In The name of God

University Tehran

Engineering Facility

Electrical and Computer Engineering

Deep Learning with Applications

HW # 1

Amin Fadaeinedjad

810195442

Spring 99

Abstract

In this Homework we were going to implement n neural network from scratch, and we needed use different layer at the end of the network and for this different type of networks we tried different batch size for the computation. We also see that if we use noisy data in our train dataset and we see that it doesn't have a big impact on the test accuracy. The next part of the HW relates to the Adversarial attacks and the effects that may cause on the network, and after that we see what we can do to reduce the risk of such adversarial attacks.

Before anything it is needed to be mentioned that the data we have used in the HW is 784 dimensional and we want to use PCA method to reduce the dimension of the data's therefor the data set that we are using in this HW has 128 dimensions and the python code is in file named PCA.ipynb.

Question 1

We need to implement the network with 128 input nodes and 150 hidden nodes and at the end of the network we are going to 10 nodes. And in this implementation we are going to use the Hinge loss to for the loss function in our neural network, it is implemented as the equation below.

$$Hinge\ Loss \quad L = \sum_{i=0, i\neq j}^{N} \max(0, s_i - s_j + 1)$$

In equation for the Hinge loss s_i is the score for the $i'th$ class and s_j is the score for the $j'th$ class or in other words score for the correct class.

$$\frac{\partial L}{\partial s_i} = step(s_i - s_j + 1) = \begin{cases} 1, s_i - s_j + 1 \geq 0 \\ 0, s_i - s_j + 1 < 0 \end{cases}$$

$$\frac{\partial L}{\partial s_j} = \sum_{i=0, i\neq j}^{N} -step(s_i - s_j + 1)$$

$$J^* = argMax(s_i(x))$$

The equations show the gradient of the lost function to the class scores and we use this update the weights of the network we have.

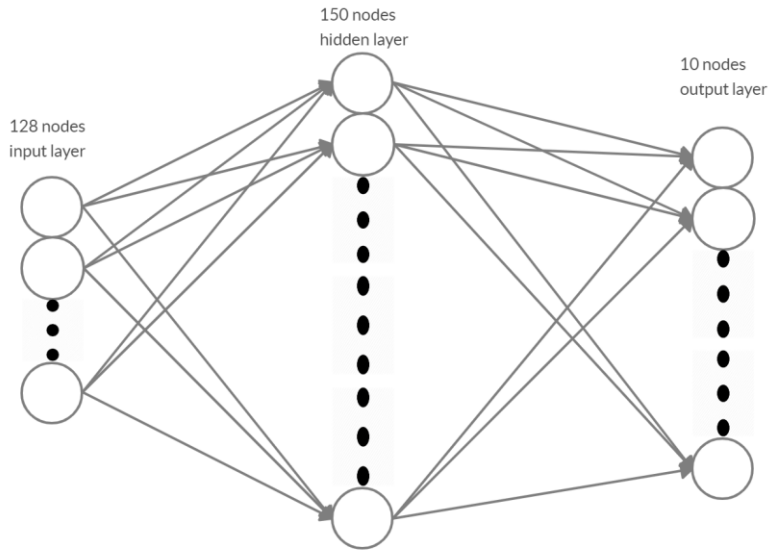

Figure 1

According to figure 1 we can see the structure of the network, in the gradient decent algorithm we also divide the gradient by the batch size train data given to the network.

We can use several different learning rates for different number of epochs the result shows the accuracy and the time needed for training the neural network.

Model Name	Number epoch	Batch size	Learning rate	Accuracy for Train data	Accuracy for Test data	Training duration (sec)
Hinge loss	10	128	0.05	87.5%	85.9%	114.2
Hinge loss	20	128	0.005	84.9%	83.2%	229.8
Hinge loss	5	128	0.2	79.4%	77.3%	57.62
Hinge loss	5	128	0.1	87.45%	85.7%	57.12
Hinge loss	15	128	0.1	90.5%	87.54%	182.4

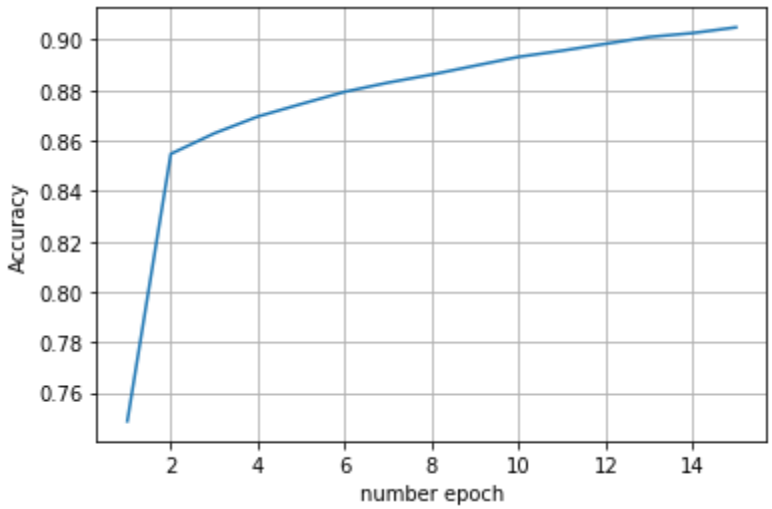

Figure 2 learning rate = 0.1

Figure 2 shows the accuracy of the network for data.

