Programming Assignment2: Neural Networks

Problem :. In this homework assignment, your task is to implement one of the common machine learning algorithms: Neural Networks. You will train and test a neural network with the dataset we provided and experiment with different settings of hyper parameters. This assignment also includes a written part which is to help you understand how to train a neural net and its solution will be used to test your code as it is not easy to debug neural networks.

Note: For this assignment, you should not use any machine learning libraries. You should implement the neural network learning algorithm yourself. You can use any Python standard library and the following third-party libraries:

- 1. numpy: for creating arrays and using methods to manipulate arrays.
- 2. matplotlib: for making plots
- 3. pandas or other packages for visualizaing data or graphs

Dataset

Figure 1: Sample images in MNIST

The dataset we use for this assignment is the MNIST database. It is a database of handwritten digits. Images for 100 samples/digits are given in the above figure. We have split the date set into training set and test set stored in five csv.gz files. The dataset can be downloaded on the course website.

Training samples:
http://www.cs.princeton.edu/courses/archive/fall16/cos402/ex/TrainDigitX.csv.gz.
Training labels:
http://www.cs.princeton.edu/courses/archive/fall16/cos402/ex/TrainDigitY.csv.gz.
Test samples:
http://www.cs.princeton.edu/courses/archive/fall16/cos402/ex/TestDigitX.csv.gz.
Test labels for samples in TestDigitX.csv.gz:
http://www.cs.princeton.edu/courses/archive/fall16/cos402/ex/TestDigitY.csv.gz.
More test samples(no labels are provided for samples in the second test set.):
http://www.cs.princeton.edu/courses/archive/fall16/cos402/ex/TestDigitX2.csv.gz.

There are 50,000 training samples in TrainDigitX.csv.gz, 10,000 test samples in TestDigitX.csv.gz, and another 5,000 test sample in TestDigitX2.csv.gz. Each sample is a handwritten digit represented by a 28 by 28 greyscale pixel image. Each pixel is a value between 0 and 1 with a value of 0 indicating white. Each sample used in the dataset (a row in TrainDigitX.csv, TestDigitX.csv, or TestDigitX2.csv.gz) is a feature vector of length 784(28x28=784). TrainDIgitY.csv.gz and TestDigitY.csv.gz provide labels for samples in TrainDigitX.csv.gz and TestDigitX.csv.gz, respectively. The value of a label is the digit it represents, e.g, a label of value 8 indicates the sample represents the digit 8.

Note: The data files are compressed. But you do not need to upcompress them. The loadtxt() method from numpy can read compressed or uncompressed csv files.

Part 1: Manually update weights and biases for a small neural network

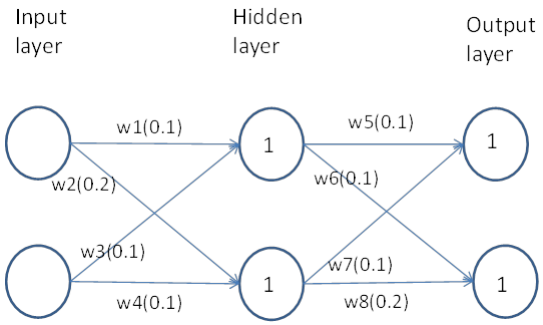

Figure 2: A Small Neural Net

Before designing and writing your code, you should work on the above neural network by hand. The small network has two neurons in the input layer, two neurons in the hidden layer and two neurons in the output layer. Weights and biases are marked on the figure. Note that there are no biases for the neurons in the input layer. There are two training samples: X1=(0.1, 0.1) and X2=(0.1, 0.2). The label for X1 is 0, so the desired output for X1 in the output layer should be Y1=(1,0). The label for X2 is 1, so the desired output for X2 in the output layer should be Y2=(0,1). You will update the weights and biases for the this small neural net using stochastic gradient descent with backpropagation. You should

use batch size of 2 and a learning rate of 0.1. You should update all the weights and biases only once MANUALLY. Show your work and results. And then use this as a test case to test your code which you implement on Part 2, train this network for 3 epochs and output the weights and biases after each epoch. These updated weights and biases should be outputed by running your code. It is imperative you do this part by yourself to make sure you have fully understood backpropagation.

Part 2: Implement a neural network learning algorithm

Your task is to implement a a neural network learning algorithm that creates a neural network of specified size and runs stochastic gradient descent on a cost function over given training data. The network you are working on in this assigment has 3 layers: one input layer, one hidden layer and one output layer. You should name your main file as `neural_network.py` which accepts seven arguments. The grader will run your code on the command line in the following manner:

>python `neural_network.py` NInput NHidden NOutput TrainDigitX.csv.gz TrainDigitY.csv.gz TestDigitX.csv.gz PredictDigitY.csv.gz

Your code should then train a neural net(NInput: number of neurons in the input layer, NHidden: number of neurons in the hidder layer, NOutpuy: number of neurons in output layer) using the training set TrainDigitX.csv.gz and TrainDigitY.csv.gz, and then make predictions for all the samples in TestDigitX.csv.gz and output the labels to PredictDigitY.csv.gz.

You should set the default value of number of epochs to 30, size of mini-batches to 20, and learning rate to 3 respectively.

