James Hurt

Project 1

Your report should contain a sections on

1. The technical description of all techniques utilized,

2. The design of the algorithms (pseudo-code, flowcharts, or some

other structured descriptive means),

3. The results of the algorithms,

4. An analysis of the results, i.e., did you obtain what you expected?

Were there any surprises? What conclusions can you draw from the

experiments? etc.

5. Well documented, structured, modular program listings.

## Technical Description

I chose to implement my project in Python3 due to its large number of libraries and the relaxed syntax of the language. The libraries used:

* NumPy is a library specializing in complicated mathematical structures. NumPy was the vector and ndarray handler, which is mathematically equivalent to a matric. I also used it for finding v = wTx as well as storing the weights and biases throughout the program
* Pandas is a library that has a simple and elegant way to parse CSV files into a structure called a Dataframe which resembles an SQL table. I used this library to bring in initial values for w and b as well as the training data with labels from CSV files. Pandas also has a function that converts a dataframe to a NumPy ndarray, making it a very easy library to implement
* PPrint is a Python library that allows for stylized printing such as how many items per line. I used this library to make the results of the network training (weights and bias) more legible.
* ArgParse has functions for passing arguments to the python file via command line. I wanted the user to be able to pass in the dimensions of the network and then input filenames for a general-purpose program. Then I went ahead and built in -a and -b parameters to allow for the script to automatically load the correct files for parts A and B of the assignment.
* MatPlotLib. This library is used for building graphs and plotting points and curves. I used this library to build the graphs asked for in part A, such as error per epoch and the data itself.

## Algorithm Design

The program is a Python implementation of a Multilayer Perceptron using Backpropagation. The design of the algorithm is in the following steps:

1. Setup
   1. Choose initial weights and bias
   2. Load training data
   3. Choose activation function
   4. Choose constants: Alpha, Beta, and Termination Threshold
2. Present point pi to the network and obtain the output
3. Calculate error of pi
4. Backpropagate
   1. Calculate the momentum term using predefined momentum constant and the previous change in wji
   2. Calculate the delta for neuron j
      1. For an output neuron, multiply the derivative of the activation function at the induced local field by error of the neuron
      2. For a hidden neuron, multiply the derivative of the activation function at the induced local field by dot product of the weights from this neuron to all neurons in the next layer with the deltas of those neurons in the next layer
   3. Multiply the delta for neuron j with the predefined constant for learning rate as well the value of the source neuron for weight wji
   4. Add the value calculated in part c to the current value of wji and assign that value as the next wji

The flow of this algorithm through my program is:

1. Init() – parse the arguments passed in from the command line and create the w1, w2, b1, b2, train\_data, and label matrices for which to run
2. Run() – randomize the data, and continue to run epochs until the termination condition is met
3. Epoch() – present each training point to the network and update weights
4. Backpropagate() – update all weights given the output, labels, and current and previous values of the weights

I also have many helper functions to keep logic in one place, such as the fi or fi\_prime function. After the network has converged, I also have functions like print\_results to actually show the results in a legible way as well as functions to present the graphs needed for part A.

## Algorithm Results

Results for part A converged in under 100 epochs almost every time. The variation is number of epochs occurred due to randomization of data. The output for the correct weights w1, connecting the inputs to the first hidden layer were:

|  |  |  |
| --- | --- | --- |
|  | From x1 | From x2 |
| w1 | -0.0513 | -5.3057 |
| w2 | 0.0881 | 4.8483 |
| w3 | 6.3651 | 0.1062 |
| w4 | -0.0443 | -4.7785 |
| w5 | 6.6881 | 0.1143 |
| w6 | 0.1049 | 5.9948 |
| w7 | -0.0507 | -5.2537 |
| w8 | -6.225 | -0.0655 |
| w9 | 6.413 | -0.0651 |
| w10 | 0.0527 | 3.4224 |

With the biases of the 10 neurons in the hidden layer:

|  |  |
| --- | --- |
|  | Bias |
| b1 | 3.2572 |
| b2 | 3.3672 |
| b3 | 3.2299 |
| b4 | 3.1794 |
| b5 | 3.315 |
| b6 | 3.4442 |
| b7 | 3.2845 |
| b8 | 3.2091 |
| b9 | 3.3707 |
| b10 | 3.0282 |

For the output layer, the weights connecting the first hidden layer to the output layer were:

|  |  |
| --- | --- |
|  | Weight |
| w1 | -3.2666 |
| w2 | -2.8687 |
| w3 | -4.7036 |
| w4 | -2.7977 |
| w5 | -4.9355 |
| w6 | -3.8123 |
| w7 | -3.2194 |
| w8 | -4.9047 |
| w9 | -5.0537 |
| W10 | -1.7356 |

And the bias on the single output neuron:

|  |  |
| --- | --- |
|  | Bias |
| b1 | -12.6516 |