Acknowledgement

The author wishes to thank Prof. Tom Gedeon for his support and encouragement throughout the duration of this project and my wife, Victoria Oakden MBA, for her patience and support.
Abstract

Cascade Correlation (CASCOR) and Extreme Learning Machines (ELM) provide viable alternatives to traditional single-hidden layer feed-forward neural network (SLFN) trained using back-propagation (BP). In this thesis the theories of ELM and CASCOR are described, implementations tested and experiments conducted to compare their performance to traditional SLFN trained with BP. The experiments demonstrate that ELM is a significantly faster algorithm than BP and somewhat faster than CASCOR. The contribution of this project is the development of a new type of Neural Network, CASCELM. In this network CASCOR is combined with ELM to create a hybrid model. Experimental results show that these networks are slower to train than ELM but offer an advantage in that as a constructive algorithm they automatically arrive at an optimum topology during training. Two new ideas are proposed for improving the Layered ELM machine, which represent an opportunity be further investigation.
# Table of Contents

## Contents

1 Introduction.......................................................................................................................... 7  
   1.1 Motivation......................................................................................................................... 7  
   1.2 Objective of Project......................................................................................................... 8  
   1.3 Contribution..................................................................................................................... 8  
   1.4 Report Organisation......................................................................................................... 8  
   1.5 Hardware Specifications................................................................................................. 8  

2 Extreme Learning Machines............................................................................................... 9  
   2.1 Literature Review............................................................................................................. 9  
   2.2 ELM Theory.................................................................................................................... 9  
   2.3 ELM Program................................................................................................................. 12  
   2.4 Testing of Program......................................................................................................... 13  
   2.5 Variation in output due to hidden layer randomisation................................................... 15  
   2.6 Hidden Layer Weight Investigation............................................................................... 16  
   2.7 The application of ELM in Genetic Algorithms............................................................ 21  
   2.8 ELM Conclusions........................................................................................................... 22  

3 Cascade Correlation............................................................................................................ 23  
   3.1 Introduction..................................................................................................................... 23  
   3.2 Literature Review............................................................................................................ 23  
   3.3 Performance Evaluation................................................................................................. 24  
   3.4 Conclusion....................................................................................................................... 25  

4 CascELM............................................................................................................................. 26  
   4.1 Introduction..................................................................................................................... 26  
   4.2 Shallow Cascade ELM..................................................................................................... 26  
      4.2.1 Shallow cascade ELM results..................................................................................... 27  
      4.2.2 Single Cascaded ELM............................................................................................... 29  
      4.2.3 Full Cascaded ELM................................................................................................. 30  
      4.2.4 Comparison of ELM and CASCELM...................................................................... 32  
   4.3 Conclusion....................................................................................................................... 33
List of Figures

Figure 1: Example of a simple ELM ................................................................. 10
Figure 2: Comparison of training test accuracy and speed for ELM and SLFN ........ 13
Figure 3: Comparison of training results for double spiral data set ..................... 14
Figure 4: Comparison of training accuracy as hidden layer size increases ............. 14
Figure 5: Selection of ELM with 25 neurons in hidden layer ............................ 15
Figure 6: Selection of ELM with 50 neurons in hidden layer ............................. 15
Figure 7: Demonstration of over fitting with ELM trained using diabetes data set .... 16
Figure 8: ELM training capacity after normalising/scaling hidden layer
Figure 9: Change in accuracy (double spiral data set) as weight range varies
Figure 10: Change in accuracy (diabetes data set) as weight range varies
Figure 11: CASCOR output when trained with double spiral for 1, 6, 12 cascades
Figure 12: CASCOR accuracy against number neurons
Figure 13: CASCOR training time as hidden layer size increases
Figure 14: Shallow ELM Cascade
Figure 15: Comparison of ELM and shallow cascade ELM training accuracy
Figure 16: Quality comparison, ELM and shallow cascade, 20 nodes in hidden layer
Figure 17: schematic of type 1 CASCELM
Figure 18: CASCELM type 1 results for double spiral data set
Figure 19: CASCELM type 2 results
Figure 20: Schematic showing Full Cascaded ELM
Figure 21: Surface plot demonstrating relationship between accuracy, neurons/cascade and the number of cascade
Figure 22: comparison of ELM and CASCELM for diabetes
Figure 23: Three alternate hidden layer cascade topologies for ELM
Figure 24: comparison of results for three types of hidden layer structure
1 Introduction

1.1 Motivation

Considering how simple the fundamental mechanisms of Neural Networks are, they have proved to be a ripe area for research with many variations on the humble perceptron first developed by Frank Rosenblatt in 1957.

The development of multi-layered topologies solved the early problems of the perceptron only being able to classify linearly separable data - but at the expense of computational complexity. In order to train the hidden layers of these multilayer networks, different algorithms need to be developed and these typically involve long periods of training with iteration. Gradient descent is used to gradually home in on a good solution. However, the data has a profound effect on how well the network will train and these algorithms often take a long time to find a good solution and sometimes get trapped in local minima. For this reason, training times for a multilayer perceptron can be very long, particularly when large amounts of data are involved and performance is very dependent on the topology chosen by the programmer. Often significant experimentation is required to find a suitable topology and feature set for use in training.

CASCOR is a constructive algorithm which removes the need for back propagation, thus reducing training times and enabling the network to arrive at an optimum topology automatically as part of the training process. Training stops as soon as the network achieves appropriate quality when tested with test data. The disadvantage of CASCOR, however, is that it tends to generate very deep networks: e.g. the input signal has to pass through many layers of neurons before it reaches the output. These networks are relatively quick to train but can be slow to compute final data. Several variations on the original CASCOR algorithm have been proposed including Casper, and A_Casper [3]. These variations solve some of the problems associated with the original CASCOR algorithm by reducing the depth of the network.

ELM are a more recent development in neural network algorithms and take a different approach to the problem of training to solve none linear classification. They promise to provide very fast training times and a high degree of accuracy. Their advocates offer many positive reasons for their adoption. In this paper those claims are investigated.