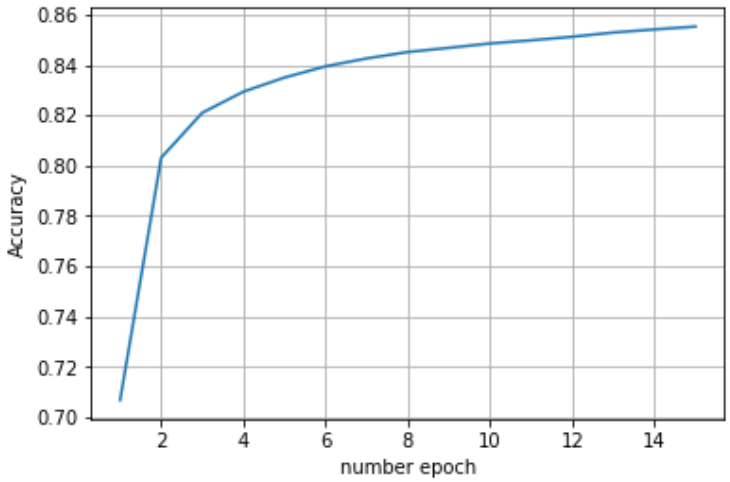

Figure 3 learning rate = 0.01

Figure 3 shows that if we use smaller learning rate we may reach a certain accuracy a bit longer but the way is smother than the time when we use a large learning rate.

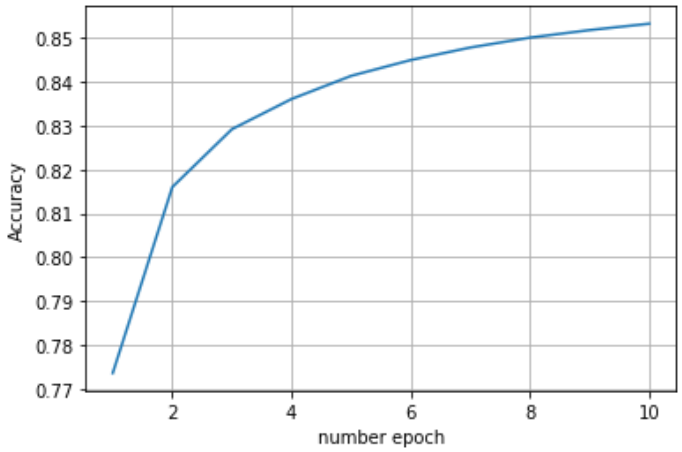

Figure 4 batch size = 1024 learning rate = 0.1

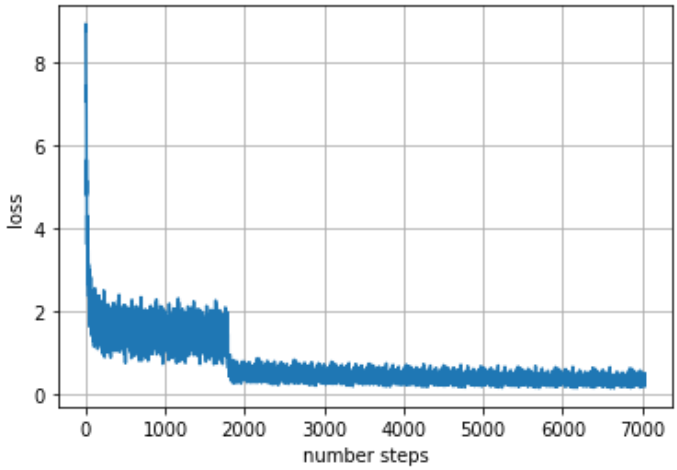

Figure 5 learning rate = 0.1

Figure 5 show the loss of the network.

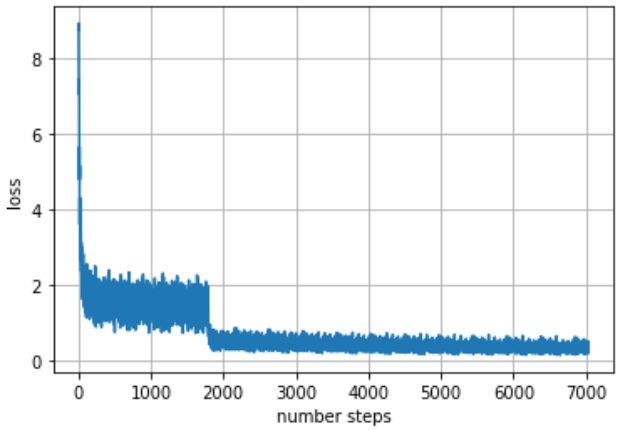

Figure 6 learning rate 0.01

Figure 6 show the loss of the network when the learning rate is 0.01 we can see that the learning rate didn't affect the loss of the network that much.

