Module Code: COC102

Module Title: Advanced AI Systems

Title: ANN Implementation

Student Name: Max Pearson

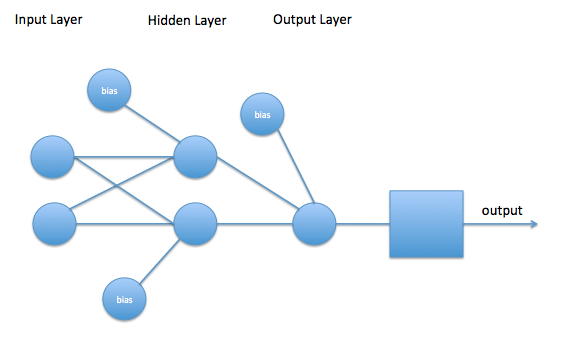
Student ID: B123103

1. Introduction

The task was to construct a program called a Neural Network Multi-Layer Perceptron using a back propagation algorithm. This algorithm calculates errors and finds classifications in data by using mathematical functions such as: the hyperbolic tangent function, the logistic function and chain rule to derive changes from neuron to neuron.

The program works by setting 3 sets of Neurons called input, hidden and output. These sets or ‘layers’ can communicate with each other through weights, which are initially random. *Figure1* is an example of a Multi Layer Perceptron with 2 inputs nodes, 2 hidden nodes and 1 output node, including bias for input and hidden.

In order to create this Multilayer Perceptron the programming language *python* was used with packages including *random*, *math, sys, pylab*, *numpy*, *matplotlib*, *scipy*, and *cPickle*. These packages allowed the construction of graphs and were installed by following [2], also allowing the Network to save and load data. Source [1] was used to construct the function ***xlsx*** as a means of populating input from the file CWKDATA.xlsx and [3] was used to formulate the network.

Figure 1.

1. Implementation
   1. Program Details

Classes

* **NN**: Main Neural Network Class.
* **Neuron**: Contains index, activation, delta, sum and bias for each neuron.
* **Layer:** Holds multiple Neuron objects for each Input, Output and Hidden layers.
* **Data** Processing data from CWK.xlsx and formatting for use with NN.
* **Layers:** Holds multiple Layer objects, which are the structure of the Network, allowing NN full access to all Neurons with a simple function.

Variables and Objects

* NN
  + ***netName****:* Used to save network with a given name.
  + ***filename****:* Unused variable
  + ***states****:* Allows for saving of network weights, changes and parameters via a text file(accessed via epoch)
  + ***learning\_rate****:* Parameter used to change weight values (variable) subject to change by bold driver.
  + ***Momentum****:* Parameter used to change weight values (constant).
  + ***Network****:* Contains entire structure of network including layer, weights, weight changes.
  + ***trainingErrors****:* Array for storing training error network output when storing files.
  + ***validationErrors****:* Array for storing validation error network output when storing files.
  + ***learningRates****:* Array for storing changes in learning rates across epochs.
  + ***testDataPredictions****:* Array for storing predictions achieved from train function.
  + ***testDataOutputValues****:* Array for storing output to evaluate against predictions.
  + ***testErrors****:* Array for storing test errors, not currently used!.
  + ***exitCounter****:* Counter for declaration of when a network should terminate, how many increases in error has to occur to terminate.
* Data
  + ***Document****:* Store data from CWKDATA.xlsx in object for data normalisation.
  + ***tagLine****:* Used to identify the column in data retrieved.
  + ***maxN****:* Maximum value of all data contained.
  + ***minN****:* Minimum value of all data contained.
* Layers
  + ***Layers****:* Array storage for multiple layer objects.
  + ***NumNeurons****:* Number of Neurons contained within the entire Network.
  + ***NumLayers****:* Number of Layers stored within the entire Network.
* Layer
  + ***name****:* Name of layer, normally Input, Hidden or Output.
  + ***Neurons****:* Array containing multiple Neuron objects.
* Neuron
  + ***activation****:* Activation value of Neuron assigned via activate function.
  + n: Neuron number identifier.
  + ***delta****:* Delta value of Neuron assigned via ***setDelta*** function.
  + *S:* Sum of Activations of Layer-1 + Neurons bias value.
  + ***bias****:* Bias value of Neuron.

