Implementing Deep Learning in a Neural Network

Fall 2017

Note: Items denoted with ☞ should be turned in with the written part of the homework.

You are to write your own neural network code and not (for example) simply use a neural network library such as TensorFlow!

Summary

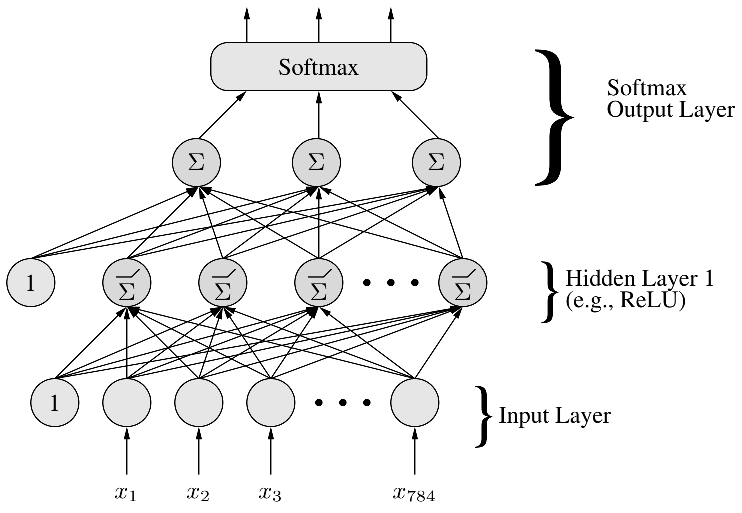
In this programming exercise you will code two deep-learning neural networks. The first comprises a single hidden layer with ReLU function and a softmax output layer, as represented in figure 1(a). The second comprises two hidden layers with ReLU functions and a softmax output layer, as represented in figure **??**(b). These neural networks are to be trained to perform digit recognition using the MNIST data set.

Background: The MNIST dataset

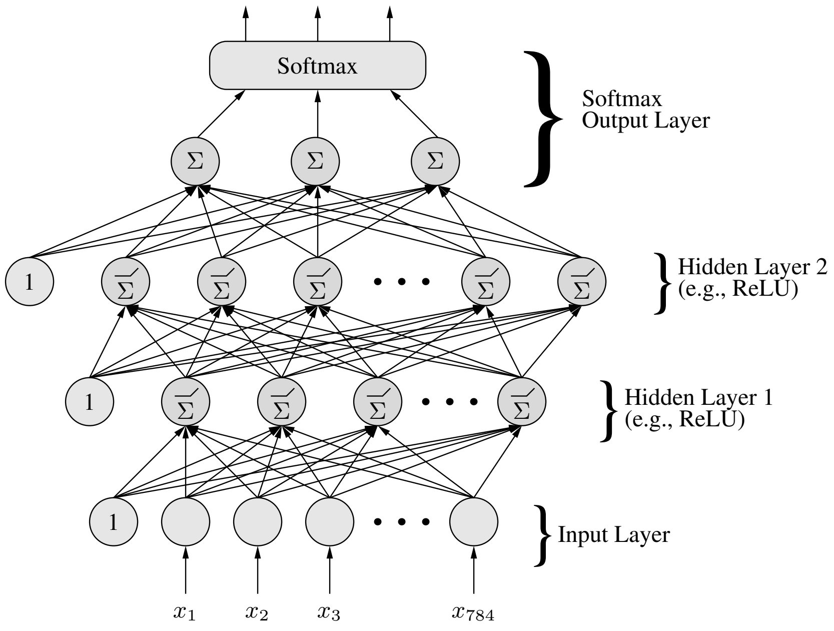
The MNIST dataset consists of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau. Each image is labeled with the digit it represents. Each image is 28×28 so that when vectorized it has dimension 784. The following python code reads in the dataset into training and testing vectors, and prints out some dimensions.

```
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt
import numpy as np


mnist = input_data.read_data_sets("/tmp/data/") # or wherever you want
                                                # to put your data

X_train = mnist.train.images
y_train = mnist.train.labels.astype("int")

X_test = mnist.test.images
y_test = mnist.test.labels.astype("int")

print("X_train.shape=",X_train.shape,"  y_train.shape=",y_train.shape)
print("X_test.shape=",X_test.shape,"  y_test.shape=",y_test.shape)

# plot one of these
some_digit = X_train[1]
some_digit_image = some_digit.reshape(28,28)

plt.imshow(some_digit_image, cmap = matplotlib.cm.binary,
            interpolation="nearest")
plt.axis("off")
```