And there is a point in the Hinge loss function if the score of the correct class is more than the score of another class +1 then gradient decent does not have an effect on that node.

Question 2

In question 2 we are going to implement the same network but with different activation function call $Gelu$ or Gaussian error linear unit and a RBF at the end of the model.

The difference that this network has compared to the previous network is that RBF works like metric learning method witch tries to minimize the score of the correct label and increase the score of the wrong scores.

$$Loss = \max\left(0, d_j(x)\right) + \sum_{i=0, i\neq j}^{N} \max\left(0, \lambda - d_i(x)\right)$$

It can be understood from the equation that if the score of the node is more than λ then that node has no influence on the loss.

$$Gelu(x) = 0.5x\left(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)$$

$$J^* = argMin\left(d_j(x)\right)$$

The $gelu$ function has cons and pros we can say that for positive input out gradient does not vanish and this help us use gradient decent to change the networks parameters and for the negative inputs the gradient is small but not zero and this could also help preventing from vanishing gradient.

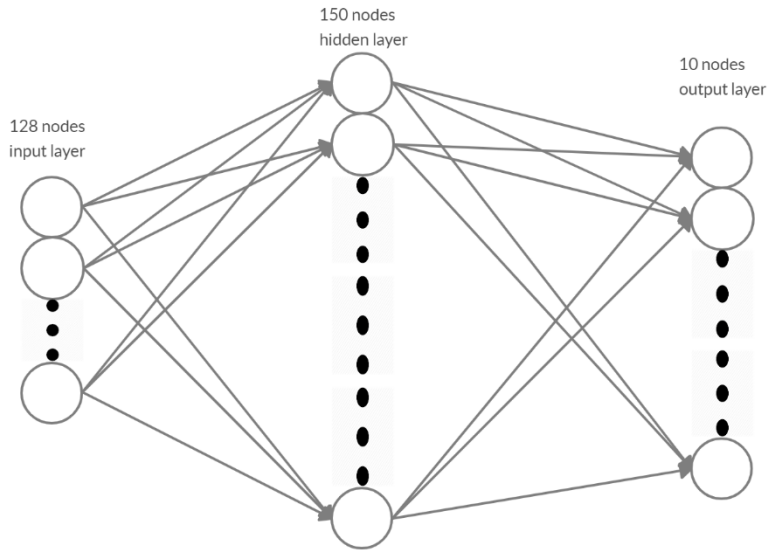

Figure 7

The structure of the network is just like the one in question 1 with 128 input nodes, 150 hidden nodes and 10 output nodes.

Model Name	Number epoch	Batch size	Learning rate	Accuracy for Train data	Accuracy for Test data	Training duration (sec)
RBF loss	10	128	0.05	85.3 %	84.06%	382.4
RBF loss	10	128	0.1	87.2%	85.75%	385.96
RBF loss	10	128	0.01	76.26%	76.19%	367.12