Class Methods

* NN
  + ***feed\_forward****:* Implements a feed forward algorithm as formulated in Figure2.
  + ***backPropagation****:* Implements a back propagation algorithm as formulated in Figure 2.
  + ***decayWeight****:* Implements a weight decay algorithm as formulated in Figure 2.
  + ***train****:* Given a set of training input patterns, pass input patterns through feed forward algorithm and back propagation with n iterations.
  + ***test****:* Given a set of test input patterns, pass input patterns through feed forward algorithm.
  + ***getError****:* Find sum of activations in output layer.
  + ***bold\_driver:*** Based on error changes increment or decrement learning rate parameter.
  + ***captureState:***Capture state of Network to state array and delete every first state every 12 states created.
  + ***getStateTypeAt****:* Return array with name given at epoch state.
  + ***saveNetwork:***Save state via epoch to file NetworkN.txt
  + ***getWeights:***Given 2 index values find associated weight.
  + ***getW:***Return weight array.
  + ***getChange:***Given 2 index values find associated weight changes.
  + ***getC:***Return weight changes array.
  + ***getLayer:***Given index of layer return layer object
  + ***getLayerNeurons:***Given index of layer return Neurons array within layer.
  + ***getLayerSize:***Given index of layer return number of Neurons contained in layer.
  + ***getPrediction:*** Return output activation index 0.
  + ***runProgram:*** Give network Training Set and Validation Set and start program.
  + ***initialiseWeights:*** Initialise Random weights based on number of Neurons.
  + ***checkCount:*** Check if error has increased 10 times if so terminate training and validation.
  + ***plotErrors:*** Plot Validation against Training Errors obtained by the Network.
  + ***plotLearning***: Plot Learning rates.
  + ***plotPrediction:*** Plot Actual Output against predictions obtained by the Network.
* Data
  + ***getMax:*** Get maximum value and assign *maxN.*
  + ***getMin:*** Get minimum value and assign *minN.*
  + ***populate***: Populate document object with values obtained from CWKData.xlsx.
  + ***xlsx***: Read in data from and xlsx file (obtained from [1]).
  + ***getBy***: Using ***tagLine*** to find values via column index.
* Layers
  + ***addLayer:***Add layer to layers array.
  + ***updateLayerSize****: Update weights size.*
  + ***\_\_updateWeightSpace:*** Create new 2 dimensional array based on size.
* Layer
  + ***\_\_createLayer:***Create N amount of Neurons for layer.
* Neuron
  + ***setDelta:*** Set delta value of Neuron.
  + ***activate:*** Activate Neuron via logistic function and assign as variable activation.
  + ***setSum:*** Set S value of Neuron.

Functions contained in mlp.py

* ***plotNN:***Plot all arrays Associated with the Network given.
* ***saveNN:***Save Neural Network Object in .bin file.
* ***loadNN****:* Load Neural Network Objects via .bin file.
* ***loadNetwork:***Load Network state via .txt file.
* ***showNetwork:***Show Network weights via network name.
* ***getNetworkArrays:*** Core function which specifies data sizes, neural network configurations and returns Validation, Testing, Training all normalised for each network to use.
* ***runNetwork:*** *Run Program if file is main.*

Functions contained in Utility.py

* ***vector:***Create array with indexes 0-length, used with plotting x-axis coordinates.
* ***plotNetworks:***Plot all 6 Network Learning Rates in one graph.
* ***viaTan:***Normalise Data for tanh activation function.
* ***viaSigma.***Normalise Data for logistic activation function.
* ***normalise:***Normalise Data with chosen function.
* ***setWeights:*** Set 2 Dimensional Array with given input and return.
* ***hyperbolic\_tangent:*** Given a number return tanh of the number.
* ***hyperbolic\_tangent\_dv:*** Given a number return inverse function of tanh.
* ***generateRandFor:*** Generate random number between two values.

