MNIST Classification using Multi-layered Perceptrons

Hoang Ba Cong

February 2020

1 Introduction

In this report, I will describe and explain the formula and implementation for classify new handwriting data from MNIST. There are 6 parts consisted in this report.

2 Presenting the problem

Initially, I will read the data which are abundance of vectors consisting 784 numbers corresponding the brightness of pixels in each image and the labels showing the actual number of image. Then, my task is to implement a neural network architecture to identify new handwriting data from the test set.

3 Architecture of neural network

Basically, there will be L weighted matrix $W^l \in R^{d^{(l-1)} * d^{(l)}}$ in multi-layered neural network consisting of L layers. Each node i in one layer will be connected to nodes j from the other layer with certain weight $w_{i,j}$ since MLP is fully connected. Bias of layer l_{th} is equivalent to $b^l \in R^{d^{(l)}}$. Every output of a layer except for input layer is calculated according to formula $a^l = F^l(W^l * a^{(l-1)} + b^l)$. In this problem, I reckon then softmax function is most appropriate activation function as the outputs are one-hot vectors.

4 Feedforward and Backpropagation

The problem is how to optimized all the weight W^l in order to enhance the accuracy of neural network. Hence, the network has 2 phase which is Feedforward and Backpropagation:
Note that the pair data for training is (x_i, y_i) where y_i is one-hot vector. Each step of Feedforward calculation will be described below:

- $Z^1 = W^{(1)T}X + B^1$
- $A^1 = softmax(Z^1)$
- $Z^2 = W^{(2)T}A^1 + B^2$
- $\hat{Y} = A^2 = softmax(Z^2)$

The key in classifying data is minimizing loss function commonly based on cross entropy: $J(\mathbf{W}, b) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji} log(\hat{y}_{ji})$. Let us denote the set of all optimal weight , the goal of learning for neural network is find such that: $\Theta = \Theta argmin\mathbf{J}(\mathbf{W}, b)$. In this case, the efficient way to find θ is momentum gradient decent. It is made by iterative through steps in the following:

- Initiate θ with random positive and negative weights $(-0,5 <= 0,5)$
- $\theta_{new} = \theta - eta *_\theta J + gamma * (\theta - \theta_{old})$ where eta and gamma are learning rate and momentum factor .
- Update θ and reimplement feedforward.

To do this, calculating ${}_\theta J$ is necessary so we have the derivation of loss function according to one element of weight matrix of one layer:
$\frac{\partial J}{\partial w_{ij}^L} = \frac{\partial J}{\partial z_j^L} * \frac{\partial z_j^L}{\partial w_{ij}^L} = e_j^L a_i^{(L-1)}$ where $\frac{\partial z_j^L}{\partial w_{ij}^L} = a_i^{(L-1)}$ cause $z_j^{(L)} = w_j^{(L)T} a^{(L-1)} + b_j^{(L)}$. Similarly, derivation of loss function according to bias of last layer is $\frac{\partial J}{\partial b_j^L} = \frac{\partial J}{\partial z_j^L} * \frac{\partial z_j^L}{\partial b_j^L} = e_j^L$
Now, we have to calculate
$e_j^L = \frac{\partial J}{\partial z_j^L} = \frac{\partial J}{\partial a_j^{(l)}} * \frac{\partial a_j^{(l)}}{\partial z_j^L}$
$= (\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} . \frac{\partial z_k^{l+1}}{\partial a_j^{(l)}}) f^{(l)'}(z_j^{(l)})$
$= (\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} . w_{jk}^{(l+1)}) f^{(l)'}(z_j^{(l)})$
For this problem, we can infer the formula for backprogapation calculation with the steps below:

- $E^{(2)} =_{Z^{(2)}} = \frac{1}{N}(A^{(2)} - Y)$
- ${}_{W^{(2)}} = A^{(1)} E^{(2)T}$; ${}_{b^{(2)}} = \sum_{n=1}^{N} e_n^{(2)}$
- $E^1 = (W^2 E^2) . f^{'}(Z^{(1)})$
- ${}_{W^{(1)}} = A^{(0)} E^{(1)T} = X E^{(1)T}$; ${}_{b^{(1)}} = \sum_{n=1}^{N} e_n^{(1)}$