The motivation for this project is to investigate the claims made for ELM and to understand how they function. A further motivation is to combine CASCOR and ELM into a single algorithm CASCELM, and collect data, in the hope this will combine the positive aspects of both algorithms.
1.2 Objective of Project

There are two key objectives to this project:

- To evaluate an ELM and compare its performance to SLFN BP and CASCOR algorithms
- To implement and evaluate a Cascade Correlation Extreme Learning Machine (CASCELM).

Matlab was chosen as the development platform for this project. It has excellent matrix manipulation functions and contains many functions, which are useful for neural network development and testing.

1.3 Contribution

The project contribution is a Matlab project file containing a number of useful programs:

- A Matlab script file containing an implementation of CASCELM
- A Matlab script file containing example code to run the ELM and CASCELM functions. Many of the examples shown in the report can be run from this
- A series of experiments evaluating ELM and CASCELM

1.4 Report Organisation

The report is broken down into chapters. After this introduction, chapters two and three, Extreme Learning Machines and Cascade Correlation begin by introducing the subject and reviewing relevant literature. The theory and algorithms are explained, the implementation is described, experiments discussed and results presented. Chapter four describes the main contribution of this project CASCELM. It describes the algorithm, implementation and experimental results. Chapter five describes experimentation into alternate hidden layer topologies inspired by cascade topologies. Chapter six describes the program code created for the project with instructions on how to use it. Chapter seven is a conclusion together with the identification of areas for further investigation.

1.5 Hardware Specifications

The experimental results described in this report came from code running on a MSI steel series laptop with the following hardware specifications:

- Processor: Intel® Core™ i7-4700HQ CPU@2.40GHz
- Installed Memory: 8.00 GB
- System type: Windows 8.1, 64 bit Operating System

GPU acceleration was not used in any of the experiments.
2 Extreme Learning Machines

2.1 Literature Review

The term *ELM* was first used by Huang, Zhu and Siew in 2006. Their paper “Extreme learning machines: A new learning scheme of feed-forward neural networks” [1] describes the algorithm. Since then many papers have appeared which discuss applications and variations on the algorithm.

*Echo State Machines (ESM)* make use of a very similar algorithm to *ELM* and were introduced by Jaegar, Haas [2] in 2001 predating *ELM* by five years. In this paper they describe how the random hidden layer provides non-linearity between the input and output layer thus allowing the network to train on non-linear data, without the computational expense of training the hidden layer. Their paper focuses on the use of *ESM* for regression in infinite data sets when training machines with recurrent networks. In particular they focus on the application of these machines as frequency generators rather than classifiers.

A very useful paper in understanding the difference between *ELM* and *ESM* by Bin, Yiblin and Xuewen, “Comparison of Echo State Network and Extreme Learning Machine on Non-linear Prediction” [3] explains in detail what the differences are between the networks, and contrasts performance. In terms of algorithm *ESN* is a *recurrent network*, with feedback connections between neurons within the layers and between layers whereas *ELM* are strictly *feed-forward networks*. Both algorithms make use of random values in the hidden layer of the network which remain untrained. In terms of performance the above paper finds that ESN with small data sets are less prone to over fitting, provide substantially higher stability but take longer training time. With big training data *ELM* gives overall better results than ESN. They concluded that ESN are better for small data sets and *ELM* for larger.

2.2 ELM Theory

As previously noted in the standard *SLFN*, a second layer of neurons, known as the hidden layer, is placed between the input and output nodes to allow the network to classify non-linear problems. It has been shown that the additional layer allows the network to be trained to recognise any arbitrary pattern. The well-know back-propagation algorithm is usually used to train the hidden layer. This technique usually works well but can take a long time to train the network to an acceptable quality and is susceptible to local minima. Several refinements have been made to the basic algorithm to improve speed, *Quickprop* [9] and similar. However the fundamental problem of long training time persists.
Huang et al noted that the amount of learning actually achieved in the hidden layer is relatively small compared to the output layer. Their principal contribution was to suggest that a set of random weights in the hidden layer could be used as a way to provide non-linear mapping between the input nodes and the output nodes. By having a large enough number of nodes in the hidden layer the algorithm can map a small number of input neurons to an arbitrarily large number of output neurons in a non-linear way. Training is performed only on the output neurons and performance similar to multi-layer feed-forward networks using back propagation achieved with much reduced training time.

Figure 1 illustrates the topology of an ELM network. In the example there are 2 inputs, 1 output and 3 hidden layer nodes. The hidden layer is initialised with random values. The more nodes in the hidden layer the more weights there will be in the output layer, hence the network will have a higher capacity for training.

![Figure 1: Example of a simple ELM](image)

Training of the network is performed by feeding input data through the hidden layer and training the output layer. As with the traditional feed-forward neural networks this is achieved by multiplying each input training pattern in turn by the weights in each neuron in the hidden layer, summing the result, adding a bias and finally applying an activation function to obtain the output from each hidden layer neuron. This is repeated for each input pattern and is implemented efficiently by representing the test data and hidden layer weights as matrices, multiplying them together, adding on the bias weight matrix and then applying the activation function to each element in the resultant matrix. The output from this stage is a matrix M of dimension m, n where m is the number of samples in the test data and n the number of neurons in the hidden layer.
Training of the output layer can be achieved using a variety of techniques but can conveniently be achieved by noting the following result

\[ M \times H = O \]

Where \( H \) is the hidden layer represented as a matrix of weights and \( O \) are the target training patterns represented as a matrix.

Re-arranging the equation gives \( H = O \times M^{-1} \)

The inverse of \( M \) cannot be computed exactly, it is the solution to a set of simultaneous linear equations, but it can be calculated approximately by finding the pseudo-inverse usually denoted as:

\[ H = O \times M^+ \]

The final matrix will contain the weights are the best least square error fit for the output layer and in addition provide the smallest norm of weights, which is important for optimal generalised performance Bartlett[11]

Calculating the pseudo-inverse is complex but Matlab provides a function \( \text{pinv}() \) which uses the Moore Penrose inverse for this purpose. It is slow to compute and represents the biggest performance cost in the algorithm. There are several ways [10] to calculate the pseudo-inverse and the code includes alternate versions. These other are faster to compute but numerically less stable and require careful fine tuning using a regularisation constant.