***sigmoid:*** Given a number return*(logistic function)*

* ***sigmoid\_dv:*** Given a number return the inverse logistic function.
* ***activation\_function:*** Given a number and chosen activation (sigmoid or tanh) return value.
* ***derivative\_function:*** Given a number and chosen activation (‘sigmoid or ‘tanh) return value.
* ***createNormalisedDataSet:*** Using data stream taken from CWKDATA.xlsx populate an array in accordance with Figure 2, with normalised values.
  1. Neural Network Functions

Formulations are based upon sources including [3] and Advanced AI Lecture Slides (Backpropagation and ANN)

*= activation of layers*

*= delta at layer*

Feed Forward

Back Propagation

Update Weights

Weight Decay

Bold Driver

1. Network Configuration

The Neural network created in this report has the following configurations:

3 Layers made up of Input, Hidden and Output, 3 parameter rates including learning rate, momentum and weight decay which are set to a default of 0.5, 0.9 and 0.03.The learning rate is variable as the function bold driver manipulates the value based on error changes but will not exceed 0.9 .The Data that is accepted by the network has to be in the form shown in Figure 2 and has to be in the ranges of (0-1) or (-1-1), or the network will not be able to function when weights get too large. (logistic function will throw stack overflow error)

The functions contained within the Network are feed forward, back propagation, weight decay and bold driver, unfortunately during the implementation of the algorithm the function annealing was not properly finished and as such may have affected its overall performance. Also it was deemed that the data should be normalised between (0-1) with the use of the sigmoid function for activations, instead of tanh for activations, although the functionality for both still remains in the program.

The mathematical formulation of each functions use in the Neural Network are detailed above and are fully integrated and fully functional within the overall system. However in order to show the data going in and out of the Neural Network i.e. processing and outputting, it had to be able to use external functions to plot graphs and format data streams and as such functions and classes such as Data, Layers, Layer and Neurons were used to simplify the program in an object sense.

During testing phases of numerous networks, stabilising the weights to stop overflows was tricky, in order to minimise risk from the Network getting trapped in global maxima, the weights were set to a random distribution between in order for the network to initially start correctly, as it was found that setting input to hidden weights beyond 0.5 and -0.5 and hidden to outputs weight beyond 2.0 and -2.0 was somewhat ineffective.

Figure 2.

* 1. Network Details

*LR*: Learning rate parameter

*Mom*: Momentum parameter

*WDF*: Weight Decay rate parameter

*Hidden*: Hidden nodes

*Inputs*: Input nodes

*Training Iterations*: Run training set N times before validation set.

Figure 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Network** | **LR** | **Mom** | **WDF** | **Hidden** | **Inputs** | **Training Iterations** |
| **1** | 0.5 | 0.9 | 0.03 | 8 | 8 | 10 |
| **2** | 0.5 | 0.8 | 0.025 | 7 | 8 | 20 |
| **3** | 0.4 | 0.9 | 0.025 | 6 | 8 | 10 |
| **4** | 0.4 | 0.8 | 0.020 | 5 | 8 | 20 |
| **5** | 0.3 | 0.9 | 0.020 | 4 | 8 | 10 |
| **6** | 0.3 | 0.8 | 0.015 | 3 | 8 | 20 |

*Performance Rating*: Performance rating based on average time to execute.

*Execution*: Epochs network terminated at.