5 Experiment

As we obtain the experiment with epochs in figure1, the more epochs the higher accuracy we could get with the decrease of loss cost. Thus, we could assume

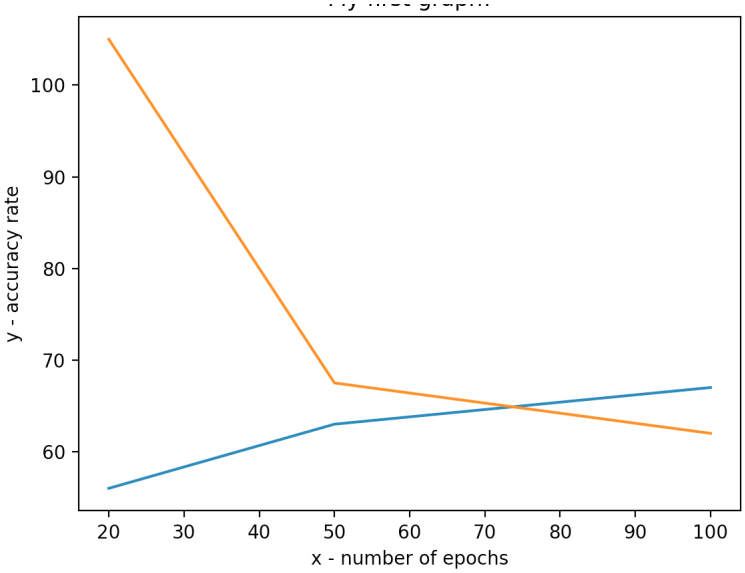

Figure 1: Experiment with epochs

that the number of epochs affected to the accuracy although runtime might be longer.

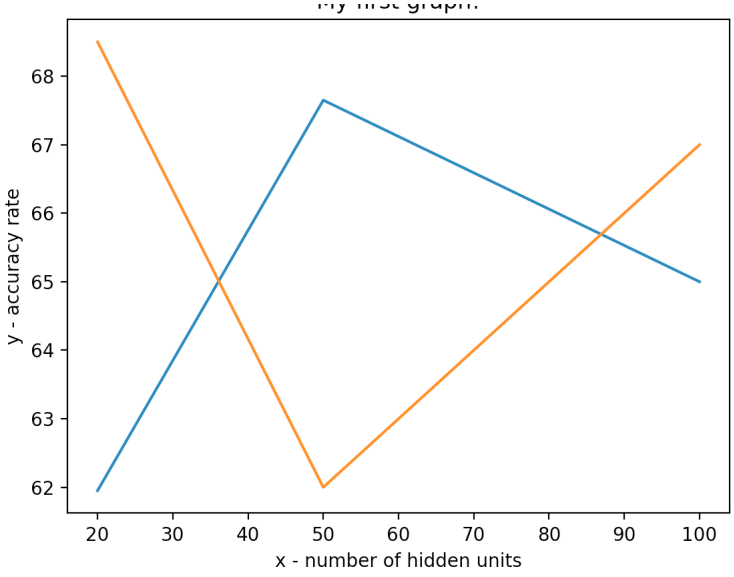

Figure 2: Experiment with number of hidden units

In figure 2, overfiting happened when number of hidden units reached 100 dimension. Otherwise, we can observe that the accuracy get low at number = 20 position, hence, from experiment, the best number of hidden units is approximately 50 with accuracy up to 68%. Moreover, after experiments, I reckon that number of hidden units is not correlation epochs because I figure out that big epochs will always improve the accuracy, on the other hand, number of hidden units will cause the overfiting problem .In conclusion, we should predict the apropriate number of hidden units in order to avoid overfiting problem.