At this point it is worth clarifying how the hidden layer works in the ELM. The use of a large array of random values to classify data may seem counterintuitive. However, the random part of the network is not responsible for learning, only the output layer is trained. The function of the hidden layer is simply to provide a non-linear mapping from input nodes to output layer. The size of the hidden layer can be as large as practical, which can be thousands of nodes, and consequently as many weights in the output layer as are required to accomplish the classification.

Another way to consider the above process is to consider the process as a transform from one dimension to another higher dimensional space. The weights in the hidden layer are represented as a matrix and the input data as a set of vectors. The process of feeding the input data through the neurons is most easily performed by multiplying the weight matrix
by the input data. This can also be considered in mathematical terms as a transform. In this case the input data is transformed from N dimensional space to M dimensional space where N is the order of the input vector and M is the number of weights in the hidden layer. If the weights in the random layer are completely random and non-linear then the transform will be non-linear and the input data will be uniformly distributed over the higher dimensions. Once this is done it is only required that one dimension is found where a linear separation is possible in the classification for the data to be classifiable.

It has been proven that if the number of nodes in the hidden layer equals the number of items in the training set then a 100% training accuracy can be achieved. Usually far fewer nodes than this are required to achieve “good” results. In general the more nodes then the better the networks ability to match the training data - but the longer training time will be because of the higher computational load. Unlike the back propagation algorithm, which makes use of gradient descent and is iterative, this algorithm arrives at a solution in a relatively short time interval (depending on the algorithm used to train the output layer).

2.3 ELM Program

A Matlab script for the ELM program was downloaded from Nanyang Technological University Singapore: http://www.ntu.edu.sg/home/egbhuang/elm_random_hidden_nodes.html

As provided the script creates an ELM machine which can be used for either regression or classification. Several data sets are included with the script. The code is quite simple and relatively easy to understand. Initially the test data provided with the script was used to check the code was working.

The algorithm is as follows:

1. Calculate the number of input and output nodes required by analysing the data
2. Produce the hidden layer weights, a matrix of random numbers (scaled into the range -1 to 1) the dimensions of which are the number of input node by the number of nodes in the hidden layer
3. Produce the hidden layer bias weights. This is a matrix of random numbers (range 0 to 1) the dimensions of which are 1 by the nodes in the hidden layer
4. Multiply the input data by the hidden layer weights
5. Extend the bias matrix to the size of the output matrix above and add it to the previous result
6. Apply the chosen activation function to the previous output. It was found that sinc gave the best results but many functions can be used which offer faster execution times and may perform adequately for specific data sets and applications.
7. Find the output weight matrix by using the *Matlab pinv()* to find the pseudo inverse of the above matrix and multiplying it by the output training patterns.

### 2.4 Testing of Program

The diabetes test data was used to train and test a traditional *SLFN* and an *ELM* and the results compared in figure 2. From left to right the three graphs compare training accuracy, Testing Accuracy and Training times. In all graphs data one is *backprop* and data two is *ELM*. As can be seen the *ELM* requires more nodes to achieve the same results as the traditional back prop algorithm but training time is significantly reduced.

![Figure 2: Comparison of training test accuracy and speed for ELM and SLFN](image)

To make it easier to visualise the results an alternative data set was used. The two spiral data set was chosen for this purpose. In this data set two coaxial spirals are generated one of which has a classification of zero and the other a classification of one. The training data consists of x,y pairs. In the next experiment the *ELM* network was used to classify data from the double spiral training set. Data was created using a *matlab* script and used to
train the network. The network was tested using an array of data and the results visualised using a simple scatter graph. Figure 3 shows the results.

![Figure 3: Comparison of training results for double spiral data set](image1)

Figure 3 shows four graphs
- A plot of the test data for reference.
- The second graph shows the test data output from a SLFF with 20 nodes in the hidden layer trained using the Matlab BP algorithm.
- The third graph shows the test data output from an ELM network with 20 nodes as a comparison to the NN back-propagation training
- The final graph shows the output from an ELM network with 200 nodes

Note that although the ELM network requires ten times as many nodes in the hidden layer to achieve the same performance as BP algorithm the training time was still significantly smaller. The approximate training time was .06 seconds compared to 6.4 seconds for the back-propagation algorithm or 100 times faster.

![Figure 4: Comparison of training accuracy as hidden layer size increases](image2)
Figure 4 compares training accuracy with the number of nodes in the hidden layer in an ELM machine. As can be seen accuracy improves steadily up to about 30 nodes and then gradually levels off. Training time increases steadily but even when an accuracy of 95% is reached the training time is still only .01 seconds for the test set used.

2.5 Variation in output due to hidden layer randomisation

Due to the random nature of the hidden layer there is substantial variation between one experimental run and another. This is more evident when there are less nodes in the hidden layer and is illustrated in figures 5 and 6. Figure 5 compares the output from 9 runs from an ELM with 25 nodes in the hidden layer initialised with different random numbers each time. Figure 6 shows the output from an ELM with 50 nodes in the hidden layer. As expected the ELM with 50 nodes performs better than the one with 25 nodes but importantly there is also less visible variation between the worst and best cases.