*Training Set*: % of Data Set for Training Network

*Validation Set:* % of Data Set for Validation of Network

*Test Set*: % of Data Set for Testing Network

Figure 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Network** | **Performance Rating (speed)** | **Execution (epochs)** | **Training Set** | **Validation Set** | **Test Set** |
| **1** | 5/10 | 36 | 70% | 15% | 15% |
| **2** | 3/10 | 48 | 70% | 20% | 10% |
| **3** | 6/10 | 32 | 70% | 15% | 15% |
| **4** | 8/10 | 18 | 70% | 20% | 10% |
| **5** | 4/10 | 38 | 70% | 15% | 15% |
| **6** | 7/10 | 20 | 70% | 20% | 10% |

1. Network Testing

All Network tests were generated through *python*’s scientific packages: *matplotlib, scipy, numpy and pylab.* Which are core libraries of python but have to be installed before running the program. All generated images Figure 5-8 will be stored under ‘Network Test’ file in the programs main directory.

* 1. Network Predictions

Figure 5

****

****





* 1. Network Errors

Figure 6







* 1. Network learning rates

Figure 7







Figure 8.

1. Evaluation

In order to accurately measure the differences between the networks the data has been graphed to illustrate small changes in error, learning rate and prediction values. Overall the 6 networks performed very well obtaining a minimum point in less than 50 epochs. In order to first establish which networks were worth testing, networks that were weak and consistently not terminating had to be eliminated. The Eliminated networks contained 5 training cycles and terminated beyond the limit of 500 epochs, with a learning rate and error that could not be graphed accurately against the other networks, so a training cycle of 10-20 iterations was selected as a base of iterations.

If we choose to look at Network predictions we can see that each network preforms very differently, Network 1 is fairly consistent in fitting small deviations in output very quickly but is unable to accurately increase its gradient enough to fit its first spike, fortunately by epoch 50 it has taken enough training cycles and injected enough noise to accurately follow the spike in output. A good indication that this network is learning for longer is that its learning rate stays at a constant for around 20-25 epochs, meaning its error is at a constant rate of decrease until it reaches its minimum and starts to converge. This Network is the base configuration in which all 6 networks should be compared to, as it contains the Neural networks default values.

Network 2 predictions start of very erratic, under fitting and over fitting is clearly a very big problem here, this may be that the network base configurations cannot account for double the training iterations of Network 1 with 75% of the training examples to work with, as such may need a lower weight decay factor and a lower learning parameter, the drop in hidden nodes may be stopping this network from accurately getting a difference in changes earlier on. If we look at the learning rate changes in Network 2 we can see that it attempted to converge very quickly but was unable to reach termination as it found an increase in error very close to termination, whether this meant the termination counter should be less than 10 or not this network has not performed very well and as such received a poor performance rating.

Network 3 is very similar to Network 1 in fitting output but outperforms it early on as the training and validation errors start to drop earlier at around epoch 10 instead of 12-13, also as we follow the predictions further on, around epoch 20-50 the network is able to have higher peaks in smaller deviations of output compared to Network 1, however when the network tries to fit a large spike it only accounts for around half the gradients increase, this network configuration is clearly the benchmark for finding small deviations, the initialisation learning rate and momentum are clearly perfect for this network, considering the training set on this network is only 10 this network performed very well. Also this network only stayed at a constant decrease in error for around 10-15 epochs, the only difference being is that this network could perform much better overall if it had more training cycles.

Network 4 was the fastest network to converge and was also able to accurately follow the spike in output. However it was unable to follow the output trend early on, mainly because it has a low number of hidden neurons and cannot account for differences in small spaces of time. In order to achieve some accuracy the network should have more values to compare and derive in order to make smaller gradient changes or large gradient changes quickly and effectively. This network may have performed the best in minimising errors but it didn’t accurately graph a prediction, if this network was able to stay at a constant learning rate for longer with say 2 more hidden nodes it may outperform Networks 1 and 3.

Network 5 is almost identical to Network 3 for predictions but fails to find minima as quickly. Network 5 starts to converge around 40 epochs as opposed to Network 3 which starts to converge around 30 epochs, Network 5 is able to follow output closer but terminates far beyond and as such is deemed useless.