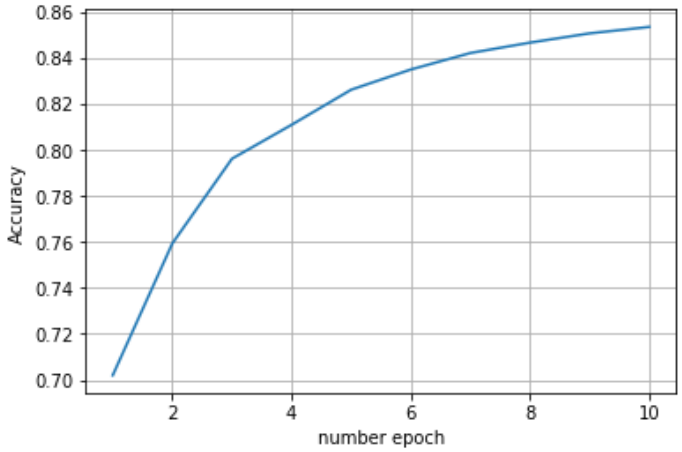

Figure 8 learning rate = 0.05

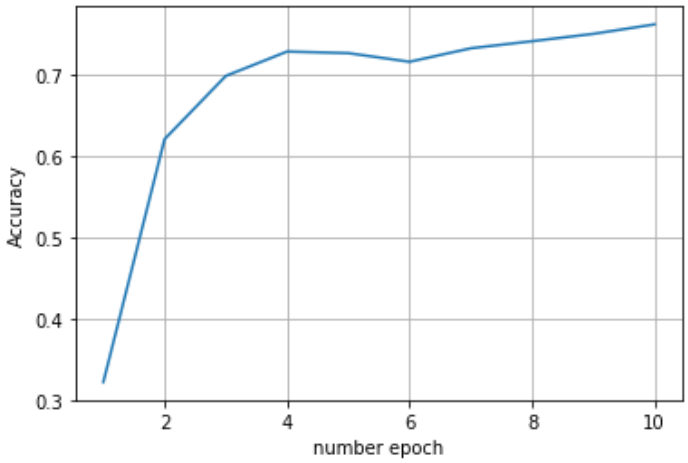

Figure 9 learning rate = 0.01

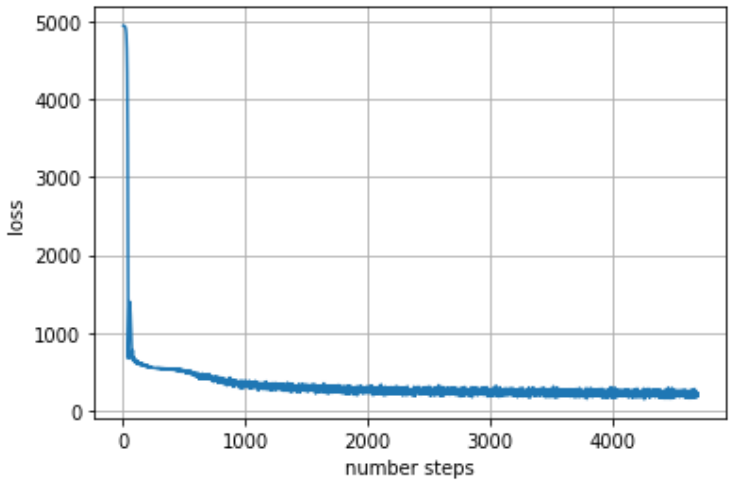

Figure 10 learning rate =0.05

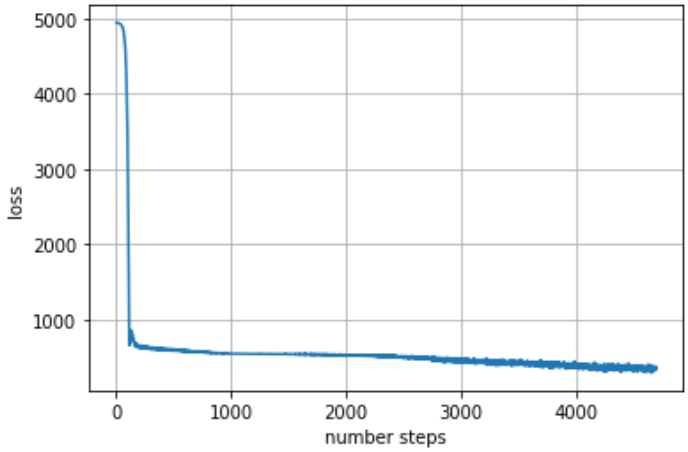

Figure 11 learning rate = 0.01

Question 3

We done the Training in the previous parts of the HW and the result is shown in question 1 & 2.

But still we are going to show some other case of the training network.

Stochastic gradient decent is quite better than batch training the reason is that we have a big dataset and if we were going to train the network with all this data points then we will face some problems in this case because of the large matrix and multiplications in the training process it will take long and that isn't good so we come up with another called the SGD or the stochastic gradient decent and the way that this work is that instead of sending all the data point to the network.

Model Name	Number epoch	Batch size	Learning rate	Accuracy for Train data	Accuracy for Test data	Training duration (sec)
Hinge loss	5	1	0.01	87.3 %	85.81%	990.44
Hinge loss	5	1	0.005	91.4%	87.93%	1021.1
RBF loss	10	1	0.1	69.39%	68.71%	2870

