CS440/ECE448
Homework/MP 7

Assigned 4/6/2017
Extra Credit Due 4/21/2017 11:59 PM
Due 4/23/2017 11:59 PM

In this homework/MP, you will be implementing a multi-class perceptron to classify a hand-written image as a digit, either 0, 1, 2, …, 9. In addition, you will use a neural network library to perform the same task using a multilayer perceptron (MLP), and you will be able to compare the results.

This assignment contains both a machine problem and a written component. You are welcome to use any programming language, but keep in mind that in the second part of this assignment you will be allowed to use a library. As such, we recommend that you choose a language that has a neural network library. If you choose Python, your options include (but are not limited to) sklearn, TensorFlow, and Keras.

In the ZIP file corresponding to this homework, we have provided you with labeled data which you can use to accomplish this task. The format of the images files is as follows. Each image spans 28 lines, and each line is 28 characters long (plus the new-line character). As such, the resolution for each image is 28x28. Each pixel can have three values: a background pixel, represented as whitespace ‘ ’, a foreground pixel, represented as a hash sign ‘#’, and a border pixel, represented as a plus sign ‘+’. To obtain a numeric vector for the image, use the value 1 for foreground pixels, 0 for background pixels, and 0.5 for border pixels. Therefore, the image vector should have 784 dimensions.

In the ZIP file, there are six files:

- **trainingimages**: This file contains a plain-text representation of each image in the training set.
- **traininglabels**: A label (an integer 0-9) corresponding to each image in the **trainingimages** file.
- **testimages**: Images that have been set aside for testing. The file follows the same format as **trainingimages**.
- **testlabels**: A label (an integer 0-9) corresponding to each image in the **testimages** file.
- **train.p**: If you choose to use Python, we have parsed the **trainingimages** and **traininglabels** files for you and provided a pickle containing the training images and labels. The object contained in this file is a list of tuples, where the first element is the 784-dimensional image vector and the second element is the integer label.
- **test.p**: The same as **train.p**, except for the test data.

Part 1: Conceptual Questions
To help you obtain an intuition of the problem, answer these conceptual questions first:

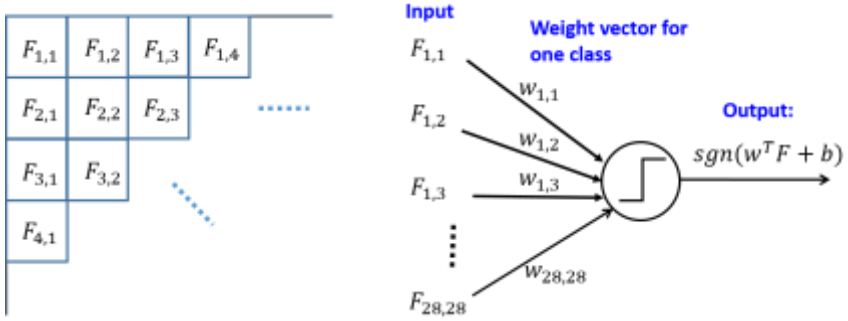
1) What is the example space, $\boldsymbol{X}$, for this problem?

2) Which model do you expect to be more expressive: a binary classifier perceptron or a binary classifier multi-layer perceptron with one hidden layer? Why?

3) Which one of the models described in question 2 do you think is more likely to overfit, and why?

4) Do you expect your perceptron to converge (assuming a constant learning rate)? Why or why not?

Part 2: Implementing a Perceptron
For this part, you must implement a non-differentiable multi-class perceptron from scratch, train it on the training data, and evaluate its results on the test data provided. You should use a one-vs-all approach. That is, you will train 10 different perceptrons, each one specialized to differentiate one digit from all the others. To predict the label for an image, you can return the class label whose perceptron has the highest confidence. Lastly, your perceptron should have a bias unit.



When training a perceptron, there are a few variables and settings that you must specify that, if chosen wisely, can lead to faster convergence or a better model. These include:

- The learning rate, α. You should try a constant learning rate and one that decays as $O\left(\frac{1}{n}\right)$, where n is the current epoch.
- How the training data is ordered: in random order vs. a fixed order. Often, randomizing the order of training examples each epoch will lead to a better model in the end.
- How the weights are initialized: all zeros or randomly.

To choose the right values for each setting listed above, you will need to separate your training data into a training set and a validation set. You can then use the training set to train the perceptron's weights and the performance on the validation set to choose which model you expect to perform best on unseen data. Alternatively, you can use cross-validation to choose these hyperparameters. After your training is complete, you should expect to achieve at least 80% accuracy on the test data.

1) Try adjusting each setting listed above, and report which combination (learning rate, training example ordering, and weight initialization) worked best. If you used random initial weights, specify what distribution the weights came from (uniform, Gaussian, or another distribution).

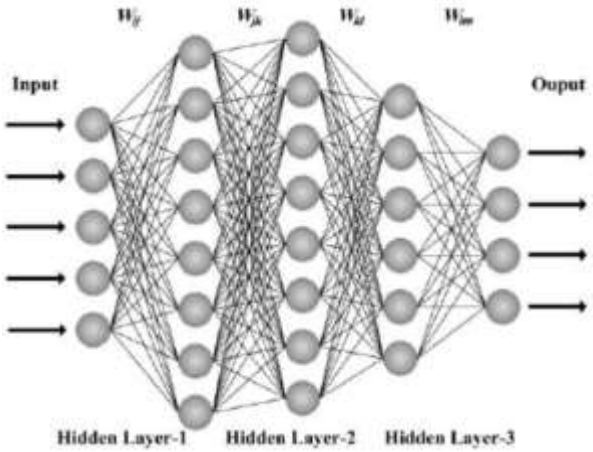
2) How many epochs were necessary to train your perceptron?

3) What accuracy did you achieve on the training set?

4) What accuracy did you achieve on the test set?

Part 3: Using a Multilayer Perceptron

For this part, you will use a library to implement a multi-layer perceptron. Like the single-layer perceptron in part 2, the multilayer perceptron has many hyperparameters that you must specify. These include:



- The number of hidden layers. The more hidden layers are in your model, the more expressive your model will be, but the more likely it will be to overfit.
- The number of neurons in each hidden layer. The same reasoning applies here.
- The activation function for each hidden layer. Common choices are:
  - Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
  - Hyperbolic tangent: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
  - Rectified Linear Unit (ReLU): $f(x) = \max\{0, x\}$

As with the perceptron, you should split your training data into training and validation data, the latter of which you can use to choose the optimal settings for these hyperparameters. After your model has been trained, you should expect to achieve approximately 90% accuracy or better on the test data.

1) Which combinations of hyperparameters did you try, and which combination worked best?

2) How many epochs were necessary to train your multilayer perceptron?

3) What accuracy did you achieve on the training set?

4) What accuracy did you achieve on the test set?

5) How did the results obtained using the multilayer perceptron compare to those obtained using the single-layer perceptron in part 2? Explain the differences.

Submission Instructions

When submitting your assignment, you should submit two files:

- A ZIP file containing the code you wrote for this assignment. You may choose to include the dataset, although you don't need to.
- A PDF containing your answers to the questions asked above.

Make sure that your PDF is uploaded as a separate file as opposed to packaged into your ZIP.