By this point it is clear to see that the validation set should not be 20% of the data set, as it clearly shows in Networks 2, 4 and 6 that a constant decrease in error never occurs and as such the large or small deviations that may occur randomly in the data set are never fully understood by the network. In order to achieve a steady predictor and an overall efficient performance in the Network it has to be able to have more scope in training, by mixing the configurations of Networks 3 and 4 we would achieve the best performance in both finding the minima and generating a good prediction.

* 1. Conclusion

By comparing all 6 networks it is evident that Networks 3 and 4 performed the best with regards to speed and accuracy, by combining attributes of both Network’s, stronger predictors and faster convergence to minima will be created. Network 3 displayed a higher accuracy in predictions with a larger testing set but less training iterations. Network 4 decreased in error quicker because of its low hidden node value small testing set and higher training iterations.

By implementing a network configured with a 15% testing set, 15 training iterations, between 5 and 6 hidden neurons, an initial learning rate of 0.5, a momentum of 0.9 and a weight decay factor 0.025, also decreasing termination value to 8 instead of 10. The Network should achieve a termination between 20 and 30 epochs with a good accuracy in predictions.

To conclude this report, in order to represent a Neural Network that best performs when given the data set provided, Network 3 was chosen as it both shows speed of converging and good prediction values. However to improve this Network the learning rate should be increased to 0.5 but all other values should stay the same.

Proposed Network Structure

Training Data Set = 70%

Validation Data Set = 15%

Testing =100%

Training iterations = 15

Data Set = 560 patterns

Termination Value = 8 increases in error

Weight as (Figure 9 and 10)

