CMU Deep Learning 11-785 Spring 2018

Homework 1

For part 1 of the homework you will write your own implementation of the backpropagation algorithm for training your own neural network. You are required to do this assignment in the Python (Python version 3) programming language. Do not use any audodiff toolboxes (TensorFlow, Keras, PyTorch, etc) - you are allowed to use only numpy like libraries (numpy, pandas, etc).

The goal of this assignment is to label images of 10 handwritten digits of “zero”, “one”,..., “nine”. The images are 28 by 28 in size (MNIST dataset), which we will be represented as a vector x of dimension 784 by listing all the pixel values in raster scan order. The labels t are 0,1,2,...,9 corresponding to 10 classes as written in the image. There are 3000 training cases, containing 300 examples of each of 10 classes, 1000 validation (100 examples of each of 10 classes), and 3000 test cases (300 examples of each of 10 classes). The can be found in the file digitstrain.txt which has been posted on Piazza.

The way you choose to design your code for this homework will affect how much time you spend coding. We recommend that you look through all of the problems before attempting the first problem. A good foundation will make the rest of these problems easier.

Submissions will be graded through autolab. Your submission must be titled handin.tar and your python submission file contained within handin.tar must be labeled hw1.py or else the autograding system will not properly recognize your code. Additionally, your tar file must have hw1.py contained within a hw1 folder.

PROBLEM 1:

Here you must read an input file. Each line contains 785 numbers (comma delimited): the first values are between 0.0 and 1.0 correspond to the 784 pixel values (black and white images), and the last number denotes the class label: 0 corresponds to digit 0, 1 corresponds to digit 1, etc. As a warm up question, load the data.

For this problem you must write a function that takes a file path as an argument which contains this data. Your function must return two values (X and Y) that contains the data from the file as described. Specifically, the first return value (X) must be a matrix where the rows are individual examples of images, and the columns are individual pixels (N x 784 matrix). The second return value must be a list/array of real numbers representing the labels of the examples (rows) in X.

eg:

1.0,0.0,1.0,0.0,....0.0,0.25,0.0,0.0
...
1.0,0.0,1.0,0.0,...,1.0,0.0,0.0,0.96776

X = [
[1.0,0.0,1.0,0.0,....0.0,0.25,0.0,0.0]
...
[1.0,0.0,1.0,0.0,...,1.0,0.0,0.0,0.96776]

]

Y = [5,...,2]

```
def load_data(filePath):
        X = []
        Y = []
        #INSERT YOUR CODE HERE
        return X, Y
```

PROBLEM 2:

Implement the backpropagation algorithm in a zero hidden layer neural network (weights between input and output nodes). The output layer should be a softmax output over 10 classes corresponding to 10 classes of handwritten digits (e.g. an architecture: 784 > 10). Your backprop code should minimize the cross-entropy entropy function for multi-class classification problem. This step should be done with a full step of gradient descent, not SGD or RMSProp.For this problem you must write a function that takes as an input a matrix of X values, a list of Y values (as returned from problem 1), a weight matrix, and a learning rate and performs a single step of backpropagation. You will need to do both a forward step with the inputs, and then a backward prop to get the gradients. Return the updated weight matrix and bias in the same format as it was passed.

The list of weight matrices will be a list with 1 entry where the only entry is a matrix in the format where the rows represent all of the outgoing weights for a neuron in the input layer and the columns represent the weights for the incoming neurons. A specific row column index will give you the weight for a neuron to neuron connection.

The list of bias vectors will be in the form where each entry in the list is a vector with the same length as the first set of weights. (e.g. for an architecture of 784 > 10, there will be a single element list with a vector of size 10)

```
def update_weights_perceptron(X, Y, weights, bias, lr):
        #INSERT YOUR CODE HERE
        return updated_weights, updated_bias
```

PROBLEM 3:

Extend your code from problem 2 to support a single layer neural network with N hidden units (e.g. an architecture: 784 > 100 > 10).

For this problem you must write a function that takes as an input a matrix of X values, a list of Y values (as returned from problem 1), list of weight matrices, a list of bias vectors, a list of bias vectors, and a learning rate and performs a single step of backpropagation. You will need to do both a forward step with the inputs to get the outputs, and then a backward prop to get the gradients. Return the updated weight matrix and bias in the same format as it was passed.

The list of weight matrices is a list with 2 entries where each entry in the list contains a single weight matrix as previously defined in problem 2. For a network with shape 784 > 100 > 10 the passed list of weight matrices would look like this: [Matrix with shape 784x100, Matrix with shape 100x10]. Note: Though a hidden layer of size 100 is used as an example here, your code must be able to support a hidden layer of dimension N.

The list of bias vectors will be in the form where each entry in the list is a vector with the same length as the first set of weights. (e.g. for an architecture of 784 > 100 > 10, there will be a two element list with an vector of size 100 and a vector of size 10)

```
def update_weights_single_layer(X, Y, weights, bias, lr):
    #INSERT YOUR CODE HERE
    return updated_weights, updated_bias
```

PROBLEM 4:

Extend your code from problem 3 (use cross entropy error) and implement a 2-layer neural network, starting with a simple architecture containing N hidden units in each layer (e.g. with architecture: 784 > 100 > 100 > 10).

For this problem you must write a function that takes as an input a matrix of X values, a list of Y values (as returned from problem 1), list of weight matrices, a list of bias vectors, and a learning rate and performs a single step of backpropagation. You will need to do both a forward step with the inputs to get the outputs, and then a backward prop to get the gradients. Return the updated weight matrix and bias in the same format as it was passed.

The list of weight matrices is a list with 3 entries where each entry in the list contains a single weight matrix as previously defined in problem 2. For a network with shape 784 > 100 > 100 > 10 the passed list of weight matrices would look like this: [Matrix with shape 784x100, Matrix with shape 100x100, Matrix with shape 100x10]. Note: Though a hidden layer of size 100 is used as an example here, your code must be able to support a hidden layer of dimension N.

The list of bias vectors will be in the form where each entry in the list is a vector with the same length as the first set of weights. (e.g. for an architecture of 784 > 100 > 10, there will be a two element list with an vector of size 100 and a vector of size 10)

```
def update_weights_double_layer(X, Y, weights, bias, lr):
    #INSERT YOUR CODE HERE
    return updated_weights, updated_bias
```

PROBLEM 5:

Extend your code from problem 4 to implement different activations functions which will be passed as a parameter. In this problem all activations (including the final layer) must be changed to the passed activation function

```
def update_weights_double_layer_act(X, Y, weights, bias, lr, activation):
    #INSERT YOUR CODE HERE
    if activation == 'sigmoid':
        #INSERT YOUR CODE HERE
    if activation == 'tanh':
        #INSERT YOUR CODE HERE
    if activation == 'relu':
        #INSERT YOUR CODE HERE
    #INSERT YOUR CODE HERE
    return updated_weights, updated_bias
```

PROBLEM 6:

Extend your code from problem 5 to implement momentum with your gradient descent. The momentum value will be passed as a parameter. Your function should perform “epoch” number of epochs and return the resulting weights.

```
def update_weights_double_layer_act_mom(X, Y, weights, bias, lr, activation, momentum, epochs):
    #INSERT YOUR CODE HERE
    if activation == 'sigmoid':
        #INSERT YOUR CODE HERE
    if activation == 'tanh':
        #INSERT YOUR CODE HERE
    if activation == 'relu':
        #INSERT YOUR CODE HERE
    #INSERT YOUR CODE HERE
    return updated_weights, updated_bias
```