![Figure 5: Selection of ELM with 25 neurons in hidden layer](image1)

![Figure 6: Selection of ELM with 50 neurons in hidden layer](image2)

To explain this phenomena it is necessary to understand the purpose of the hidden layer in an ELM. Remember that the hidden layer is not trained, its sole purpose is to provide non-linear mapping from input to the output layer usually increasing dimensionality in the process and thus improving the likelihood of training the output layer to provide a good mapping. For this to work satisfactorily the random numbers need to be uniformly distributed over the selected range. Even if the random numbers are uniformly distributed when created any clustering of data in the rows or columns of the weight matrix will produce some linearity in the mapping and hence reduce the learning capacity of the network. The more random numbers generated the less likely it is that there will be significant clustering of weights.
From this analysis it might be tempting to conclude that the more weights in the hidden layer the better the network will perform albeit with the reservation that additional weights in the output layer take somewhat longer to compute. For the double spiral data set this does seem to be the case - but other data sets do not behave as well. Over fitting of training data whilst in the training phase is just as much a problem with ELM as back-propagation. The Figure 7 illustrates this by comparing test and training accuracy for the Diabetes training set.

![Figure 7: Demonstration of over fitting with ELM trained using diabetes data set](image)

As can be seen in the diagram, testing accuracy peaks at only 77% accuracy at approximately 20 nodes and then falls off steadily. The experiment was continued until a hidden layer of 500 nodes was reached, training accuracy reached almost 100% but testing accuracy had fallen off to 70%. A similar result is seen when training using back propagation.

From this it can be concluded that over fitting results if there are too many neurons in the hidden layer. This is dependent on the data set.

### 2.6 Hidden Layer Weight Investigation

During the ELM experiments it became apparent that the random values of the hidden layer have a critical impact on the performance of the network. It is interesting to hypothesise why this is and what constitutes good or bad values.
In a typical NN implementation the weights can conveniently be represented as a matrix, one row for each neuron and one column for each input/weight in each neuron. The operation of applying the weights and calculating the value of the node is conveniently performed by multiplying the weight matrix by the input matrix, thus all of the training pattern’s input can be calculated in the same operation. The process of multiplying one matrix by another can also be considered a transform. In other words we can consider this operation as a random transform which transforms the inputs into a different dimensional space where the number of nodes in the hidden layer is the order of dimensionality of the transformed space.

Each column of the weight matrix can be considered as the axis through which to transform the inputs and because the contents of each column is random the axis we are going to transform into is random. The length of each axis is also random therefore the inputs will be scaled randomly. This process distributes the original data over a larger dimensional space. If the weights are uniformly random then the data will be uniformly distributed in a non-linear manner over the higher dimensional space, thus making it easier to classify.

It was conjectured that if the lengths of the input vectors were preserved during the transform then there would be more linearity in the transform and hence the network should perform worse.

A simple experiment was designed to test this theory. The weight matrix was generated randomly as with the standard ELM algorithm, but an extra step was performed to normalise each row vector of the matrix. When the input vectors are transformed their length will be preserved, the operation of multiplying the inputs by the weights and then summing them is equivalent to the vector projection operation. Normalising the rows or columns of the matrix is easy to do in Matlab using normc() or normr() functions.

Figure 8 compares the results of classification using a hidden layer with and without normalisation and with various ranges of weights. To reduce any chance of variation due to the random nature of the hidden layer the same weights were used in all experiments, generated once and then saved and loaded from a file. The diagram top left is the output from the trained network using random hidden weights whilst the one on to the right was generated after the row vectors of the hidden matrix had been normalised. It was anticipated that the diagram on the right would demonstrate lower learning capacity
As can be seen there is a significant reduction in quality when the rows are normalised which appears to support the hypothesis. However further research revealed an error in the assumption. When the vectors are normalised there is an average increase in length caused by the normalisation. The original vectors are scaled to the range -1 to 1 after they are created, giving an absolute mean of 0.5 but after normalisation the vector has a length of 1 and for a two component vector each component will have a mean length of ~0.707.

Further experimentation revealed that even this small increase in length resulted in a significant reduction in performance for the ELM network. Scaling the weights down after normalization reduced the error. The remaining diagrams show output with the weights scaled into the range -0.5 to 0.5 and -0.1 to 0.1. Once the weights were scaled into the range -0.1 to 0.1 performance was very similar to a hidden layer with non-normalised weights.

Figure 8: ELM training capacity after normalising/ scaling hidden layer
Although results were unexpected, this experiment highlighted the importance of the value of weights on the performance of the network. An experiment was conducted to further investigate this. The data was run through an ELM where the number of weights in the hidden layer and the scale for the weights were modified and the results plotted against each other to create a 3D surface plot and the result shown in Figure 9.

The plot shows that the less nodes in the hidden layer the more sensitive the networks learning capacity is to the magnitude of the weights. In fact small values for the weights, down to about .01 always seem to perform better. Very small values do not work well - probably due to floating point error.
The experiment was repeated using the Diabetes training data and a similar pattern is visible in figure 10. The weight scale needs to be in a larger range, .1 to 200 rather than .01 to 10, to see the same effect. As previously noted the diabetes test set has a strong tendency to over-fit if too many nodes are included in the hidden layer.

It’s unclear why this should be. At first it was thought it may be caused by misclassification after the network has been trained, however analysis of the output was inconclusive. It is related to the magnitude of the input data. If the input data is normalised, like the diabetes training set, the effect is less pronounced but still visible. The spiral data set experiment was repeated after normalizing the data and the results compared with the previous experiments. It was found that the range of weights was less significant, as with the diabetes data set but there is still significant reduction in performance if weights are very large and normalizing the weights and scaling them down does seem to improve performance.

One possible conclusion is that if by chance all the random weights in one neuron are large, then this neuron will perform badly. As the random numbers are randomly distributed this is quite likely to happen for some of the neurons in a large hidden layer ELM. Normalising the weights and then scaling them down ensures that this does not occur and should improve the performance of the network but more experimentation is needed in this area.