Figure 9

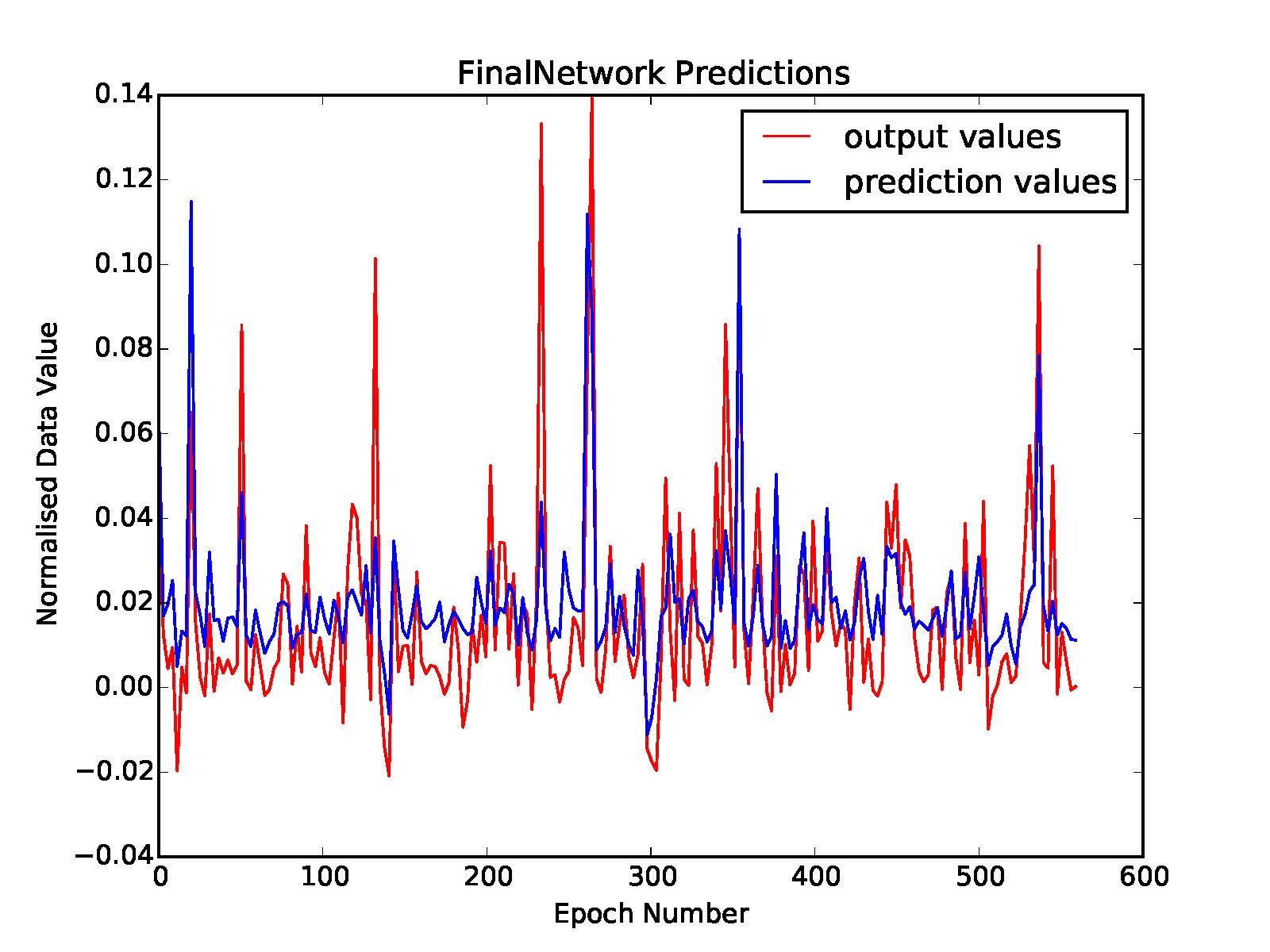
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Layer (j) | | | | | |
| Layer (i) | 8 | 9 | 10 | 11 | 12 | 13 |
| 0 | -4.41480199671 | 1.25257784096 | 0.274692904998 | 0.943857476099 | 1.01060375289 | 0.139205291935 |
| 1 | 1.10072113624 | 0.951462013908 | 0.744367853446 | 1.09228962643 | 0.903865530787 | 0.992662460068 |
| 2 | 0.7642198062 | 0.789102909712 | 1.06684285828 | 1.12862510861 | 0.831950743054 | 0.975055388585 |
| 3 | 1.06109043196 | 0.7884227912 | 0.987977474516 | 0.986423404717 | 0.65454568421 | 1.10870241307 |
| 4 | 0.635658299047 | 1.15676519221 | 0.915855775716 | 0.831044381837 | 0.856240578802 | 0.832036702362 |
| 5 | 0.883320025994 | 0.860275053496 | 0.756832505855 | 0.718898975933 | 1.00227604606 | 0.855762564705 |
| 6 | 0.812257227145 | 0.944644627254 | 0.93105801632 | 0.788605593635 | 0.71004962012 | 0.795789492468 |
| 7 | -2.32400104114 | 0.921194104769 | 0.415479464925 | 0.794865081235 | 0.826300736104 | 0.51353329694 |