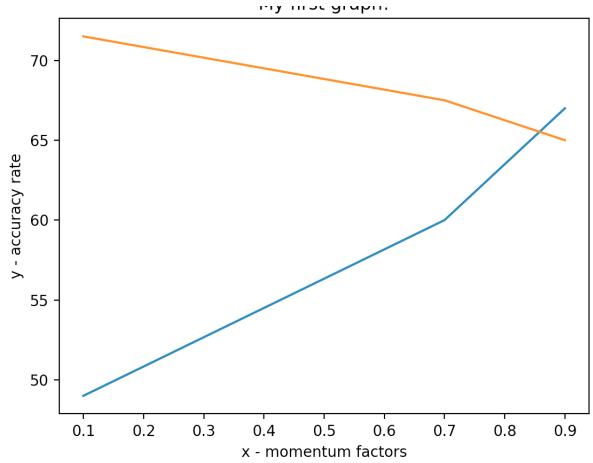

Figure 3: Experiment with momentum factors

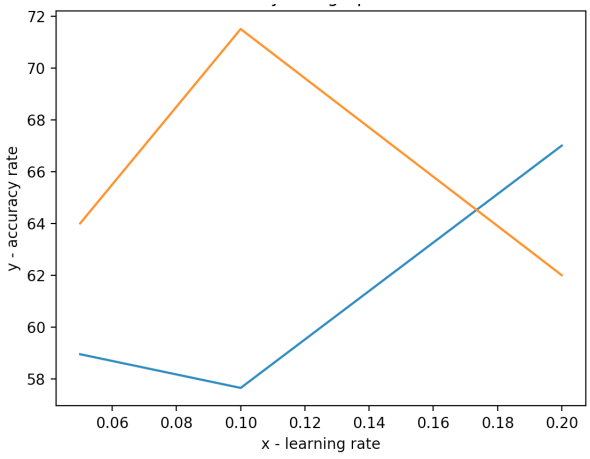

Figure 4: Experiment with learning rate

My observation of experiments with learning rate and momentum factors is that the small learning rate along with big momentum factor is likely to increase the accuracy and it proves that momentum gradient decent is efficient although it depends on the initial weight in some cases. Besides, I infer that there is inverted correlation between learning rate and epoch since if the learning rate is small, we will need big epochs to converge the weight.

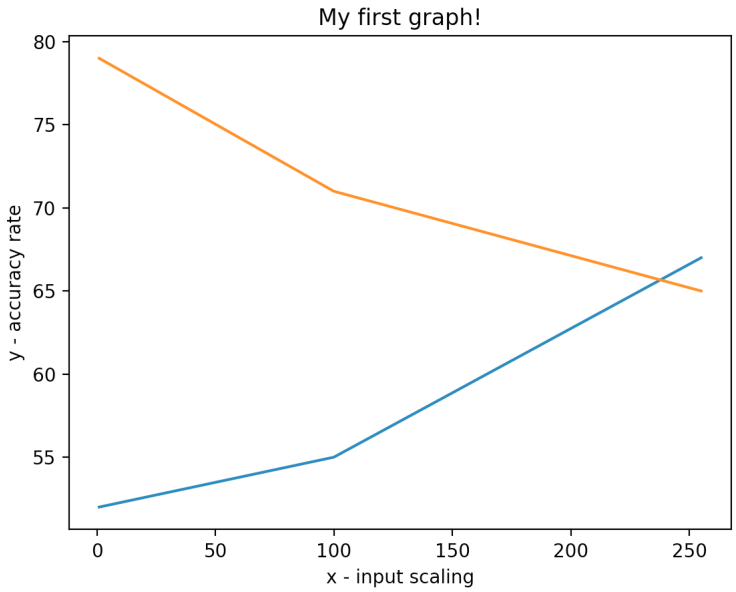

Figure 5: Experiment with number of input scaling

In final experiment with input scaling, it seems that the bigger scaling will affect a lot to training implementation as the accuracy get absolutely bad with input scaling from 0 to 1. I thing it's obvious because the bigger input scaling could save more information of images.

6 Conclusion

After implementation and experiments, MLP is efficient for data classification but we can improve the accuracy and runtime by using other gradient decent such as mini-batch gradient decent, stochastic gradient decent. Furthermore, we can see the correlation through observing the different results by adjusting factors. Hence, in my implementation, the optimal option to reach highest accuracy of 72% is: learning rate = 0.1, momentum = 0.9, epochs = 300 and number of hidden units = 50.