Figure 10: Change in accuracy (diabetes data set) as weight range varies
2.7 The application of ELM in Genetic Algorithms

As previously noted, training times for feed-forward neural networks using back-propagation can be very long. This is particularly noticeable when the network is being trained using a large number of features. In many real-world classification problems, the experimental data used to train the network may contain a very large number of independent features, not all of which are relevant to the classification problem at hand. Significant optimisations may be achieved by selecting only the relevant features for training. Unfortunately it is often not obvious which features are important - and the only way to find this out is to train many networks using different combinations of input features to see which perform the best. Genetic algorithms can be used to automate this process. The algorithm is as follows:

1. Create starting genomes – which features are going to be used to train on for the first
2. Train population
3. Select individuals with best performance
4. Perform any mutations used the genome
5. Repeat until exit condition is met

This technique works well for many problems but can result in very long execution times when BP is used to train the individual networks. ELM provides a good alternative to BP in these cases. Additionally, as the key issue here is relative performance between networks trained using different features rather than the overall performance of the network, it is possible to reduce the number of neurons in the hidden layer, employ simpler activation functions and simpler techniques to compute the pseudo inverse of the output when computing the output layer weights.

This technique was applied to an experiment conducted by another student, Xuanying Zhu, at the ANU and she provided the following paragraph summarising her experience of the code:

“This project is about producing a stress curve to display individuals’ stress level by using their physiological signals. To do this, a Neural Network needs to be trained to measure stress. To achieve accuracy, for each iteration, GA is used to filter representative features, which are later used to train the Neural Network. Since it was estimated that such training may cost approximately 13 weeks, we decided to replace the back-propagation with ELM. Then just by one week, we finished the training.”
2.8 ELM Conclusions

*ELM* are very fast to train and provide good results. Conversely the random nature of the hidden layer makes their performance very variable particularly when the number of nodes in the hidden layer is smaller. The range of values for the hidden layer is important otherwise misclassification occurs. Keeping the weights small is an optimum strategy.

Having too many neurons in the hidden layer leads to over fitting and poor generalisation or results. Determining the ideal number of neurons for the hidden layer is problematic and in general can only be done through experimentation and analysis of test data.
Chapter 3: Cascade Correlation

3.1 Introduction

The previous chapter demonstrates how Extreme Learning Machines provide a solution to the problem of long training times caused by training the hidden layer of neural networks. The results are quite conclusive in this regard providing the correct number of nodes in the hidden layer is chosen the network will achieve arbitrarily good training accuracy. Selecting the correct number of nodes is problematic for both ELM and Neural Networks trained using back propagation though.

The Cascade Correlation (CASCOR) algorithm provides a possible solution to the problem of choosing the correct network topology. In this chapter the CASCOR algorithm is described and tested. The algorithm is quicker to train than BP so experiments are run to compare the performance of CASCOR with ELM.

3.2 Literature Review

The Cascade Correlation Algorithm was introduced in 1990 by Scott E. Fahlman and Christian Lebiere in the paper “The Cascade-Correlation Learning Architecture” [4]. In the abstract the authors describe the algorithm thus:

“Cascade-Correlation is a new architecture and supervised learning algorithm for artificial networks. Instead of just adjusting the weights in a network of fixed topology, Cascade-Correlation begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multi-layer structure”

The algorithm is often referred to as a constructive algorithm, it creates the structure of the network as it runs. The process begins with a single neuron which is trained in the usual way using training data. The network is tested with testing data and the accuracy checked to see if it meets the criteria. If it does the algorithm stops, otherwise another neuron is added to the network and the process repeated. The original algorithm has the feature that the weights in the original neuron are frozen once an additional layer has been added. Each successive neuron uses all the outputs from the previous layers as its input.

The training algorithm is efficient and avoids the problem of back-propagating errors through the layers that the traditional hidden layer with back-prop training suffers from. Training time is generally much faster as a result. However the original algorithm tends to generate very deep networks with many neurons cascaded and these structures are rather inefficient to represent and to compute when compared to a traditional neural network.
There have been several refinements to the original CASCOR algorithm. The CASPER algorithm [5] was proposed by Nick Treadgold and Tom Gedeon in 1996. This is a constructive algorithm which builds the network in a similar way to CASCOR. The difference is in the way the weights are trained. In the original CASCOR algorithm weights in all the previous cascades are frozen once a new cascade is added, in CASPER training continues on all weights by means of RPROP. The CASPER algorithm has been shown to create a network with fewer hidden neurons than CASCOR and thus provides greater efficiency. A_CASPER and AT_CASPER[6] algorithms refine CASPER still further and offer various advantages in certain circumstances.

3.3 Performance Evaluation

Two programs were used for this evaluation. One written in C++ the other in Matlab script. Both programs were produced by other students as part of earlier projects and were executed for the purpose of comparing CASCOR performance to ELM. The results from the program written in C++ are shown in the following diagram. From left to right, the outputs are from networks with single neurons in 2,6,12 cascades. As can be seen the accuracy improves steadily as the number of layers increases. Once the network gets above 12 cascades training halts because the test accuracy is 100%.

![Figure 11: CASCOR output when trained with double spiral for 1,6, 12 cascades](image)

A comparison of network size reveals that cascade networks tend to require less neurons to generate similar accuracy to ELM, figure 12. As can be seen in the figure the CASCOR achieves near 100% accuracy with 11 neurons compared to ELM which requires at least 50 for similar results.
In terms of training time the C++ version of CASCOR is much slower than ELM running Matlab. For the C++ program used in the experiments 11 cascades were required to obtain 100% accuracy with the test data and training time was 1.56 Seconds compared to .06 seconds for the ELM algorithm with 200 neurons, figure 13.

**Figure 12: CASCOR accuracy against number neurons**

**Figure 13: CASCOR training time as hidden layer size increases**

### 3.4 Conclusion

With the version of CASCOR written in Matlab cascade networks achieve similar accuracy with less nodes to ELM but training times are much higher (in the order of minutes). Examination of the code suggests that it is not very efficient in its use of the Matlab matrix manipulation functions but it is not known if this is the reason it is so slow or if it is the algorithm. Further experimentation is required in this area to clearly understand the difference.
4 CascELM

4.1 Introduction

As previously noted ELMs can be trained very quickly to solve classification problems. In general the larger the hidden layer the higher the learning capacity of the network. However the size of the hidden layer is critical to performance. Too small and the network will not have sufficient capacity to learn but too large, learning times will suffer and overfitting occurs. Finding the ideal size for the layer is problematic. If the number of nodes is greater or equal to the number of training patterns then the network will be able to achieve 100% learning. However this is not a useful conclusion as in most cases we would expect the network to achieve satisfactory learning with far less nodes than this. The random nature of the hidden layer further exacerbates this problem of finding the ideal layer size because depending on the random weights added we may require more or less weights.