Note: you can use numpy.loadtxt() and numpy.savetxt() methods to read from or write to a csv.gz file.

The nonlinearity used in your neural net should be the basic sigmoid function.

$$\sigma_{\boldsymbol{w},b}(\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}.\boldsymbol{x}+b)}}$$

We will be using mini-batches to train the neural net for several epochs. Mini-batches are just the training dataset divided randomly into smaller sets to approximate the gradient.

The main steps of training a neural net using stochastic gradient descent are:

- Assign random initial weights and biases to the neurons. Each initial weight or bias is a random floating-point number drawn from the standard normal distribution (mean 0 and variance 1) .
- For each training example in a mini-batch, use backpropagation to calculate a gradient estimate, which as you saw in class consists of following steps:
    1. Feed forward the input to get the activations of the output layer.
    2. Calculate derivatives of the cost function for that input with respect to the activations of the output layer.

3. Calculate the errors for all the weights and biases of the neurons using backpropogation.

- Update weights (and biases) using stochastic gradient descent:

$$w \to w - \frac{\eta}{m} \sum_{i=1}^{m} error_i^w$$

where m is the number of training examples in a mini-batch, $error_i^w$ is the error of weight w for input i, and η is the learning rate.

- Repeat this for all mini-batches. Repeat the whole process for specified number of epochs. `At the end of each epoch evaluate the network on the test data and display its accuracy.`

For this first part of the assignment, use the quadratic cost function.

$$C(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{2n} \sum_{i=1}^{n} \|f(\boldsymbol{x_i}) - \boldsymbol{y_i}\|^2$$

($\boldsymbol{w}$: weights, $\boldsymbol{b}$: biases, n: number of test instances, x_i: i^{th} test instance vector, y_i: test label vector, if the label for x_i is 8, then y_i will be [0,0,0,0,0,0,0,0,1,0], the ideal output of the trained neural network, f(x): function that the neural network guesses which maps input vector x to a label vector $f(x)$).

You should do the following tasks with your neural net

1. Create a neural net of size [784,30,10]. This network has three layers: 784 neurons in the input layer, 30 neurons in the hidden layer, and 10 neurons in the output layer. Then train it on the training data for 30 epochs, with a minibatch size of 20 and η = 3.0 and test it on the test data(TestDigitX.csv.gz).

   `Make plots of test accuracy vs epoch. Report the maximum accuracy achieved. Now run you code again with the second test set(TestDigitX2.csv.gz) and output your predictions to PredictDigitY2.csv.gz. In addition to your code, you should also upload both PredictTestY.csv.gz and PredictTestY2.gz to CS Dropbox.`

2. Train new neural nets of the original specifications (specifications in 1) but with η = 0.001, 0.1, 1.0 , 10, 100.

   `Plot test accuracy vs epoch for each` η `on the same graph. Report the maximum test accuracy achieved for each` η`. (remember to create a new neural net each time so its starts learning from scratch.)`

3. Train new neural nets of the original specifications but with mini-batch sizes = 1, 5, 10, 20, 100.

   `Plot maximum test accuracy vs mini-batch size. Which one achieves the maximum test accuracy. Which one is the slowest?`

4. Try different hyperparameter settings(number of epochs, η, and mini-batch size, etc.).

   `Report the maximum test accuracy you achieved and the settings of all the hyper parameters.`

Note: You should try your implementation on the small network given in Part 1 before running your code on the MNIST data set. When you run your code on the small neural net, you should initialize the weights and biases as given in Figure 2, use the two samples and the same learning rate and batch size when you do it by hand in Part 1.

`Then train this network for 3 epochs and output the weights and biases after each epoch.` (Make sure that the weights and biases outputed by the learned net after one epoch are the same as those you updated manually.)

Part 3: An alternate cost function

Now replace the quadratic cost function by a cross entropy cost function.

$$C(w,b) = -\frac{1}{n}\sum_{i=1}^{n} y_i ln\left[f(x_i)\right] + (1-y_i) ln\left[1-f(x_i)\right]$$

You only need to modify one function slightly to account for the change in the cost function if your code is well structured. `Train a neural net with the original specifications. What is the test maximum accuracy achieved?`

Part 4:L_2 regularization.(Optional, for bonus points)

Use L_2 regularizers on the weights to modify the cost function.

$$L_2 : C(w,b) = C_0 + \frac{\lambda}{2n}\sum_{w} w^2$$

`Train a neural net with the original specifications. What is the maximum test accuracy achieved with` L_2 `regularizer by a neural net of the original specifications?`

What and how to turn in:

- 1. Turn in hard copies in class on the due date.

  A printout of all your python **scripts**, **answers** and **plots** to questions in all sections. Plots should be made by python code or other software.

- 2. Upload your code and predictions to CS dropbox by the due date.

  Using this DropBox link,

  http://dropbox.cs.princeton.edu/COS402_F2016/Programming_Assignment2,

  Upload all your python **scripts** and the two prediction files: **PredictTestY.csv.gz** and **PredictTestY2.gz.** You should only turn in uncompressed .py files. All code should be working and well documented. If appropriate, a readme.txt file explaining briefly how your code is organized, what data structures you are using, or anything else that will help the graders understand how your code works.