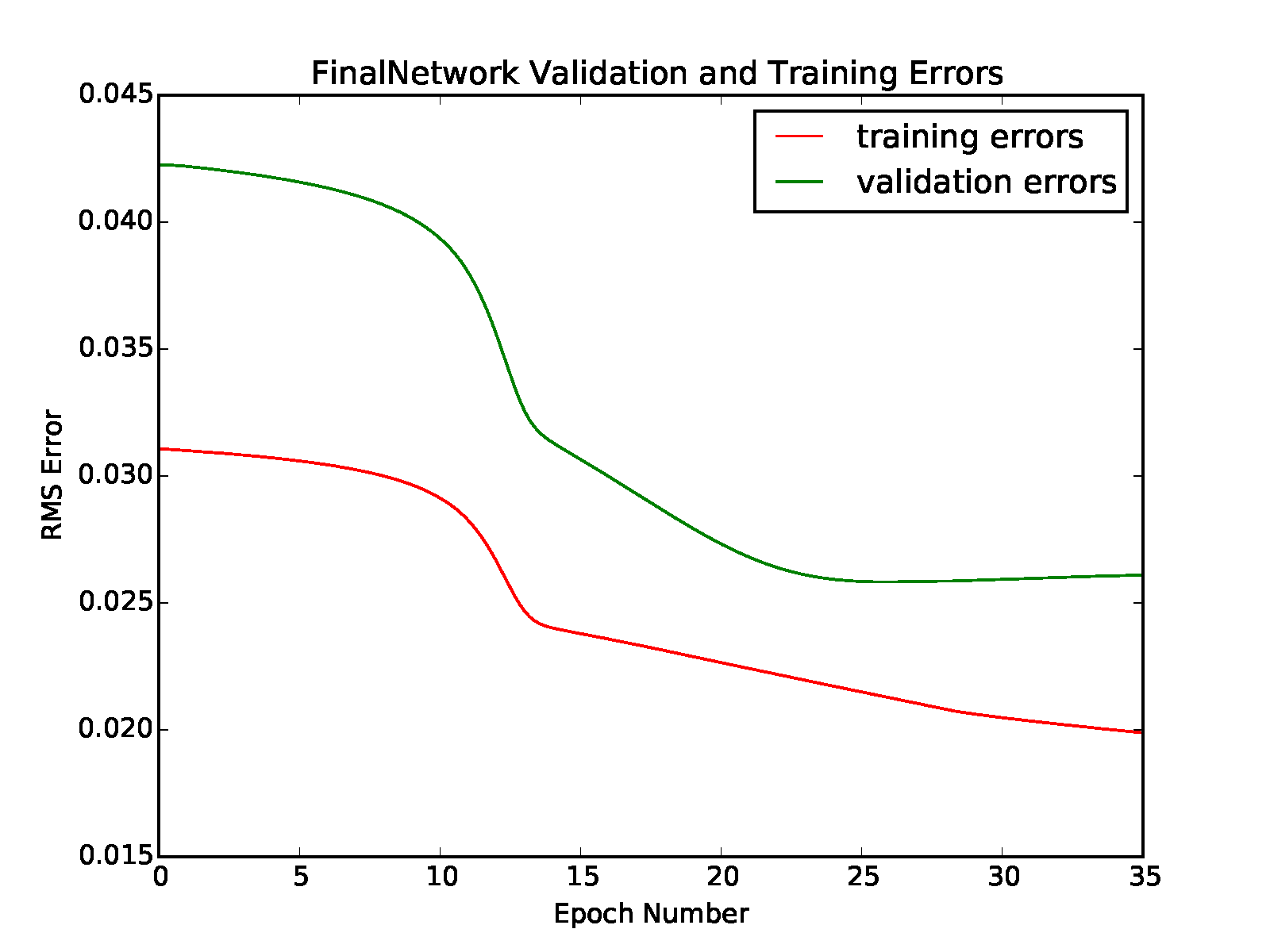
Figure 10

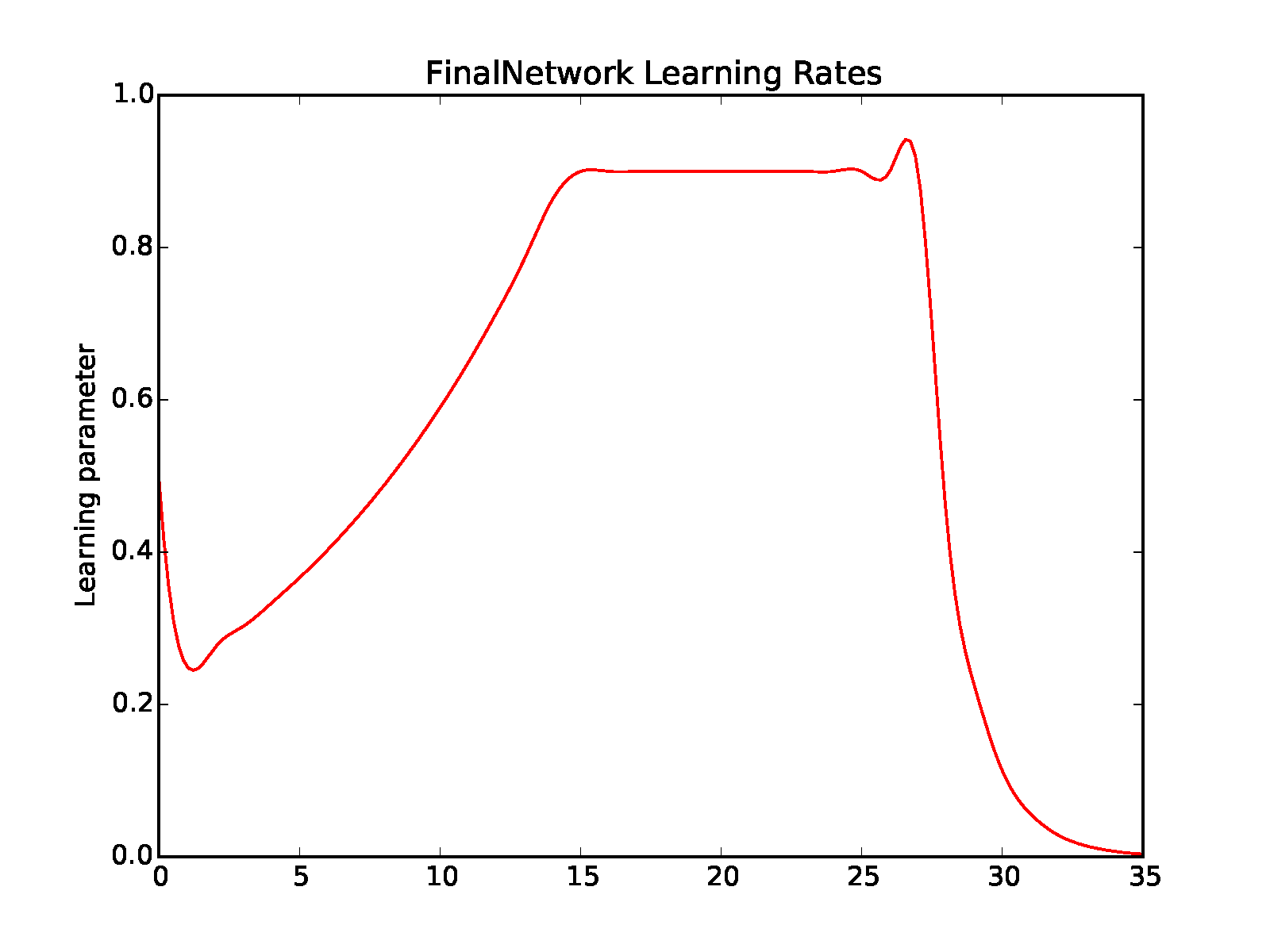
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Layer (j) | | | | | |
| Layer(k) | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | -4.52979398322 | 0.446423284147 | -1.2227476045 | -0.0748956050026 | -0.0856776458392 | -2.72166514612 |

Testing Network on Entire Data Set

Terminated after 24 epochs ( 3 test were executed and 24 was the minimum, 32 was the maximum)







References

[1] R. [closed], 'Reading xlsx files using Python', *Stackoverflow.com*, 2015. [Online]. Available: http://stackoverflow.com/questions/4371163/reading-xlsx-files-using-python. [Accessed: 15- Mar- 2015].

[2] Pen and Pants, 'Install Python, NumPy, SciPy, and matplotlib on Mac OS X', 2012. [Online]. Available: http://penandpants.com/2012/02/24/install-python/. [Accessed: 10- Mar- 2015].

[3] The Clever Machine, 'Derivation: Error Backpropagation & Gradient Descent for Neural Networks', 2014. [Online]. Available: https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/. [Accessed: 15- Mar- 2015].