COMP 135 – Machine Learning – Fall 2016

Empirical/Programming Assignment 3

Due date: Monday, 11/21 (by the beginning of class, both paper and electronically)

In this assignment you will the Backpropagation algorithm for neural networks and evaluate its performance with changing network architecture.

1 Data

In this assignment we use only one dataset for digit recognition which you can find on the course web page.[1] As in previous assignments we have already split the data into train and test portions. The files are in arff format so you should be able to use your reader from assignment 1 to read the data files. To be clear you can reuse your own code from a previous assignment but please do not use weka's code or code from any other source for that purpose.

2 Network structure and the backpropagation algorithm

In this assignment we apply a neural network to classification problems with more than two labels. To implement this we use multiple output units with "one hot" coding. For example, since our dataset has 10 possible labels we will have 10 output units. The encoding on label=2 is given by assigning 0100000000 to the corresponding output units. Your code should work on any arff dataset with numerical features. Therefore it should read the dataset to figure out the number of labels and output units to be used.

During training we use the binary labels as the required outputs of the corresponding units. During testing each output unit calculates a score for the example. This is given by the value x_i in the forward pass of the backpropagation algorithm in the slides. We then predict the label which has the highest score.

We will work with networks with d (depth) hidden layers each of width w. As in the slides there is an edge (and weight) between nodes in consecutive layers, the first hidden layer is connected to all input (features in the examples) and the output layer is connected to the last hidden layer. In the case where $d = 0$ the output layer is directly connected to the input layer, i.e., there are no hidden nodes. In this case each output node is a "perceptron-like" unit - it calculates a linear threshold decision boundary over the inputs, but it differs from perceptron in that it uses the sigmoid function and has a different update rule. In this way, by varying d and w we can explore the power of large networks over single linear units, and the ability of our algorithm to successfully learn such networks.

The (stochastic gradient descent variant of the) backpropagation algorithm is exactly as given in lecture slides. The pseudocode given there is directly applicable to multiple output units. As in that code, we go through the dataset sequentially and update the weights with each example. You should implement this algorithm repeating the linear scan over examples 200 times so that the overall code has this structure

```
learn(w,d,traindata,testdata)
Construct network with w,d and Initialize weights
Repeat 200 times
   For each example in traindata
       Update weights using backpropagation formulas
   End For
End Repeat
For each example in testdata
   Calculate scores of output units and evaluate preiction (correct/incorrect)
End For
Calculate error rate
```

[1]For more information on this dataset please see `https://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/`

The weights should be initialized with independent random numbers uniformly sampled in the range $[-0.1, 0.1]$. The learning rate should be $\eta = 0.1$.

3 Implementation and Evaluation

You should implement your code in a modular manner and provide an interface so that we can evaluate it with the arguments w, d and data files as in the pseudo code above. You may assume that the data files reside in the same directory as the code. This will allow us to test your code on different data without modification.

You should write your own code for all portions of the assignment. If this helps in your implementation it is fine to use a library for matrix and vector operations but otherwise please do not use any special libraries or code from any source.

Once the code is implemented please evaluate the algorithm with $d = 0, 1, 2, 3, 4$ and $w = 1, 2, 5, 10$, measuring the accuracy on the test set in each case. This means $1 + 4 * 4 = 17$ runs of the algorithm (since when $d = 0$ the width is not relevant). Plot the results so as to show the accuracy as w and d changes. Choose a plot or plots that help illustrate the trends in the results. In your submission include this/these plots and discuss them: how does the performance vary when we increase d and w?

4 Submitting your assignment

- You should submit the following items both electronically and in hardcopy:
  (1) All your code for data processing, learning algorithms, test program, and the experiments. Please write clear code and document it as needed.

  Please make sure that your code runs on *homework.eecs.tufts.edu*. Please include a README file with instructions how to compile and run your code. This should be done so that we can directly run your code (without any modifications) with arguments: w,d,traindata,testdata. If needed please package an interface to your code that enables this.
  (2) A short report with the results and plots as requested and a discussion with your observations from these plots. Please make sure to address the questions posed above in your discussion.

- **Please submit a hardcopy** in class.

- **Please submit electronically using provide** by 4:30 (class time). Put all the files from the previous item into a zip or tar archive (no RAR please). For example call it `myfile.zip`. Then submit using `provide comp135 pp3 myfile.zip`.

Your assignment will be graded based on the **code**, its **clarity, documentation and correctness**, the **presentation** of the results/plots, and their **discussion**.