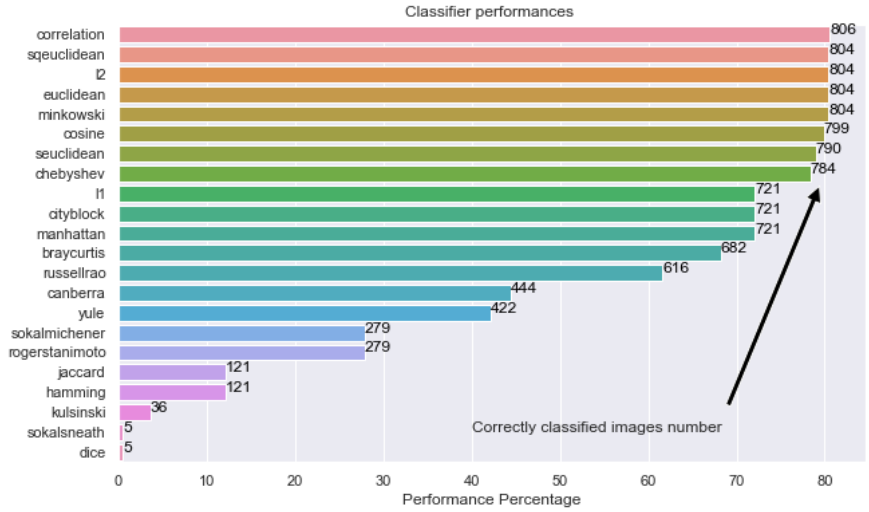


Figure 4: Based on alternative distance measures result

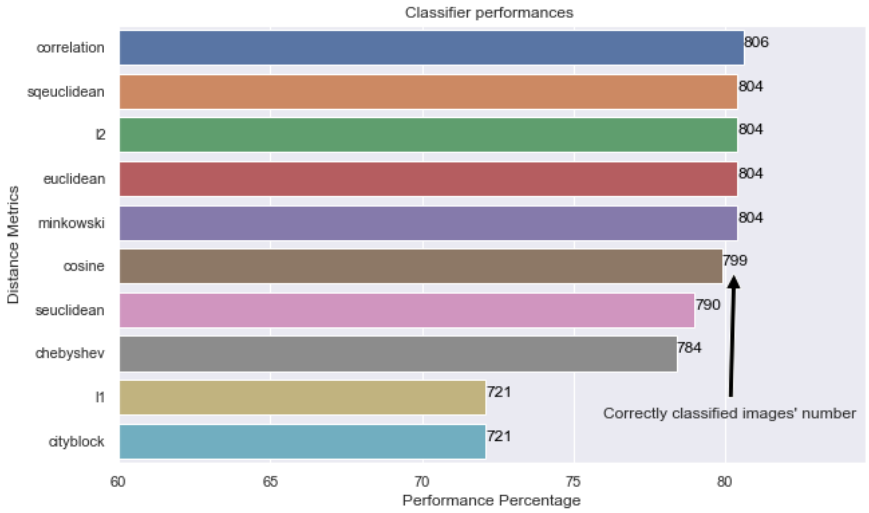


Figure 5: Top 10 distance measures

3.6 Analysis of the result

Through Figure 5,the best distance measure is correlation. In top 5 metrics, while correlation metric has %80.6 performance rate, the rest 4 metrics shows the same performances which is %80.4. This shows all of them has almost same performance. However other metrics performs clearly lower performances. Figure 4 shows how drastically changes the performances of different metrics.In practice, we should choose the suitable method for distance calculation.

4 Task 3: Implement a multi-class perceptron algorithm

4.1 Introduction

In this section, a multi-class perception algorithm implementation is used for classifying the same handwritten digit data set. Basic perceptron model consist of one node and input size + 1 (bias) weight. Since the goal is classifying 10 digits, our model consists of 10 nodes and each node has input size + 1 weights. That makes the weights matrix (257x10) dimensional array. Adding 1 as bias into the input matrix provides the advantage of making prediction easily with a single dot product. After the implementation, algorithm trained on training set and evaluated on both training and test sets. Later on training process improved with an error function. And these two training processes are compared on training set.

4.2 Implementation of Perceptron Algorithm

As a classifier; a class named *Network* is created. During the initialization of the class;

- training images augmented with 1's for the bias and is saved as an internal input parameter.
- Desired outputs is saved as parameter named actual.
- Weights matrix is created in a randomized way between 0 and 1.
- learning rate parameter is the only required parameter.