It would be convenient if we could start with a relatively small number of weights, test the network and if performance is substandard gradually add more weights. In this section we explore a modification of the ELM architecture which makes this approach possible. The principle is the same as the cascade networks explored earlier. As previously noted these networks start with a single layer, which is trained, and gradually add more layers until the network reaches a satisfactory learning capacity. The following sections describe an adaptation of the approach using ELM.

4.2 Shallow Cascade ELM

The first experiment involved the addition of a cascade in the existing ELM structure. The term Shallow Cascade was adopted for the algorithm and Figure 14 shows the general principle. When training the input pattern is presented to the random weight input and the output from the random weights calculated as per the standard ELM. At this point the cascade occurs. The input data matrix is joined onto the output from the random layer. This becomes the input to the final layer of the ELM and training is achieved by presenting the training pattern to the output and calculation of the Moores Penrose matrix.
4.2.1 Shallow cascade ELM results

The code modification required to do this is quite simple, we join the matrix which contains the input values to the network onto the output from the activation stage. This results in a matrix with additional rows.

Figure 15: Comparison of ELM and shallow cascade ELM training accuracy

Figure 15 shows a graph comparing the performance of ELM and cascade ELM trained using the double spiral data set. Series 1 and 2 plots show the respective accuracy as the number of nodes in the hidden layer is increased from 1 to 60. As can be seen performance of both types of network start of the same but the cascade network shows a distinct advantage from 5 to about 25 nodes. Above that number the difference is less noticeable.
As the number of nodes in the random layer increases so the relative effect of the cascade becomes less noticeable so we would expect the two topologies to provide similar performance at the higher number of nodes but the advantage that the cascade produces is surprisingly large. The Shallow cascade ELM took on average ~8% longer to train.

Figure 16 compares two spiral test data output from the trained network. On the left is an ELM with 20 nodes in the hidden layer, on the right the same data from a cascade ELM with 20 nodes. As can be seen the addition of the cascade provides a substantial improvement in learning ability.

![Quality comparison, ELM and shallow cascade, 20 nodes in hidden layer](image)

**Figure 16:** Quality comparison, ELM and shallow cascade, 20 nodes in hidden layer
4.2.2 Single Cascaded ELM

For the next experiment a sequence of shallow cascade ELM machines were cascaded together. Two topologies were tested and the results compared. When comparing these topologies it helps to consider each ELM as a self-contained unit. In each ELM the input to the random layer is from the input to the network. This is combined with the cascaded data from previous layers. In the first topology only the output from the previous layer is cascaded. Figure 17 illustrates the general arrangement of data flow in a four layer cascade:

![Diagram of a four layer cascade](image)

**Figure 17: schematic of type 1 CASCELM**

Training of the network is performed as follows:

1. Input data is presented at layer 1 and fed forward through the random generated hidden layer.
2. The Moores Penrose inverse is used to calculate the values for the output layer using the training data for the network.
3. The weights for the output layer are then fixed.
4. The algorithm continues to the second layer.
5. Training is very similar except that before the output layer is trained the output from the previous layer is combined with the output from the hidden layer.
6. Training is very similar except that before the output layer is trained the output from the previous layer is combined with the output from the hidden layer. until all layers are trained or the error margin in the output is within a target range.

The main feature of this algorithm is the feeding of the previous cascade into the trained layer of the next cascade rather than into the random layer. This is important because if the cascade is fed into the random layer any correlation learnt in the previous cascade will
be lost again. On the other hand the input data needs to go through the random layer before it can be used for training.

![Figure 18: CASCELM type 1 results for double spiral data set](image)

Performance of the network was compared for various numbers of nodes and layers and the results plotted in figure 18. The four graphs show training and test accuracy from a type 1 CASCELM with 1, 5, 10 and 20 nodes in each cascade.

The data shows that the ability of the network to learn improves slightly as cascades are added but generally only for the first few cascades. The number of neurons in each cascade has a more significant effect on its learning capacity.

This is easily explained when the topology of the network is considered in more detail. Clearly the amount of information which can be stored in each cascade is limited by the number of neurons in the output layer and if only the previous layer contributes to the next cascade then as cascades are added the network rapidly reaches its full learning capacity. When there are many cascades then earlier cascades will have little or no effect on the result.

### 4.2.3 Full Cascaded ELM

To try and improve the performance of each layer the cascade was extended to include all the outputs from the previous layer.
Figure 19 shows the general arrangement of the cascade structure used. The algorithm for training the network is almost the same as with the previous example except all the outputs from the previous layers are combined when training the next layer. The same data was run through this network and the results shown in Figure 20.

The graphs show that even with only 1 neuron in each cascade the network is capable of reaching 95% accuracy with test data. As might be expected the more neurons in each cascade the less cascades are required to reach a high degree of accuracy.
Figure 21 demonstrates the relationship of neurons/cascade, number of cascades and accuracy in a single surface plot. The shape of the surface along the two axis shows:

(x) A pure ELM as the number of neurons in the hidden layer increases
(y) A pure cascade machine with one neuron in each hidden layer

4.2.4 Comparison of ELM and CASCELM

CASCELM training times are longer than simple ELM but not excessively so. To train a network with 5 neurons per cascade and 30 cascades took 0.0781 seconds on the experimental machine this compares to .02 seconds for a simple ELM machine with 40 neurons in the hidden layer which achieved slightly better accuracy levels, 98% compared to 96%. In general the CASCELM as implemented in these experiments was unable to reach the accuracy levels of a simple ELM machine, without excessively large numbers of neurons in each cascade, but this may be an implementation issue which can be resolved by future experimentation and refinement of the algorithm, see further work.

