CENG 463 - Homework 3

Analyze on Classification Mnist Dataset with MLP

To correctly classify the Mnist data set I have used a network with the following properties;

Layer info = [784, 80, 32, 10], Learning rate = 0.001, Batch size = 1000, Number of epochs = 120

Activation function = Sigmoid, Loss = Cross Entropy

The network has been able to correctly classify 0.95% of images on the test dataset and it gave the best result among the results of the other networks I have tried. Since it has sufficient number of neurons and hidden layers it doesn't cause any overfitting and underfitting problem.

Initially the weights were randomly assigned big numbers and they were all positive. Because of that when these weights are multiplied along the layers, they caused a large change in the cost. Thus, the gradients were also large. This caused oscillating around the minima and the network didn't learn. To solve this problem weights are initialized in range [-1,1]. Besides this, in order to increase the accuracy, the images have been normalized around the x=y=0 point. Also, minibatches have been used to accelerate the learning process.

Some other methods can be used to increase the accuracy. More data will enable to a better learning, so we can increase the size of the dataset. Also, we can increase the hidden layer size and we can use dropout to prevent overfitting problem. Regularization parameter can be added to the loss function.

During the development there was a trade-off in between learning rate and the speed of learning. We see examples of it in the below graphs. The learning rate of the second graph is bigger than the first one and it has reached an accuracy of 90% when epoch is 40, while the first one reached an accuracy of 85% when the epoch is 40. However, if we continue to increase the learning rate, the accuracy lines become more zigzag like shapes and again we need more epochs to get good results.

Also, when we increase the model complexity the model is getting more likely to overfit but if we decrease it too much than, it becomes to underfit.

One of the most important things was choosing the right hyperparameters for this problem. I have conducted several experiments to do it.

I have used sigmoid because it is widely used for similar tasks. Also, if we have large number of nodes and layers the computations of activation functions becomes important and the derivation of sigmoid function is very simple.

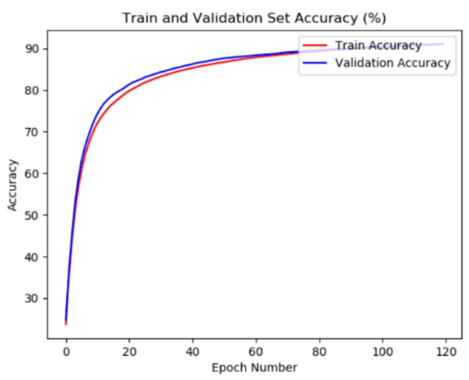
I have used cross entropy for loss function because of it is commonly used for classification.

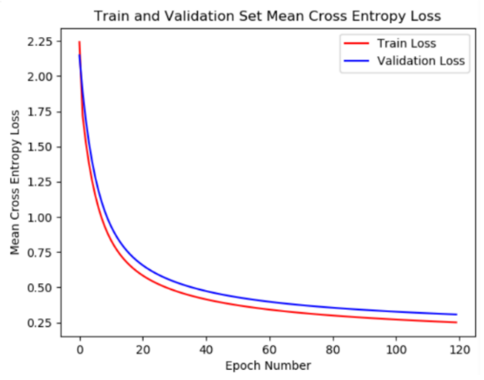
Layer info = [784, 80, 32, 10]

Learning rate = 0.0001

Test accuracy = 0.908

Test cross entropy loss = 0.306



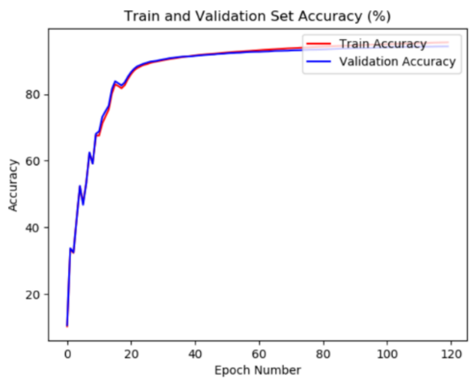


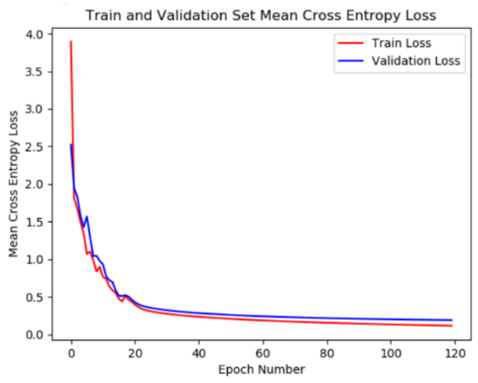
Layer info = [784, 80, 32, 10]

Learning rate = 0.0005

Test accuracy = 0.94

Test cross entropy loss = 0.194



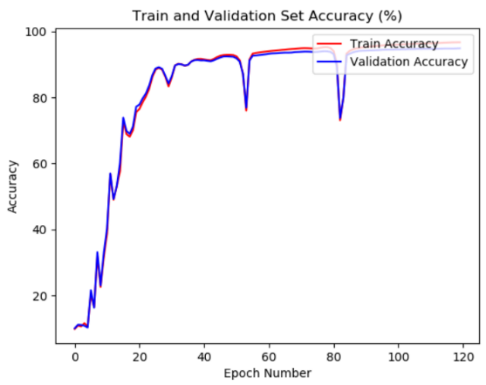


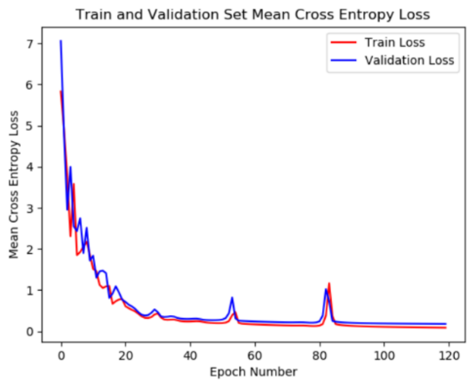
Layer info = [784, 80, 32, 10]