Training process is based on iteration number. Basically; in each iteration, predicted labels of the inputs compared with actual labels and if it is misclassified the weights of more activated nodes penalized while the correct nodes' weights activation empowered. Prediction is just the maximum activated index of the dot product of input and weights matrices.

As a second training method, training function is updated by adding an error function. The definition of the error is (d-y) desired - predicted. However it can take negative values depend on the value of desired and predicted. In order to eliminate this problem the error is defined as; $Error = 1/2 * (d - y)^2$

4.3 Experiment

Experiments applied on training and test sets for 50 iterations with 0.01 learning rate. Since the initialization of the weights is randomized, repetition of experiments gave slightly different graphics. Experiment results on training set presented below in Figure 6 and 7. The results on test set can be reached in code.zip file.

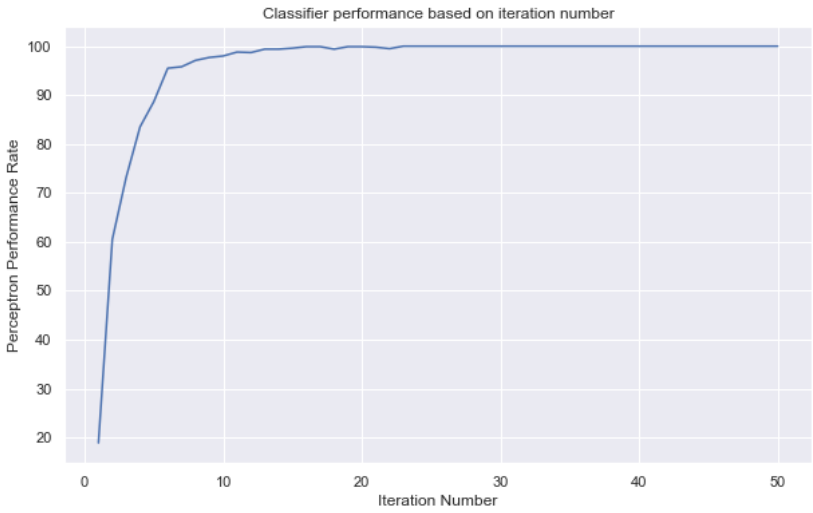


Figure 6: Network performance on training set

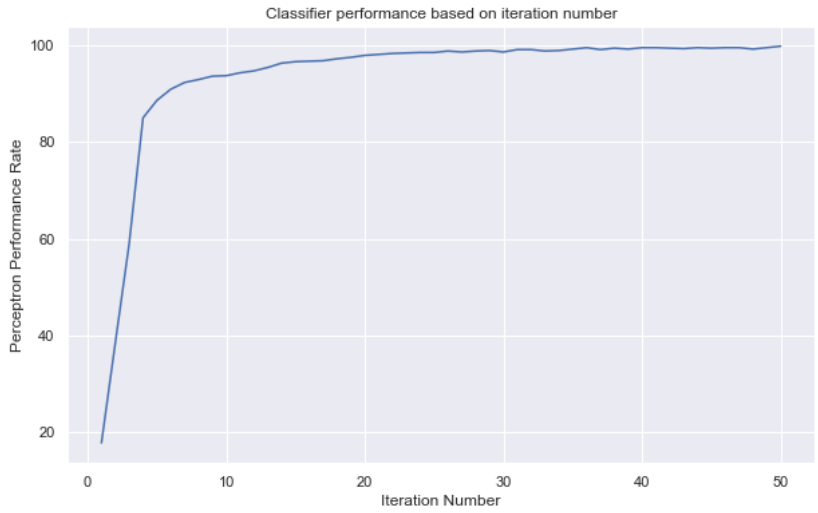


Figure 7: Network performance on training set with error function update

As can be seen from the comparison of two Figures, while perceptron algorithm from slide 36, second lecture accuracy rate reached 100 percent after 20 iterations, perceptron algorithm in the G´eron's textbook could reach after 50 iterations. But the former's convergence speed is slower than the latter's.

5 Task4: Linear Separability

5.1 Introduction

In this session, answers were sought to the questions of whether the digit i is linearly separable from digit j and whether the digit i is linearly separable from all remaining digits within the scope of Cover's theorem.

