COMP30230 Programming Assignment Report

Code Structure

Introduction of Code Structure

Self.number_of_inputs = number of input neurons

Self.number_of_hidden_units = number of hidden neurons

Self.number_of_outputs = number of output neurons

self.input_neurons = list of input neurons

self.outputs = list of output neurons

self.hidden_neurons = list of hidden neurons

self.weights_lower = list of lower level weights

self.weights_upper = list of upper level weights

self.weight_changes_lower = list of lower level weight changes

self.weight_changes_upper = list of upper level weight changes

self.activations_lower = list of lower level activations

self.activations_upper = list of upper level activations

Class MultiLayeredPerceptron Functions:

Randomize(): Initialiases weights to random values between 0 and 1

Forward(Input, use_sigmoid): Forward propagation

Backward(target, use_sigmoid): Backwards Propagation

Update_weights(learning_rate): updates the weights with changes * learning rate

Sigmoid(sig, is_backwards): Sigmoidal activation function

Hyperbolic_tangent(tanh, is_backwards): Hyperbolic tangent activation function

GetError(target): Generates error value

ToString(epoch_num): tostring method for debugging purposes

Experiment Introduction

I found this assignment tough, between figuring out what language to program it in (initially I chose C by quickly abandoned the idea) to figuring out what forward and backpropagation does to implementing it but overall, I enjoyed this assignment. In course of the assignment,

I ran 18 experiments in total. 9 for XOR and 9 for SIN. Each experiment changed the max_epochs and learning_rate. This allowed for a good range of results across different values. Below you can see a key for the experiments including: the experiment number, value of max_epochs, and value of learning_rate.

Experiments Key

XOR

1. Max_epochs = 10000, learning_rate = 0.1
2. Max_epochs = 10000, learning_rate = 0.5
3. Max_epochs = 10000, learning_rate = 1
4. Max_epochs = 1000, learning_rate = 0.1
5. Max_epochs = 5000, learning_rate = 0.1
6. Max_epochs = 1000, learning_rate = 0.5
7. Max_epochs = 5000, learning_rate = 0.5
8. Max_epochs = 5000, learning_rate = 1
9. Max_epochs = 1000, learning_rate = 1

SIN

1. Max_epochs = 10000, learning_rate = 0.1
2. Max_epochs = 10000, learning_rate = 0.5
3. Max_epochs = 10000, learning_rate = 1
4. Max_epochs = 5000, learning_rate = 0.1
5. Max_epochs = 5000, learning_rate = 0.5
6. Max_epochs = 5000, learning_rate = 1
7. Max_epochs = 1000, learning_rate = 0.1
8. Max_epochs = 1000, learning_rate = 0.5
9. Max_epochs = 1000, learning_rate = 1

Experiment Results

Question 1 and 2 (XOR)

The inputs to XOR was ([0,0], [0,1], [1,0], [1,1])

The desired outputs were ([0], [1], [1], [0])

The first experiment I ran was with 10000 max_epochs and 0.1 learning_rate. The error rate started at 0.161 and reduced to 0.0034 over the course of the epochs. The generated outputs were:

[0] - 0.09859809873747738

[1] - 0.9240878715740612

[1] - 0.916148409216802

[0] - 0.06915403519910668

What I found in my experimentation is that as you increased the learning rate it produced a smaller error and a more accurate answer with a learning rate of 0.5:

[0] - 0.028025053800497565

[1] - 0.975826748603099

[1] - 0.9761563172387082

[0] - 0.021068375859543565

As we can see here all the generated outputs have increased in accuracy with the error rate now ranging from 0.16 to 0.0002. With a learning rate of 1 the produced output was:

[0] - 0.02220838446220258

[1] - 0.9821003329446003

[1] - 0.9798040098569157

[0] - 0.01982874801437289

As we can the jump from 0.1 – 0.5 produced a significant improvement in accuracy but the jump from 0.5 – 1 it starts to plateau.

The next set of experiments I ran increased ranged over different values for max_epochs. Starting at 1000 with a learning rate of 0.1 the results are as follows:

[0] - 0.488345307513323

[1] - 0.518401462971133

[1] - 0.49656481061262053

[0] - 0.5139531089193424

As we can see the results are almost all halfway between 0 and 1. So we know that 1000 epochs is not enough.

The next value I ran with was max_epochs = 5000 with a learning rate of 0.1. The results are as follows:

[0] - 0.2186349231762324

[1] - 0.7206247368485074

[1] - 0.7171716450496605

[0] - 0.33639431082789367

As we can see the results here are starting to edge towards theyre proper values as seen in the original outcome.

The rest of the tests I ran on XOR consisted of max_epochs of 1000, and 5000 with learning rates of values (0.5, 1). These showed more of the same results with a higher learning rate producing a more accurate score.

Question 3 and 4 (SIN)

The next batch of experiments I ran was with SIN.

For SIN we had to generate 500 random examples and corresponding outputs (which is done in the code). The first 400 examples were used for training while the remaining 100 were reserved for testing.

The first set of tests (Tests 1 through 3 inclusive) I ran with max epochs of 10000 and learning value of (0.1,0.5,1) what I gathered from these tests is the higher the learning rate the larger the testing error. I think this is due to overfitting of the training data.

Test 1: Learning rate = 0.1

Testing Error: 1.0836171678215702, Training Error: 0.01132136518644199

Test 2: Learning rate = 0.5

Testing Error: 6.504699684465735, Training Error: 0.11684034912202879

Test 3: Learning rate = 1

Testing Error: 6.100974431944437, Training Error: 0.12299167369945047

As we can see above the testing error drastically jumps up when we increase the learning rate for 0.1 to 0.5 but plateaus when we increase it to 1. This patter repeats itself when we set max_epochs to 5000 and 1000 respectively as see in the results below:

Learning rate = 0.1

Testing Error: 3.331087235809632, Training Error: 0.0335948031169573 (Max epoch = 5000)

Testing Error: 2.5726930894233204, Training Error: 0.034409252372682636 (Max epoch = 1000)

Learning rate = 0.5

Testing Error: 9.166953102155459, Training Error: 0.08179048769254738 (Max epoch = 5000)

Testing Error: 8.534559228844575, Training Error: 0.08894925310814092 (Max epoch = 1000)

Learning rate = 1

Testing Error: 6.155501354535468, Training Error: 0.17303803686759256 (Max epoch = 5000)

Testing Error: 11.882696589375628, Training Error: 0.15502033597410184 (Max epoch = 1000)

Conclusion

In conclusion I have both learned a lot throughout this assignment and this course. I now know how to properly program and implement a Neural Network / MLP. This will be of great use as I know how to customize an MLP to my own specific needs. Initially when I started this assignment I used C but in hindsight I'm very happy I changed to python as while C is a fast language it can be quite painful at times to program in it if you don't have the prerequisite experience. When I moved to python, I set out using NumPy arrays but quickly revered back to 2-Dimensional Python lists.

In this assignment I have learned how to implement both forwards and backwards propagation how to manually split data into training and testing. I have also learned how to calculate the error and use it has a meaningful evaluator of how good a certain instance of an MLP is.