Figure 21: Surface plot demonstrating relationship between accuracy, neurons/cascade and the number of cascade
Figure 22 compares the performance of \textit{ELM} and \textit{CASCELM} networks with the same number of weights in the hidden layer. The left diagram has two nodes in each layer the right four. As can be seen \textit{ELM} outperforms the \textit{CASCELM} in most cases but eventually does reach approximately the same accuracy if enough cascades or enough weights are added. Training times are approximately the same for similar numbers of weights.

![Figure 22: comparison of ELM and CASCELM for diabetes](image)

### 4.3 Conclusion

In this chapter three new algorithms have been introduced which enable \textit{ELM} to be used in new ways. The shallow cascade provides an improvement in training performance for most data sets. The \textit{CASCELM} architecture with one cascade was shown to work but did not give very good performance. The \textit{CASCELM} architecture with a full cascade was shown to provide good training accuracy but at the expense of more neurons and training time when compared to standard \textit{ELM}. It has the advantage though that it is a constructive algorithm and can therefore start with a simple architecture and gradually build up to a sufficiently complex one in less time than it would take to create multiple \textit{ELM}. More research and development is required to refine the algorithm to the point where it could be used but the experiments detailed here show that the theory is sound and the algorithm works as predicted.
5 Cascaded weights in ELM

5.1 Introduction

This section discusses experiments with alternate topologies for the random weight layer in the ELM. In the basic ELM the weights in the random layer are connected to the input and output layer but not to each other. This is simple to implement and performs well. In this second we experiment with the topology to see if adding additional connections

Fig 23 shows the three network topologies discussed in this section. The first network is the standard ELM topology for input, hidden and output weights. As previously discussed the hidden layer is initialised to random values and only the weights at the output layer are trained. As can be seen the hidden layer nodes get their input only from the input nodes and the output layer gets its input only from the hidden.

![Diagram showing three network topologies for ELM]

Figure 23: Three alternate hidden layer cascade topologies for ELM

The second diagram shows the first cascaded hidden layer topology. This topology is identical to the previous except that each hidden layer node, except the first one, takes the output from the previous neuron as an additional input. The third diagram shows the second cascade hidden layer topology. In the topology the outputs from all previous nodes used as additional inputs to each neuron.

5.2 Experiment

The ELM matlab code was modified to include two new hidden layer functions to implement each of the new cascade types. This involves the addition of a third type of weight. The weights which modify the output from the previous neuron in the cascade to
the new input. The number of weights in the output layer, e.g. the number the weights which are trained, remains the same regardless of the topology of the hidden layer cascade.

A script was produced which would load data and then test the three topologies with different numbers of nodes in the hidden layer so as to compare performance. Because of the random nature of the hidden layer nodes the output is quite noisy so to smooth it out each test is run 20 times and the results averaged. Figure 24 compares the three types of cascade using the diabetes training set, similar results were obtained with the double spiral set.

In all cases the simple hidden layer topology is superior. The single cascade is slightly worse and the more complex cascade worse still. Additionally the extra complexity of the algorithm results in longer training times.

![Figure 24: comparison of results for three types of hidden layer structure](image)

### 5.3 Conclusion

The experimental results show that the addition of extra connections between the neurons in the hidden layer, and additional random weights do not improve the performance of the network. Further experiments were tried where the additional cascade weights were set to on, performance remains the same. This suggests that the addition of the cascade does not improve the classification performance of the ELM.

This result is slightly surprising as one might expect that the addition of extra weights would improve the ability of the hidden layer to non-linearise the input data. However without additional weights to train in the output layer the training ability of the network is limited. Further the computation overhead involved in adding extra weights and performing the cascade in the hidden layer depends on the number of neurons but is significant. Therefore the conclusion is that adding a cascade to the hidden layer is not recommended for ELM.
6 The Program

The ELM implementations was coded in Matlab Script. An existing C program was used for the CASCOR implementation, although some modifications were required for it to work with the current version of Visual Studio (VS12).

6.1 Matlab script Instructions

Copy the ELMToolbox directory to a suitable location on a computer. The project requires that there is a version of Matlab installed on the computer before it will run. Open the script runExperiment.m and run it. Matlab will probably ask to change the working directory, enter yes. An option menu will appear in the console asking the user to choose which experiment to run. Choose from the following options

1. ELM machines with hidden layer count incrementally increasing from 1 to 60 using diabetes data
2. Comparison of training data, NN, ELM with double spiral data set
3. Graph showing relationship of accuracy to hidden layer size with double spiral data
4. Graphical representation of variation in hidden layer with 25 and then 50 neurons in hidden layer
5. Demonstration of over fitting in ELM using diabetes training set
6. Demonstration of normalizing weights and scaling of weights
7. Shallow cascade experiment 1: Accuracy plotted against number of hidden neurons
8. Shallow cascade experiment 2: Qualitative comparison of double spiral data
9. Cascade type 1 experiment comparing 1, 5, 10 and 20 neurons in each cascade
10. Cascade type 2 experiment comparing 1, 5, 10 and 20 neurons in each cascade
11. Cascade type 2 surface plot showing accuracy for neurons in each cascade against number cascades
12. Comparison of ELM and CASCELM training times
13. Comparison of ELM and CASCELM with double spiral training set
14. Cascade weight experiment

6.2 Program code description

There are four main scripts files required for the ELM experiments:

ELM.m
Script to run a basic ELM. This is a modified version of the code downloaded from The University of Singapore's website. It has been modified to make it more readable and slightly better structured.

**multiCascadeELM.m**
Script to create a CASELM. This script can create any of the cascaded ELM discussed in this report including the Cascaded weight ELM.