5.2 Experiment

According to Cover's Theorem:

- if the number of points in d-dimensional space is smaller than 2d then they are almost always linearly separable;
- if the number of points in d-dimensional space is bigger than 2d then they are almost always linearly non-separable.

In order to acquire all possible pairs of digits *combinations* function from *itertools* package has been used. First, a simple test has been applied to the set of all images of the digit i and j pairs. If the number of total points of i and j pair is smaller than 2*256 dimension, then it is accepted as linearly separable. The result shows that the number of points of *(0,1)* and *(0,2)* pairs are bigger than 2d (512), they are linearly non-separable. Later, the performances calculated on trained network with the points of pairs. The results after 100 iteration per pair presented in Figure 8 and with 3000 iteration per pair in Figure 9.

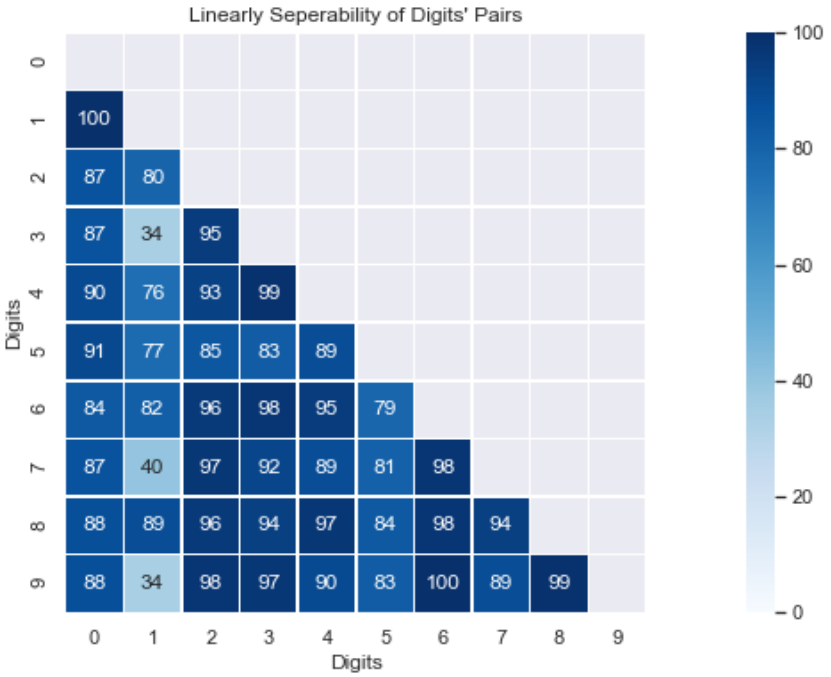


Figure 8: Linearly Seperability of Digits' Pairs with 100 iterations

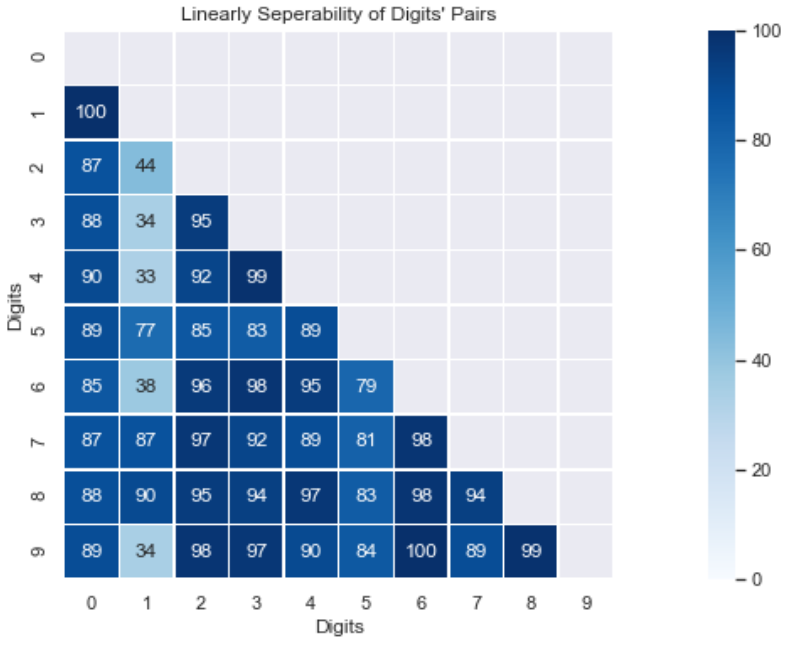


Figure 9: Linearly Seperability of Digits' Pairs with 3000 iterations

6 Task5: Implement the XOR network and the Gradient Descent Algorithm

6.1 Introduction

In this section, XOR neural network has been implemented from scratch (without TensorFlow or other framework). Sigmoid function has been used as activation function for each non-input nodes in the network. Sigmoid function is an S-type function commonly used in biology, also it is called an S-shaped growth curve. Implementation includes not only sigmoid activation function also other alternatives such as hyperbolic tangent (tanh) and rectified linear unit (ReLU) functions. The impacts of various activation functions and learning rates have been compared in Section 6.3.2.

