Deep Learning Lab
Exercise 1:Feed Forward Neural Network

Manav Madan (30.10.2018)

Introduction:

In this exercise, a multi layer perceptron model of neural network was build, which involved working with The MNIST database of handwritten digits. The structure involves basic set of functions such as forward propagation (as fprop) and back propagation (as bprop) for each layer. The task was to train the model and test it by further optimizing the different parameters such as learning rate and activation function.

Structure:

The network is designed with the help of Layer class object which further uses classes such as InputLayer,FullyConnectedLayer and SoftmaxOutput and in total 5 layers are formed. After appending these layers to layer object, the initial weights are drawn randomly from normal distribution (with mean:0 and standard deviation:variable) and the bias is defined as empty array of zeros for the network. Initially there are four activation functions (Linear, Sigmoid, Relu and Tanh) and their derivatives defined which could be easily accessed with the class Activation. Predictions are made using forward propagation, which is just a bunch of matrix multiplications and the application of the activation function(s). Weight updation is done either with stochastic gradient approach (Number of Batches: 782 with each batch containing 64 samples).

Problems Faced:

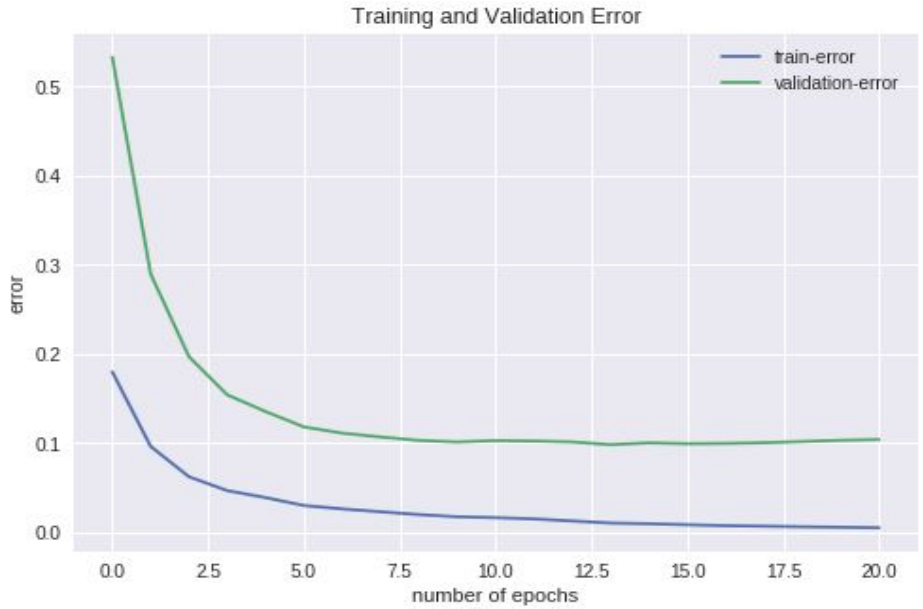
There were many errors which were faced during the training phase, here are some defined which must be discussed in detail.

1) ValueError: operands could not be broadcast together with shapes (50000,250) (64,250)
   In Class Fullyconnected layer, for the function bprop, for accumulated gradient with respect to bias needs to be updated/returned, instead of just returning the gradient we need to sum the gradient and take the average. Instead of just returning gradient :self.db = act_grad : line number 187.Use Accumulated bias :self.db = np.sum(curr_act_grad, axis=0)/n.

2) CompileError: In class SoftmaxOutput, for The gradient of Softmax output layer, the derivative Softmax_grad is needed to be normalized by the number of Samples.
   Instead of softmax_grad = (Y_pred - Y) , use (Y_pred - Y) /(Y.shape[0])

Results:

In the initial training. The number of units in the initial fully connected layer were 100 (Activation function = relu) and in the second fully connected layer was 100 (Activation Function = relu) and for the last one a total number of 10 units (Activation Function = None). The final loss is calculated with SoftmaxOutput class which has the multi class cross-entropy loss defined. Initially the learning rate is defined as 0.1 and time taken for training is equal to 99.6s.



In the final training with full data set, the number of units in the initial fully connected layer were 250 (Activation function = relu) and in the second fully connected layer was 50 (Activation Function = tanh) and for the last one a total number of 10 units (Activation Function = None). The final loss is calculated with SoftmaxOutput class which has the multi class cross-entropy loss defined. The learning rate was changed to 0.2 and time taken for training is equal to 212.9s. With 20 epochs the final training error was 0.05% and the validation error which is also the test error here was of 1.92%