Jeremy Coupland

Machine Learning: Back Propagation Assignment Report

October 2015

Method Pseudo-code:

This is the pseudo-code for the most significant methods used in this assignment.

Initializing Weights: The weights are stored using a vector of vectors.

To initialize the hidden layer weights:

```
For each node i in the hidden layer
    For each j node in the input layer
        Assign a random weight between -0,5 and 0,5 to hidden weight vector [i][j]
```

To initialize the output layer weights:

```
For each node i in the output layer
    For each j node in the hidden layer
        Assign a random weight between -0,5 and 0,5 to output weight vector [i][j]
```

Feed Forward: Takes a vector of inputs as a parameter.

```
Set a local vector equal to the vector passed as a parameter.
```

To get the output at the hidden layer:

```
For each node i in the hidden layer
    For each node j in the input layer
        Calculate the sum of all the inputs[j] * hiddenLayerWeights[i][j]
    Set the output of the hiddenLayer[i] equal to the sigmoid of the sum
```

To get the output at the output layer:

```
For each node i in the output layer
    For each node j in the hidden layer
        Calculate the sum of all the hiddenLayer[j] * outputLayerWeights[i][j]
    Set the output of the outputLayer[i] equal to the sigmoid of the sum
```

Propagate Error Backward: Takes a vector of desired outputs as a parameter.

To calculate the output layer error:

For each node i in the output layer

Set the outputError vector[i] to: outputLayer[i] * (1 - outputLayer[i]) * (desiredOutput[i] - outputLayer[i])

To calculate the hidden layer error:

For each node i in the hidden layer

Calculate the error using hiddenLayer[i] * (1 – hiddenLayer[i])

For each node j in the output layer

Sum all the outputLayerWeights[i][j] * outputError[j]

Multiply the error by the sum

Set hiddenError[i] to the error

To calculate the output layer weights:

For each node i in the output layer

For each node j in the hidden layer

Calculate the change in weight: learningRate * outputError[i] * hiddenLayer[j]

Add the change to outputLayerWeights[i][j]

To calculate the hidden layer weights:

For each node i in the hidden layer

For each node j in the input layer

Calculate the change in weight: learningRate * hiddenError[i] * inputLayer[j]

Add the change to hiddenLayerWeights[i][j]

Mean Squared Error: Takes a vector of desired outputs as a parameter.

To calculate the mean squared error:

For each node i in the output layer

Calculate the error: desiredOutput[i] – outputLayer[i]

Sum the square of the error for each node i

Return the average of the sum of the errors

Train: Takes two 2D vectors, inputs and outputs, and an integer trainingSetSize as parameters.

While(true)

    For each training example i in training set size

        FeedForward(inputs[i])

        PropagateErrorBackward(outputs[i])

        Sum the mean squared error of MeanSquaredError[i]

    Average the mean squared error for each training example

    If the average is less than the cutoff, break.

Classify: Takes a vector of inputs as a parameter.

For each node i in the output layer

    Track the index of the maximum value

Return the index

Testing Results:

The following results are averaged over the course of multiple iterations. Three different testing environments were used, as well as four different test parameters for each environment. Individual results can be found in the attached spreadsheet.

Environment 1:

	Most Mines Gathered	**Average Mines Gathered**	**Deaths**
Test 1	17.8	5.3	2
Test 2	11.6	2.8	1.7
Test 3	8.9	1.9	1.4
Test 4	33.1	10.7	4.4

Environment 2:

	Most Mines Gathered	Average Mines Gathered	Deaths
Test 1	9.6	2.6	2.7
Test 2	5.6	1.1	2.4
Test 3	5.3	1.1	5.6
Test 4	21.9	6.5	8.7

Environment 3:

	Most Mines Gathered	Average Mines Gathered	Deaths
Test 1	4.7	0.9	4.9
Test 2	3.9	0.6	5.7
Test 3	2.9	0.5	10.6
Test 4	9.4	2.2	10.6

Conclusion:

I found that, through testing, different parameters could be changed to maximize certain criteria. We found that reducing the number of hidden nodes decreases the number of deaths, while also decreasing the number of mines gathered. Increasing the number of hidden nodes does the opposite. So, depending on what the user values more, lives or mines gathered, an appropriate value can be used.