In this simple example, we will not care about statistical generalization. We will use all 4 points $(0,0),(0,1),(1,0),(1,1)$ to train our network, and the only challenge is to fit the training set. We can treat this problem as a regression problem and use the mean square error loss function. This loss function was chosen to simplify the mathematical knowledge used in this example as much as possible. In the application field, when modeling binary data, MSE is usually not a suitable cost function.

6.2 Design and Implementation

A very simple feedforward neural network has one input layer, one hidden layer and one output layer. The hidden layer contains two units, as explained in the Figure 10 below.

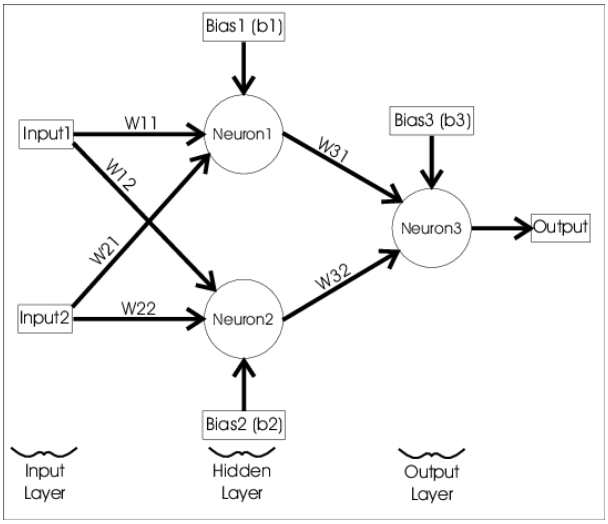


Figure 10: XOR net model[1]

Hidden layer produces a vector h as output of hidden units calculated via their activation function. The values of these hidden cells are the inputs of the second layer. The second layer is the output layer of this network. The output layer is still just a linear regression model, but now it works on output of hidden layer instead of input layer. [2]

First, XOR network is constructed by creating *hidden_weights*, *hidden_bias*, *output_weights* and *output_bias*. XOR net is simulated in *xor_network* function using numpy arrays for input layer, weights and output. Then, in order to process step 2: implementation of error function, *mse(weights)*, two functions *puremse* and *sigmoidmse* have been created. Next, step 3: gradient of error function, *grdmse(weights)* [4], the gradient of the mse function needs to be recalculated and the weights and biases need to be updated in each turn of the training process.

Finally, step 4: implementation of gradient decent algorithm. In this step;

1. weights are initialized to random values
2. implementation is iterated 10000 times with learning rate *lr* improvement. *lr* is a small positive constant called "step size" or "learning rate".

Results of the experiment are presented in the section 6.3.

XOR network training process can be summarized as following [3]:

- Initialize the weights and biases randomly. Iterate over the data
- Compute the predicted output using the sigmoid activation function
- Compute the loss using the square error loss function
- $W(new) = W(old) - \alpha \Delta B$
- $B(new) = B(old) - \alpha \Delta B$

This process repeats until the error becomes minimal.

6.3 Experiment

6.3.1 Monitoring MSE and Misclassified Inputs During Training

The error depends on misclassified inputs. If the misclassified input number decreases error decreases. Figure 11 presents the observation of the MSE change in each turn during training. The MSE values keep going down. It prove that the neural network's performance getting better and better.