Learning rate = 0.001

Test accuracy = 0.95

Test cross entropy loss = 0.18



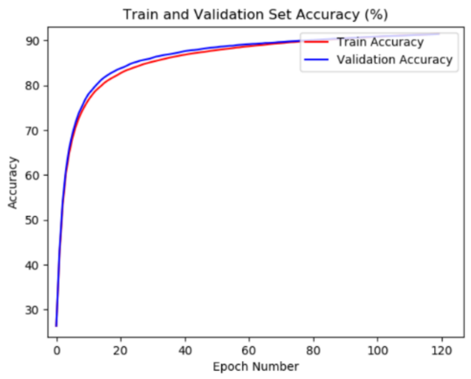


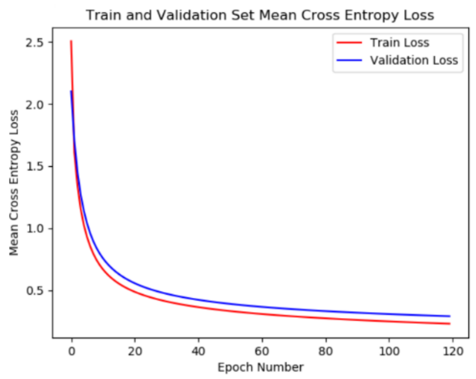
Layer info = [784, 100, 50, 10]

Learning rate = 0.0001

Test accuracy = 0.908

Test cross entropy loss = 0.306



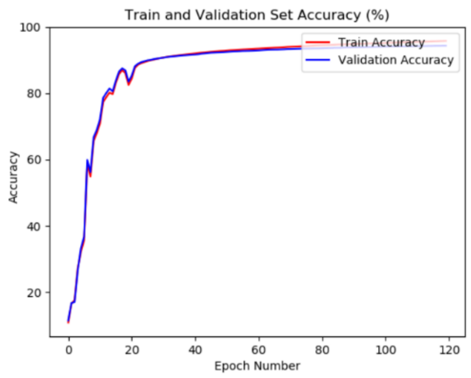


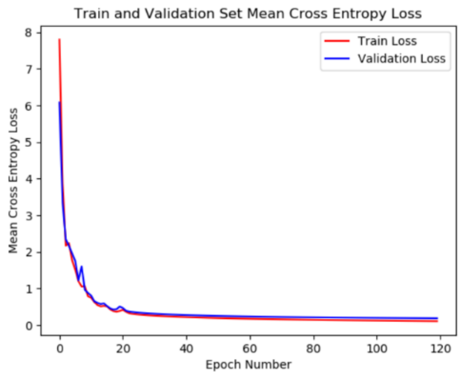
Layer info = [784, 100, 50, 10]

Learning rate = 0.0005

Test accuracy = 0.94

Test cross entropy loss = 0.18



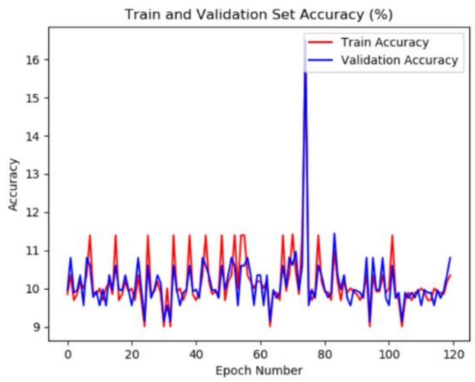


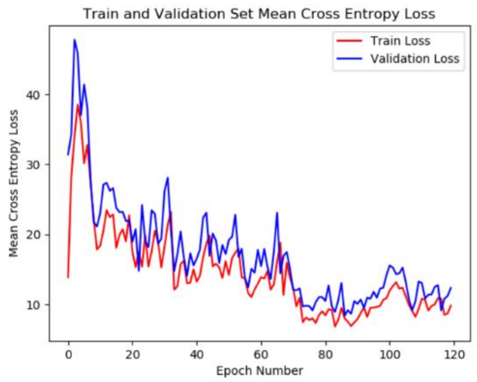
Layer info = [784, 100, 50, 10]

Learning rate = 0.001

Test accuracy = 0.1

Test cross entropy loss = 12.34



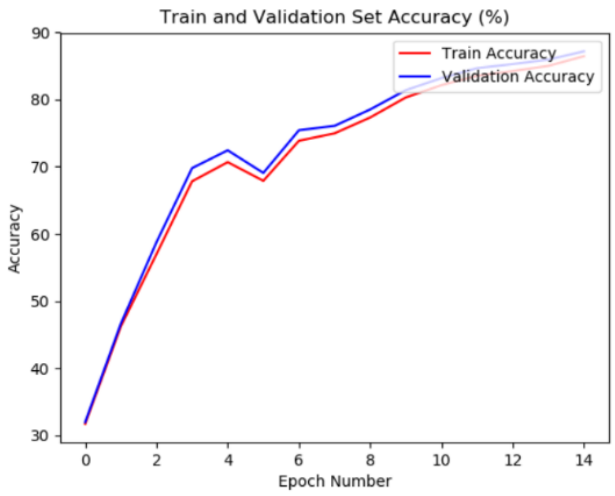


Because the learning rate is too big, divergence occurred.

Underfitting occurred when:

Layer info = [784, 15, 10]

Learning rate = 0.001



Nisa Pınar Rüzgar

220201050