COSC 420/427/527 Biologically Inspired Computation

Project 4

Due Monday, April 7, 2017

Implement a Back-Propagation Software System.[1] Your system should permit specification of:

1. number of layers
2. number of neurons in each layer (including the input and output layers)
3. learning rate
4. training, validation, and testing files (see below)
5. number of training epochs

Inputs and outputs to the net are floating-point numbers.

Your program takes three files as input (it is permissible to combine them into one file divided into three parts):

Training Set — used for training the net by modifying the weights according to the back-propagation algorithm.

Validation Set — used at the end of each epoch to test generalization (the weights are not modified).

Testing Set — used at the end of training to evaluate the performance of the trained net.

I suggest that you test your BP program on some simple problems, such as *and*, *inclusive or*, and *exclusive or*, to make sure it's operating correctly, before trying the problems on the next page.

COSC 420 students: Do *two* out of the following three problems.
COSC 427/527 students: Do *all* of the following three problems.

For this project you are permitted to work in teams of two. If you do so, each of you should write your own report, name your partner, and say a few words about the parts you did.

[1] Additional information, including data files, can be found on Dr. Van Hornweder's help page, <`http://web.eecs.utk.edu/~mclennan/Classes/420/projects/project4/backprop.html`>.

You will generate two sets of Training/Validating/Testing data from the following functions:[2]

Problem 1:[3]

$$f(x,y) = \frac{1+\sin(\pi x/2)\cos(\pi y/2)}{2}, \quad x \in (-2,2), y \in (-2,2).$$

Problem 2:

$$f(x,y,z) = \frac{3}{13}\left[\frac{x^2}{2} + \frac{y^2}{3} + \frac{z^2}{4}\right], \quad x \in (-2,2), y \in (-2,2), z \in (-2,2).$$

In each case generate (input, output) pairs from random inputs in the ranges specified, and outputs determined by the above formulas. For each problem, generate:

- 200 training patterns
- 100 validation patterns
- 50 testing patterns

For each Problem, do the following experiments:

- Experiment with the number of hidden layers.
- Experiment with the number of neurons in each hidden layer.
- Try to determine the optimum network architecture for each problem.
- Try to determine how sensitive the performance is to the architecture.

In evaluating the architectures, pay attention to performance on the training, validation, and testing datasets.[4] Note that, depending on the architecture, these problems might take several thousands of epochs to converge.

Be sure to include some graphs in your report. For example, you could plot how the error changes as you increase the number of nodes, or plot how the error changes as you increase the learning rate, or plot the error as a function of time (number of epochs), or anything else you can think of. Of course including more graphs/experiments/discussion will lead to higher grades.

[2]You can use the datasets from Van Hornweder's help page or generate your own.

[3]Note that in this formula the arguments to sin and cos are expressed in radian measure; you will have to make appropriate changes if your sin and cos expect arguments in degrees.

[4]For small architectures, the net should achieve average errors less than 4% (which corresponds to 0.001 average square error) within a few hundred, or at most a few thousand, epochs. The error may not seem to change for many epochs and then drop in steps.

Problem 3

For this part of the project, you will use neural networks to recognize handwritten digits (from 0 to 9). This section will show you how the back propagation learning algorithm that you've learned can be used for this classification task.

The data set (saved in txt format) contains 5000 examples of handwritten digits.[5] You can use either of these formats; in the '.txt' format, there are numbers in scientific notation. There are 5000 training examples in `digits.txt`, where each example is a 20 pixel by 20 pixel gray scale image of the digit. Each pixel is represented by a floating point number indicating the gray scale intensity at that location. The 20 by 20 grid of pixels is "unrolled" into a 400-dimensional vector. Each line in the `digits.txt` file represents one image.

The second file `digit_labels.txt` has 5000 lines that contains labels for the sample set. In this file the '0' digit is labeled as '10', while the digits '1' to '9' are labeled as '1' to '9' in their natural order. Although the labels in the file are 1, 2, ..., 10, for the purpose of training a neural network, we need to recode the labels as vectors containing only the values 0 and 1. For example, if $\mathbf{x}^{(p)}$ is an image of the digit 5, then the corresponding $\mathbf{y}^{(p)}$ (that you should use to calculate the error) should be a 10-dimensional vector with $y_5 = 1$, and the other elements equal to 0.

You might find it more convenient to use `merged-digits.txt`. Each line of the file represents one digit along with a label for that digit (nine 0s and one 1). In total, there are 5000 shuffled data in this file. You can split this file into train, validation, and testing sets.

Therefore your neural network must have 400 neurons in the input layer and 10 neurons in the output layer corresponding to the 10 digit classes.

The data files can be found in by clicking on the *Project Data Files* link under *Information* on the course homepage or directly at:

`web.eecs.utk.edu/~mclennan/Classes/420/files/`

You can use 90% of the samples for the training part and 10% for the test. Finally report the accuracy, the architecture that worked best, and a graph for error.

[5] This is a subset of the MNIST handwritten digit dataset (`http://yann.lecun.com/exdb/mnist/`), which was converted to '.mat' format by Andrew Ng and to '.txt' format by Zahra Mahoor.