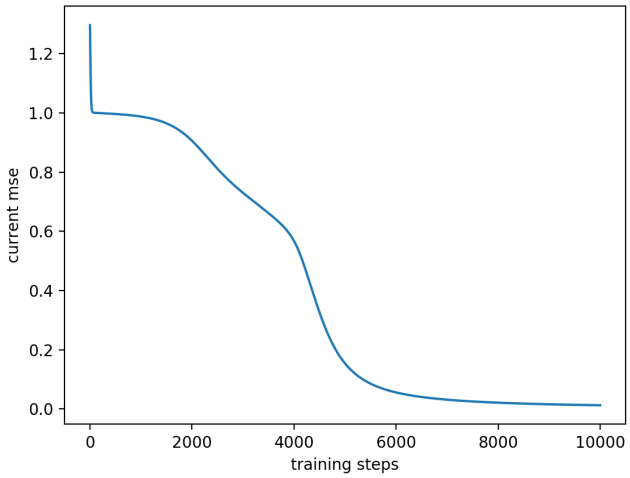
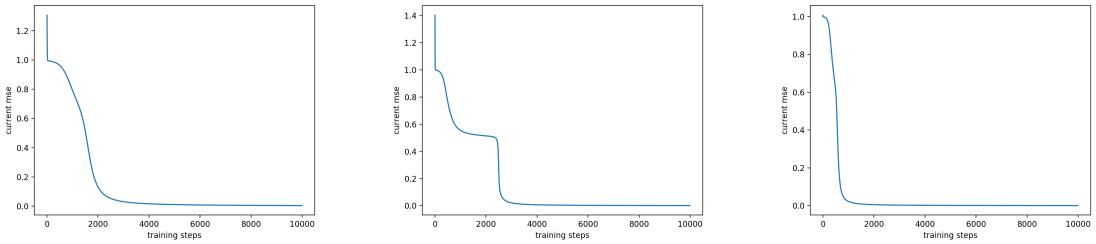


Figure 11: The mse with the training steps with lr = 0.1

In Figure 11, it is clear that the mse decline really fast in beginning. Then it decline more slower. Because the error value in the beginning is too high and with learning rate update of the weights becomes high. But when the error decreased, it updates the weights much smaller and this cause a slower convergence as the error decreases. [5].

6.3.2 Learning Rate and Activation Functions

In this section, the impacts of setting different learning rates and activation functions are compared.



(a) *mse* while *lr* is 0.2 (b) *mse* while *lr* is 0.5 (c) *mse* while *lr* is 0.8

Figure 12: the different learning resulting in different mse convergence

Figure 12 exhibits the mse change along training process with different lr rates. It provides the idea that the more higher learning rate can cause more sharp convergence.

6.3.3 Lazy Approach

An interval method is used for finding the time needed to predict the desired output correctly. In order to produce the output $0, 1, 1, 0$ 539 turn spent. That means, 539 times weights updated than it could give this output. We can see that it's a pseudo-random and the once get a final $0, 1, 1, 0$. We can know that, a part of time the result can keep that positive answer. I also think because it's a simple neural network, that random can result less steps. However, when we have a complex neural network, the result can be really different. Therefore, It's just a coincidence to have that less train times than the gradient.

6.4 Analysis of Task 5

In this task, we simply specified the solution and then stated that it got zero errors. In the actual situation, there may be billions of model parameters and billions of training samples, so we cannot make simple guesses as we do here. In contrast, the gradient-based optimization algorithm can find some parameters that make the error very small. The solution of the XOR problem given here is at the global minimum point of the loss function, so the gradient descent algorithm can converge to this point. The gradient descent algorithm can also find some other equivalent solutions to the XOR problem. The convergence point of the gradient descent algorithm depends on the initial values of the parameters. In practice, gradient descent usually does not find a clean, easy-to-understand, integer-valued solution like the one given here.

7 Conclusion

Overall, classification of hand-written digits made by several algorithms. The observation shows that neural network is better than distance based classifiers, also initialization of the weights are so much important to get good results in neural networks. Gradient descent optimization is very efficient to reduce the error. However, initialization of the weights and selecting right learning rate is important to get reasonable results.

References

[1] Siddhartha Dutta. Implementing the xor gate using backpropagation in neural networks, 2019.

[2] Leonard GC Hamey. Xor has no local minima: A case study in neural network error surface analysis. *Neural Networks*, 11(4):669–681, 1998.

[3] Tohru Nitta. Solving the xor problem and the detection of symmetry using a single complex-valued neuron. *Neural Networks*, 16(8):1101–1105, 2003.

[4] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[5] Xiao-Hu Yu, Guo-An Chen, and Shi-Xin Cheng. Dynamic learning rate optimization of the back-propagation algorithm. *IEEE Transactions on Neural Networks*, 6(3):669–677, 1995.