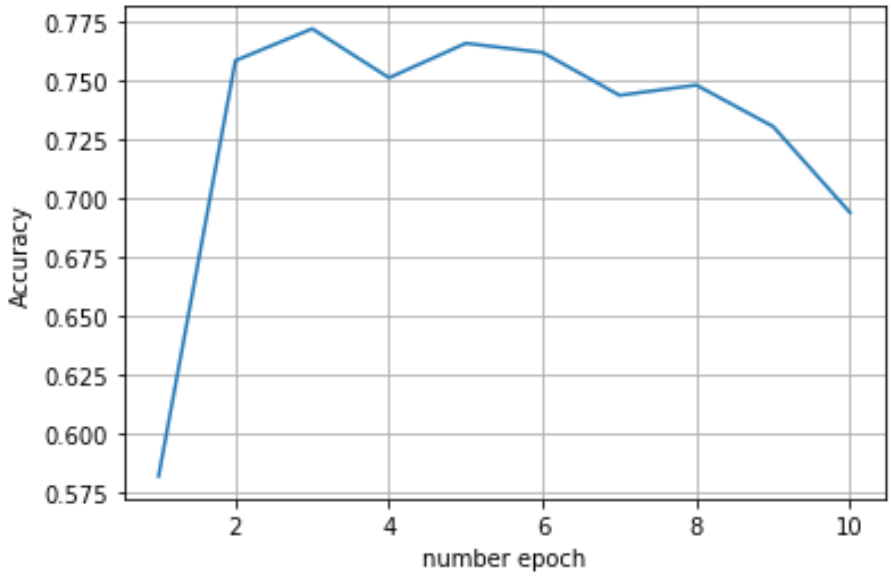

Figure 12 RBF rate=0.1

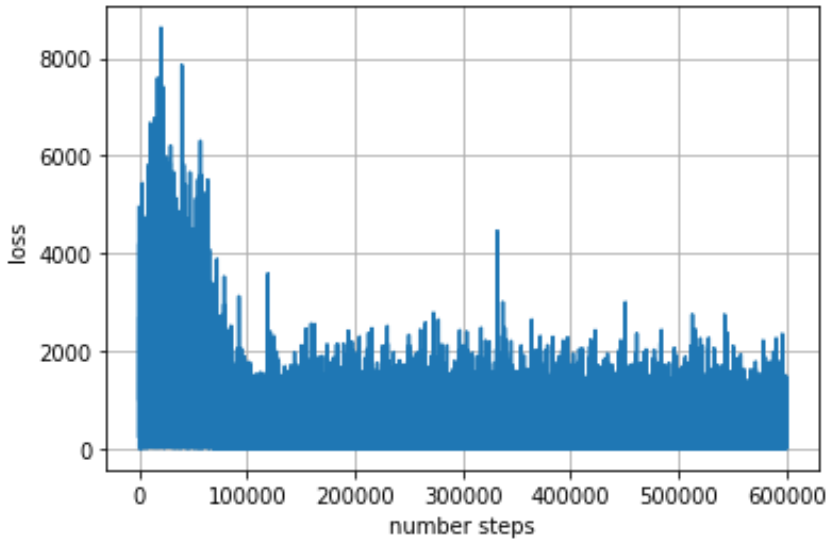

Figure 13 RBF rate=0.1

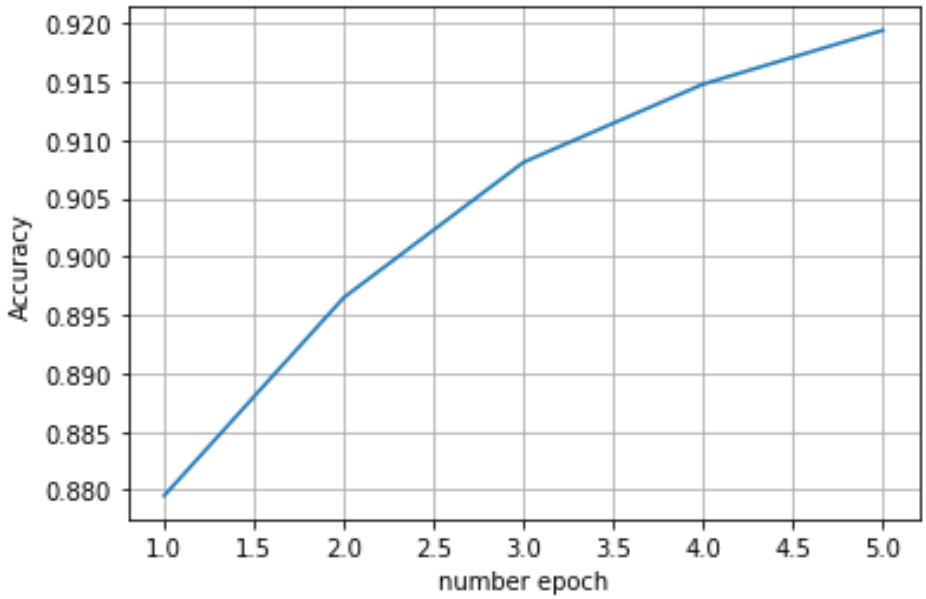

Figure 14 Hinge 0.01

The figure shows the accuracy after the epoch

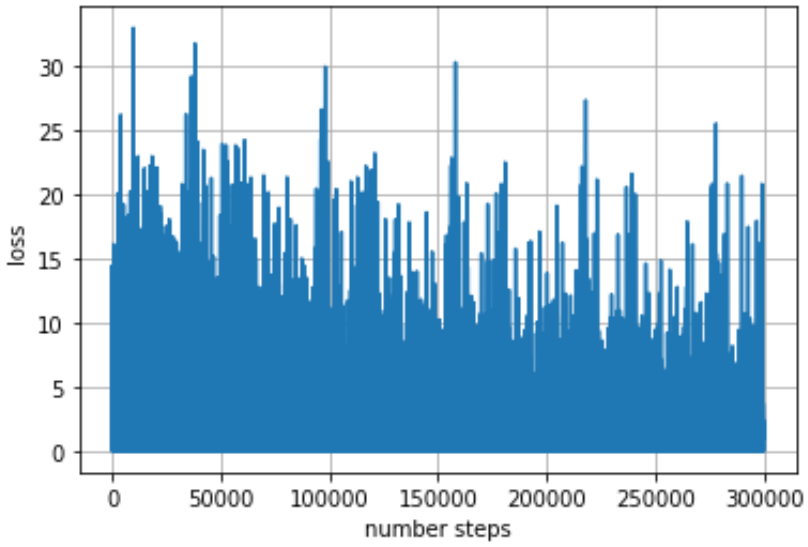

Figure 15

Question 4

In this part we are going to try the original data and compare it with the previous networks and compare the results. We can notice that when we use the main data because of the size of them the time needed to train the data set will increase.

Model Name	Number epoch	Batch size	Learning rate	Accuracy for Train data	Accuracy for Test data	Training duration (sec)
Hinge loss	3	1	0.001	87.3 %	85.81%	1365
Hinge loss	20	128	0.05	87.91%	86.01%	989

