CS Assignment 1 Readme

Mohammad Afroz Alam

1 Code Explaination

1.1 mlp.py

Contains a class MLP to build and train multi-layer perceptrons. It is used in the following way:

```
1. init : mlp = MLP(nhidden_layers, nnodes, activation_function)
        nhidden_layers      : (int) number of layers
        nnodes              : (list) number of nodes per layer
        activation_function : (str) non-linear function applied after affine transformation

2. forward : mlp.forward(input_vector)
        input_vector : (numpy array) a 1-D array of size 784

        returns the probability for each of the 10 classes

3. cross_entropy_loss : mlp.cross_entropy_loss(target)
         target       : (int) integer storing the correct class

         returns cross entropy loss of our predicted model and target

4. backward : mlp.backward(target)
          target : (int)[optional] integer storing the class (if no argument is given, then target v

          returns derivative of loss with respect to input

5. updateParams : mlp.updateParams(hyperParams, optimizer)
        This function is used for updating parameters.
        Expects the following input
        hyperParams : (list) a list of hyperParams for the type of optimizer
        optimizer   : (str) 'momentum' or 'adam'
            momentum:
             Takes the following params as a list:
             learningRate : hyperParams[0]
             gamma        : hyperParams[1]

            adam:
             Takes the following params as a list:
```

```
             beta1 : hyperParams[0]
             beta2 : hyperParams[1]
             alpha : hyperParams[2]

6. plot_grads : mlp.plot_grads(fname, input_vector, target)
         fname        : (str) filename to save the plotted gradients
         input_vector : (array)[optional] input for forward pass
         target       : (int)[optional] for loss calculation

         this function plot a graph between numerical gradients and calculated gradients(using backp

7. train : mlp.train( train_data, train_labels, val_data, val_labels, config)
          train_data   : (array) a 2-D array with shape (trainsize, 784) to train MLP
          train_labels : (array) a 1-D array with shape (trainsize, ) to train MLP
          val_data     : (array) similar to train_data
          val_labels   : (array) similar to train_labels
          config : a dictionary with following entries
                 filename   : name of file to store results
                 batchSize  : batchSize for minibatch
                 max_epochs : number of epochs to run
                 optimizer  : 'momentum' or 'adam'
                      hyperParams:
                        momentum:
                          learningRate : hyperParams[0]
                          gamma        : hyperParams[1]
                        adam:
                          beta1 : hyperParams[0]
                          beta2 : hyperParams[1]
                          alpha : hyperParams[2]

8. test : mlp.test( test_data, test_labels)
          test_data   : (array) a 2-D array with shape (testsize, 784) to train MLP
          test_labels : (array) a 1-D array with shape (testsize, ) to train MLP
```

1.2 main.py

Contains configurations for different models and used to run the classifier, plot gradients, etc.

2 Experiments

A total of 8 experiments have been performed using the following combinations of:

1. Number of hidden layers (2)[1, 2]
2. Activation function (2)[tanh and relu]
3. Optimazation algorithm (2)[momentum and adam]

Accuracy achieved on the testset in each of the above mentioned combination is listed below:

Number of Hidden Layers	Activation Function	Optimizer	Accuracy
1	Tanh	Adam	97.96%
	ReLU	Adam	97.44%
	Tanh	Momentum	94.28%
	RelU	Momentum	95.80%
2	Tanh	Adam	97.39%
	ReLU	Adam	97.43%
	Tanh	Momentum	96.23%
	RelU	Momentum	96.11%

Table 1: Accuracy for different architectures run for 10 epochs

2.1 Gradient Checking

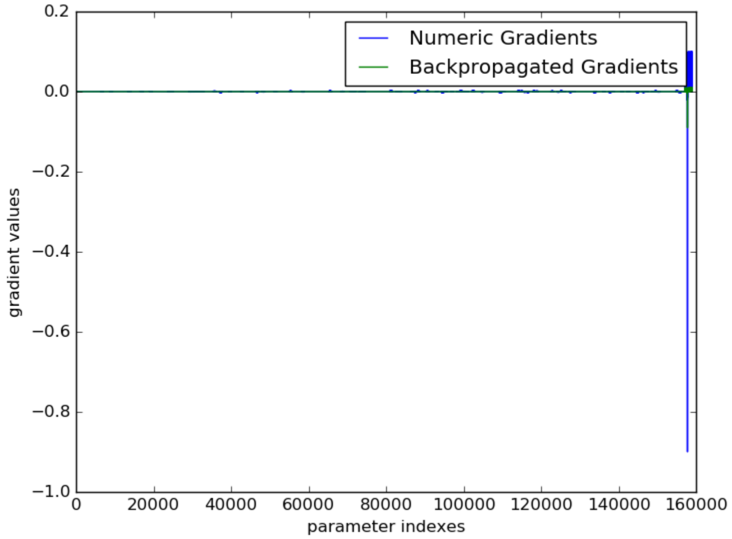

Figure 1: Backpropaged and Numerical Gradients

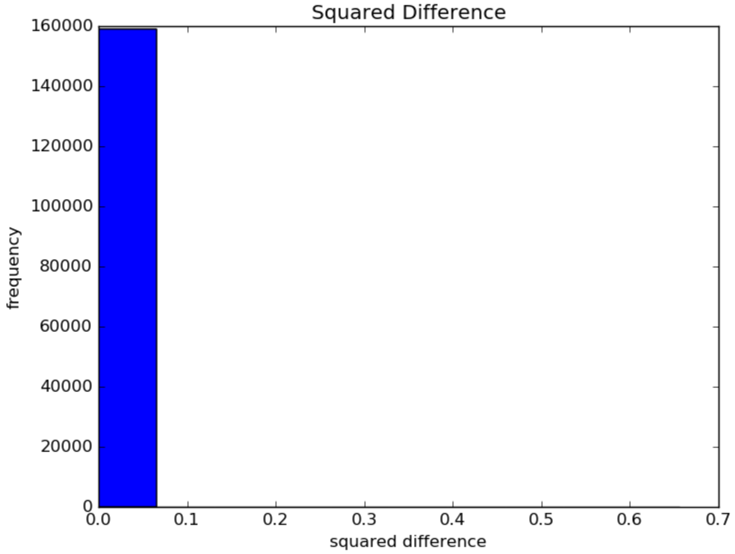

Figure 2: Histogram of squared error between Backpropaged and Numerical Gradients