Eric Braun, 10121660

13eb20

CISC-452, Assignment 2

October 17, 2016

For my assignment, I created two Python 3 files: main.py and Neural_Network.py.

The main function creates a neural network using the constructor defined in Neural_Network.py

- The neural network takes parameters of:
    - Hidden nodes
    - Number of iterations
    - Learning rate
    - Momentum
- I have hard-coded the size of the inputs and outputs:
    - Inputs coming from the normalized bitmaps: 65
    - The outputs/targets are of size 10
        - I created a vector of length 10 filled with zeros to represent each integer value 0-9
        - Starting at index 0, at whichever position the 1 is located in the vector, is the integer value
            - E.g. [1,0,0,0,0,0,0,0,0,0] = 0, [0,0,1,0,0,0,0,0,0,0] = 2
- In the main, I have hidden nodes set to 25, iterations set to 20, learning rate set to 1, and momentum set to 0.5
- In the constructor of the neural network, it would create random float weights to use of the size of inputs
- The main function then reads in the training.txt and testing.txt into matrices using the function read_in
- Then, the main function trains the network using training.txt data and the train_network function
- Finally, the function tests the network using testing.txt data and the test_network function

The feed forward function takes in the activations (nodes) of the dataset: all layers use sigmoid function

- First, it fills in the input layer using the nodes
- Next, it calculates the output from the input layer using the nodes and the input weights
    - It is the sum of the inputs * input weights
    - We then use the sigmoid function defined in Neural Network to find output
    - This is used as the hidden layer's activation
- Now, we use the hidden layer's output to calculate the output activations
    - This is the sum of the hidden layer's output * the output weights, then use the sigmoid funcion

- The output layer's output is then returned

The back propagation function takes in the targets of the dataset: all layers use the derivative of the sigmoid function

- We first determine the direction the input weight vector has to be changed in (gradient) by calculating the error (- (d – y)) then using the derivative of the sigmoid function with the first layer output
- Now we update the weight by finding the difference
    - The difference is the direction * the activation of the layer below
    - Then, subtract the learning rate * the difference + the output * momentum
- It will calculate the final error using root mean square error (RMSE) of all the nodes and outputs this error

Using the training function we iterate through the data a certain amount of given times, and keep changing the weights by feeding forward then back propagating the error and updating the weights on the way back. After each iteration the function will display the error.

Now, we call the test network function using the testing.txt data and the new weights. I made a running sum and incremented each time an integer was correctly categorized. I converted the 10 element vector back into its integer value using the transform prediction function.

I found that I could usually get the code to correctly organize about 90 – 95% of the data depending on the amount of iterations, hidden nodes, momentum, and learning rate.