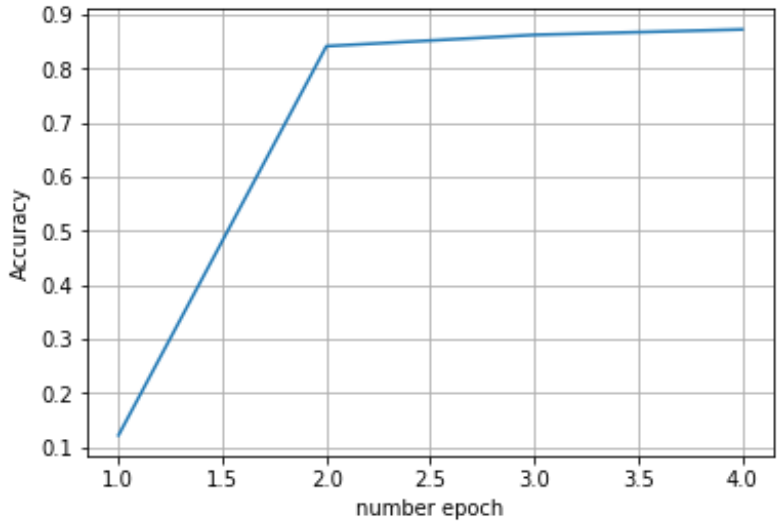

Figure 16 784 dimension rate=0.001

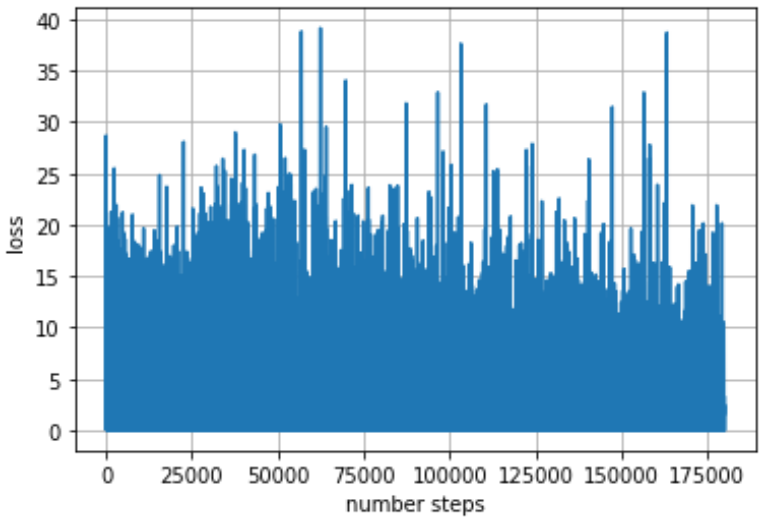

Figure 17784 dimension rate=0.001

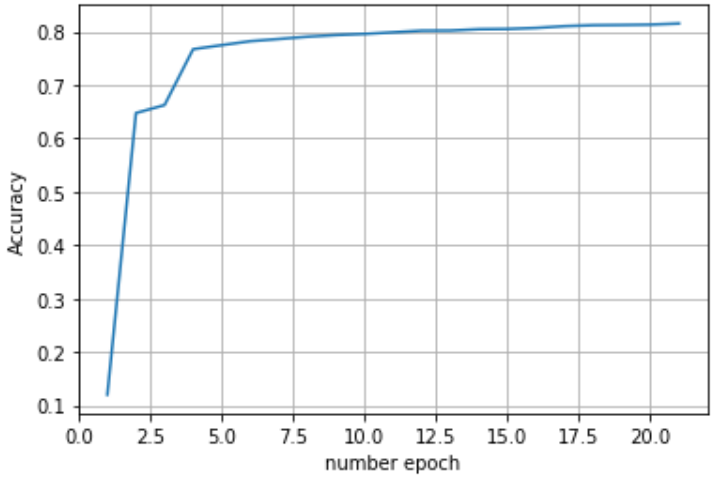

Figure 18 batch size = 128 rate = 0.05

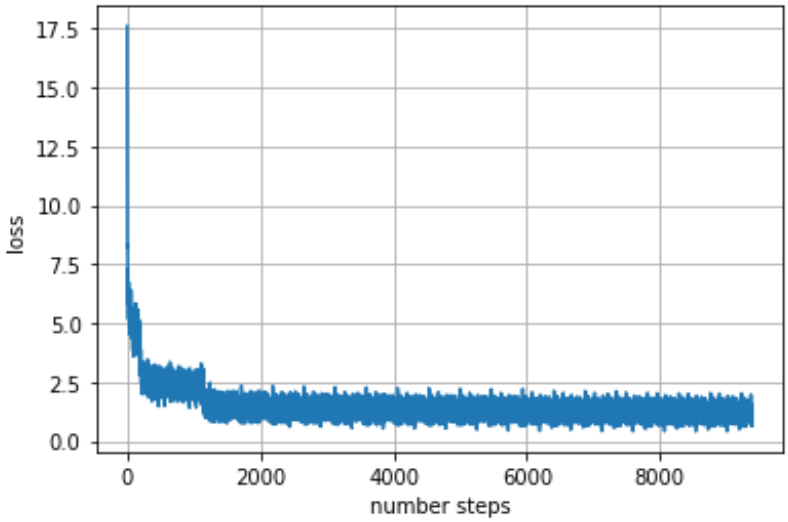

Figure 19 batch=128, rate = 0.05

From the table and the figure from this part we can see that using larger dimension for the data will increase the computation time so it shows that if we use the data after the PCA method we are saving time for our computation and this dimension reduction doesn't quite have influence on the accuracy of the network.

It is obvious that the time needed for this will take longer than the previous one.

Question 5