☞ Plot a least one instance of each digit 0 – 9. Include the plots in your homework. (It will save paper to use subplots instead of a separate figure for each digit.)



(a) Single Hidden Layer



(b) Two Hidden Layers

Figure 1: Neural networks with one and two hidden layers

Programming Assignment

☞ Using python, write function(s) to perform backpropagation training on a neural network with one hidden layer. The hidden layer should use ReLU functions, and the output layer should be adjustable so that it can use different functions. In particular, you will need to be able to use the sigmoid function and the softmax function as output layer functions.

You should write your functions so that the number of inputs is a parameter, the number of neurons in the hidden layer is a parameter, and the number of outputs is a parameter.

Your function should perform training by backpropagation. You should also make it possible to incorporate momentum in your learning. You should also make it possible to do batch learning.

In light of the second part of this assignment, you may want to also write your code so that the number of layers is also a parameter, with settable parameters for each layer.

☞ **Testing on a two-class problem:** Using the data from the "Data-driven Classifiers" problem (the data from the file `classasgntrain1.dat`), divide the data into 80% training and 20% testing. This neural network has 2 input neurons (for the two-dimensional inputs). Create a neural network having a single hidden layer with 5 neurons. Since this is a two-class problem, use the sigmoid function as the output-layer function, with a single output.

Train your network using the training data. (Since the data set is so small, it does not make sense to do batch learning.) Try different values of the learning parameter (step size) until you are satisfied that it is doing the learning correctly. Test your network using the test data. Report on the probability of error. Discuss performance relative to classifiers you have implemented in the first assignment.

After you have your network successfully learning, introduce momentum learning with $\beta = 0.8$. Train and test the network. Comment on the difference that the momentum makes, if any.

☞ **Testing on a 10-class problem:** Now train your neural network using the training data from the MNIST database, using 300 neurons in the hidden layer. Obviously, for the 10 digits, you'll need to have ten neurons in the output layer, and you should use the softmax function in the output layer.

Use a batch of size 100. Use at least three different values of the step size paremeter. For each of these values of step size parameter, make a plot of the MSE as a function of iteration every 50 iterations. (You may need to experiment to determine how many iterations you need.)

☞ Test your neural network using the test data from the MNIST database. Report the probability of successful classification.

☞ Repeat the above using momentum learning with $\beta = 0.8$.

☞ Comment on how many iterations are needed, and how long (in clock time) it takes to train.

☞ Create a network with two hidden layers, where the first hidden layer has 5 neurons and the second hidden layer has 10 neurons. (If you didn't write your code in the first place to accomodate different numbers of layers, after completing this part you may want to re-factor your code so that it can now handle different numbers of layers.)

Using the data in `classasgntrain1.dat` as before, train and test the network and report on the probability of error. Comment on the difference in performance between the one-layer and the two-layer networks.

☞ Create a network with two hidden layers, with 300 neurons in the first (lowest) hidden layer and 100 neurons in the second hidden layer. Again, try different values of the step size parameter, plotting the MSE as a function of iteration. Use a minibatch of size 100. And again, test using the test data from the MNIST database.

☞ Train your network using momentum learning with $\beta = 0.8$, and repeat the above experiments.

☞ Turn in answers to questions, program listings, plots, comments, and observations.