Q1

Let denote $u_i = f(x;w)_i$

The derivative $\frac{\partial L(\hat{y_i}, y_i)}{\partial \hat{y_i}}$ depends of the loss function.

The derivative $\frac{\partial u_j}{\partial w}$ depends on f function.

$$\frac{\partial L(\hat{y_i}, y_i)}{\partial w} = \frac{\partial L(\hat{y_i}, y_i)}{\partial \hat{y_i}} \frac{\partial \hat{y_i}}{\partial u_j} \frac{\partial u_j}{\partial w}$$

Let's see $\frac{\partial \hat{y}_i}{\partial u_j}$

For $i = j$:

$$\frac{\partial \hat{y_i}}{\partial u_j} = \frac{\partial \frac{e^{u_i}}{\sum_{j=0}^{n} e^{u_j}}}{\partial u_j} = \frac{e^{u_i} \sum_{j=0}^{n} e^{u_j} - e^{u_i} e^{u_j}}{\left(\sum_{j=0}^{n} e^{u_j}\right)^2} = \frac{e^{u_i} \sum_{j=0}^{n} e^{u_j} - e^{u_j}}{\left(\sum_{j=0}^{n} e^{u_j}\right)\left(\sum_{j=0}^{n} e^{u_j}\right)}$$
$$= \hat{y_i}(1 - \hat{y_i})$$

For $i \neq j$:

$$\frac{\partial \hat{y_i}}{\partial u_j} = \frac{\partial \frac{e^{u_i}}{\sum_{j=0}^{n} e^{u_j}}}{\partial u_j} = \frac{0 - e^{u_i} e^{u_j}}{\left(\sum_{j=0}^{n} e^{u_j}\right)^2} = \frac{-e^{u_i} e^{u_j}}{\left(\sum_{j=0}^{n} e^{u_j}\right)\left(\sum_{j=0}^{n} e^{u_j}\right)} = -\hat{y_i}\hat{y_j}$$

Q2 part 1:

The model I used for the first practical part is a model containing one hidden layer.

I first initialize the hidden layer parameters (w1, b1, w2, b2)

w1, w2 for the weight’s vectors of the data.

b1, b2 for the data biases.

The model will repeatedly calculate the forward propagation using the dot product of the data and the weight vectors then adding the bias.

The result will then go through the hyperbolic tangent function for the train data and through the softmax function for the test data.

Then the model will calculate the cost using $-\frac{1}{m}\sum_{i=1}^{m} y_i * \log\left(softmax(z_2)\right)$

We then find the gradient using backward propagation principle and update the parameters accordingly.

This whole process is basically the creation of the neural network with one hidden layer.

Then we calculate the accuracy of the predictions using a forward propagation as learned in class. We extract the arg max of each of the ten possible classes and set the prediction to it. We then calculate the equivalency rate of the real classes and the ones we extracted. Thus, finding the accuracy level.

The Evaluate file will use the weights we found in the test section to find the accuracy rate. Just as above.

Q2 part 2:

In this part we will try to overfit the model with random labels.

We will generate 128 binomial (p=0.5) random variables. We will use torch to create a neural network. Run a great amount of iteration to train our model using the Adam optimization function from Torch library. We will calculate a loss based on the cross-entropy loss at each of the iterations as the amount of iteration.

And return this trained model.

We will calculate the train loss and the temp loss using the cross-entropy loss too.

And then we print our graphs of the losses as required.

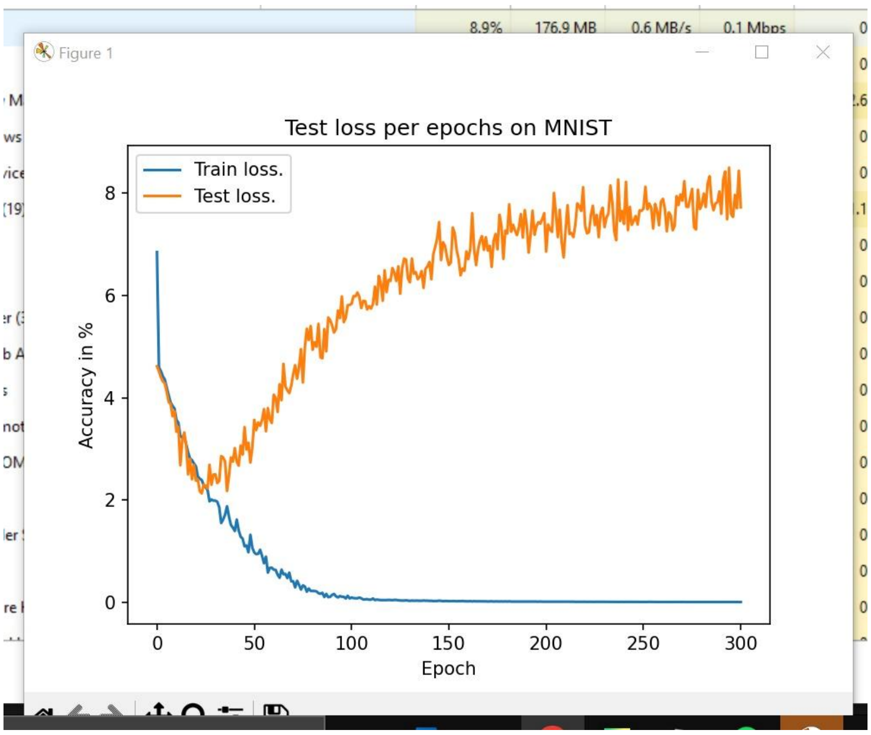
Figure 1
Test loss per epochs on MNIST
Train loss.
Test loss.
Accuracy in %
Epoch