In this part we are going to see the effect of the network when out data aren't pure and have Gaussian noise on them so it may cause some problems for the classification but we must test it to make sure want happens.

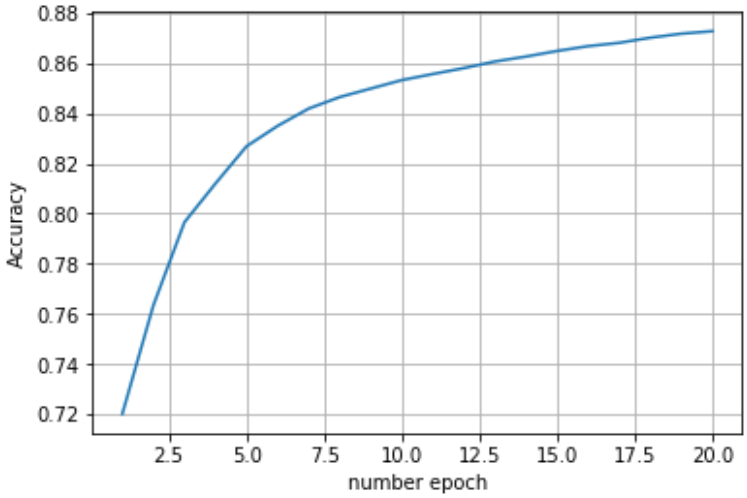

Figure 20

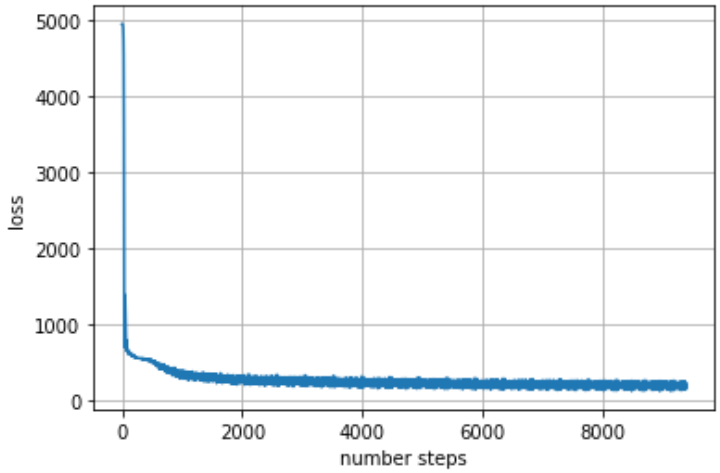

Figure 21

This shows a goof result for the network and the neural network theory why? Because that even though we added Gaussian noise to the train data isn't influenced by the noise that we have added to the network.

Model Name	Number epoch	Batch size	Learning rate	Accuracy for Train data	Accuracy for Test data	Training duration (sec)
RBF loss noise data	20	128	0.05	87.27%	85.96%	433.77

So as a result this means that we can easily give noisy data (but the scale of the noise shouldn't be that high) and we know what will not have much effect on the network.

Question 6

In this question we are going to use the FGSM method to attack our neural network and we want to see the result.

We also reached nice point that if the loss of a data is equal to zero in that case we can't find a gradient to attack the image in other words we need to have the gradient of the loss from the input data and because is some cases we are using $relu, gelu$ activation functions or loss functions that they have a max() function in it, it show that when the loss is equal to zero we have no gradient so in that case there is no direction to go in order to change the loss of the network.

$$new_X = X + \epsilon \times sign\big(\nabla_x J(\theta, x, y)\big)$$

Model name	Model accuracy	Arrack rate(ϵ)	Attack accuracy
Hinge loss	83.96%	0.1	40%
Hinge loss	83.96%	0.01	0%
RBF loss	79.22%	0.1	30%
RBF loss	79.22%	1	90%
Hinge loss 784 dim	83.21%	0.1	10%
Hinge loss 784 dim	83.21%	1	100%

It is clear that by increasing the attack rate the success of the attack will increase and we can see the result on the table above.

Part1) In this part we used $\epsilon = 0.1$ and the accuracy of the attack was 40% which means 4 out of 10 attacks were successful

The cool thing that we found out is the for the cases where the loss is equal to zero it means that the network gradient is equal zero for this structure of network and not matter how big this attack rate is it doesn't have an effect so if even we put a high value for the attack rate for the Hinge loss model we the attack accuracy will never be more than 40% for this data that we tested because only 4 data out of 10 data points had a loss more than zero.

Part2) In this model the structure of network the loss never reaches zero so in that case we don't have the problem for attacking like the previous part which is from the vanishing gradient so we can see that if we increase the value of the attack rate the success of the attack will increase unlike the previous model.

Part3) In this part we see that the changes on the attack rate has a bigger influence on the result of the network the reason is that the dimension of the input is much bigger than the other two models and the structure of the design for the case to give zeros loss is a bit harder but still it can be possible

Question 7

In this part we are going to try the method to defend from the attacks and the way that we are doing this is that we define a new label 10 (which in the HW wrote -1 but still we use 10 because it's easier to handle).

The way this method works is that we gave our data set 200 pure noise data and label them as 10 and then we train the data

The result of this experiment was like this we gave the model 100 data which are classified correctly by the neural network

Model Name	Model Accuracy	Attack rate	Attach accuracy	Number correct Reject (out of 100)
RBF loss	79.78%	1	94%	42
RBF loss	79.78%	0.3	83%	37
RBF loss	79.78%	0.1	16%	1

The way that we decide for this method is that at the last layer we will get all the scores and the if all the amount of $d_i(x)$ are larger that λ then in that case we are going to reject the decision that we are going to make and if the exists at least one score that is less that the λ then in that case we will return the minimum argument for the predicted label.

And for the updating gradient decent method we need to define a cost function for the noise data that we gave to the network.

$$loss = \sum_{j=0} \max\left(0, \lambda - d_j\left(x^{(i)}\right)\right)$$

Question 8

In this part we are going to use the model implemented in part 7 and part 6 for this part we need to try different λ to see how it reacts to this parameter.

Model name	λ	Attack rate	Network accuracy	Accuracy of attack	Number Rejection in 100	Percent rejected
RBF loss	400	1	80.1%	93%	46	49.46%
RBF loss	400	0.1	80.1%	26%	8	30.76%
RBF loss	550	0.1	79.61%	16%	0	0
RBF loss	550	1	79.61%	96%	50	52%
RBF loss	550	0.5	79.61%	88%	52	59%
RBF loss	200	0.5	79.69%	88%	52	59%
RBF loss	100	0.5	80.86%	96%	56	58.33%
RBF loss	50	0.5	81.58%	94%	55	58.5%

So as we can see the value of the hyper parameter λ does not have much effect of the accuracy of the network but still we can see that if we increase the value of it we will get less accuracy for the network and for the rejection there still isn't much effect on that either

Process

In this HW we learned how to design a neural network from scratch and exactly know how to gradient decent works on the neural network and also we saw how attacks on neural networks can cause the network to misjudge the label of a data point even though the data is quite similar to the previous data, and at the end of the HW we learned a simple method for what we can do to reject the data's that are for attacking the network and reject them so it can't cause problems for the future.

Reference

Universal Denoising Networks : A Novel CNN Architecture for Image Denoising

Gaussian Error Linear Units(GELUs), Dan Hendrycks, Kevin Gimpel

Deep-RBF Networks Revisited: Robust Classification with Rejection, Pourya Habib Zadeh, Reshad Hosseini, Suvrit Sra

https://www.youtube.com/watch?v=GlcnxUlrtek

https://towardsdatascience.com/adversarial-examples-in-deep-learning-be0b08a94953

https://medium.com/towards-artificial-intelligence/understanding-back-propagation-in-an-easier-way-you-never-before-42fe26d44a47

https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79