**NN.m**
Script to set up and run a back propagation neural network using the Matlab Fitnet function.

**runExperiments.m**
Script to run all the experiments in this report.

In addition a small script, twospirals.m, is used to create the data for the two spiral data set which was downloaded from this web page:


Each of the scripts will be described in detail and individual functions listed:

### 6.3 ELM.M

This script is largely the same as the script downloaded from the University of Singapore website. The main changes involved renaming variables and the addition of comments to make the code more legible. The main ELM function creates and runs the ELM. Parameters are listed in the following table.
The function performs as described in the earlier section of the report. In summary:

1. Input data is parsed to determine how many classifications are required. One output node is created for each classification.
2. Matrices are created for hidden layer weights and biases using random numbers.
3. Training data is fed forward through the hidden layer neurons, an appropriate activation function is applied.
4. The output layer is trained using the Moore Penrose Pseudo inverse algorithm, `pinv()` in MATLAB.

On completion the function returns the following values:

<table>
<thead>
<tr>
<th>Output name</th>
<th>Output description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrainingTime</td>
<td>Time in seconds to train the network</td>
</tr>
<tr>
<td>TestingTime</td>
<td>Time in seconds to test the network</td>
</tr>
<tr>
<td>TrainingAccuracy</td>
<td>The accuracy of the trained network when tested with training data(0 to 1)</td>
</tr>
<tr>
<td>TestingAccuracy</td>
<td>The accuracy of the trained network when tested with testing data(0 to 1)</td>
</tr>
</tbody>
</table>
There are three small functions which are also used:

- \( \text{min2}(A) \) return the smallest value in a 2 dimensional matrix A
- \( \text{max2}(A) \) return the largest value in a 2 dimensional matrix A
- \( \text{normalize}(A) \) return the normalized version of matrix A

### 6.4 multiCascadeELM.m

This script makes and runs a cascade ELM. The cascade function itself is developed from the original ELM code but is very heavily modified. To make the cascade easier to implement the original code has been refactored with several new functions adding.

The arguments for the script are listed in the following table:

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Argument description</th>
</tr>
</thead>
<tbody>
<tr>
<td>train_data</td>
<td>array of training Data</td>
</tr>
<tr>
<td>test_data</td>
<td>array of testing data</td>
</tr>
<tr>
<td>ELM Type</td>
<td>0 = regression, 1 = classifier</td>
</tr>
<tr>
<td>NumberOfHiddenNeurons</td>
<td>The number of neurons in the random layer</td>
</tr>
<tr>
<td>Activation Function</td>
<td>String representing the activation function to use. Choices are:</td>
</tr>
<tr>
<td></td>
<td>sig for Sigmoidal</td>
</tr>
<tr>
<td></td>
<td>sin for sine</td>
</tr>
<tr>
<td></td>
<td>hardlim for Hardlim</td>
</tr>
<tr>
<td></td>
<td>tribas for triangular basis</td>
</tr>
<tr>
<td></td>
<td>radbas for radial basis</td>
</tr>
<tr>
<td>display_data</td>
<td>Data set to use when displaying a graphical plot of the data, can only be used when</td>
</tr>
<tr>
<td></td>
<td>there are only two features to train on</td>
</tr>
<tr>
<td>cascadeMode</td>
<td>What type of cascade to use: 0 is no cascade,1 and 2 create cascades of the types</td>
</tr>
<tr>
<td></td>
<td>described in this report</td>
</tr>
<tr>
<td>NumberLayers</td>
<td>The number of layers in the cascade</td>
</tr>
<tr>
<td>Display_graphic</td>
<td>1 = display a graphical plot of the data.</td>
</tr>
<tr>
<td>randomCascadeMode</td>
<td>Selects how the hidden layer cascade works. 0 is no hidden layer cascade, 1,2 select</td>
</tr>
<tr>
<td></td>
<td>random layer cascade as described in this report</td>
</tr>
</tbody>
</table>

On completion the script returns the following values
The following table lists all the functions in the code:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiCascadeELM</td>
<td>Main entry function for creating a Cascade ELM</td>
<td>As shown in input table</td>
<td>As shown in the output table</td>
</tr>
<tr>
<td>scatterPlot</td>
<td>Produce a scatter plot using the display_data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trainNetwork</td>
<td>Train the network,</td>
<td>Input pattern, output pattern</td>
<td>Result of passing input pattern through the trained network</td>
</tr>
<tr>
<td>testNetwork</td>
<td>Test a network</td>
<td>Input pattern</td>
<td>Result of passing input pattern through the trained network</td>
</tr>
<tr>
<td>activationFunction</td>
<td>Apply the selected activation function to a matrix of values</td>
<td>Matrix to apply the function to</td>
<td>A matrix which contains the result of applying the activation function</td>
</tr>
<tr>
<td>trainCascade</td>
<td>Train an individual cascade in the network</td>
<td>CascadeNumber, Input to the hidden layer, training data, cascade input</td>
<td>Result of input data passed through the cascade</td>
</tr>
<tr>
<td>testCascade</td>
<td>Test an individual cascade in the network</td>
<td>CascadeNumber, Input to the hidden layer, Cascade Input</td>
<td>Result of input data passed through the cascade</td>
</tr>
<tr>
<td>randomLayerFeedForward</td>
<td>Feed data forward through the random layer</td>
<td>Cascade Number, Input to the hidden layer, Cascade Input</td>
<td>Output from the hidden layer</td>
</tr>
</tbody>
</table>
The parts which are taken directly from the original script are commented.

The functions TrainNetwork, TestNetwork, are responsible for controlling the cascades. To keep the code simple an array of hidden layer weights and biases are created when the script is first called. This is more efficient in *matlab* than creating cascades as required but does mean that the number of cascades has to be pre-decided. This is not a requirement of the algorithm however and efficient ways are available to create new cascades as the algorithm runs if required.

Each cascaded *ELM* machine has two inputs, one which goes to the random layer and one which is added to the output of the hidden layer before it is fed through the activation function and into the output layer. The input to the hidden layer is always the input to the first stage of the *ELM* machine. There is no point in feeding the cascade from previous layers in at this point as the random nature of the hidden layer would simply remove any learning the previous stage had accomplished.

The output from the cascade is either,
• Thrown away – no cascade
• Replaces the previous cascade – cascade type 1 as described in the report
• Concatenated with the other cascade inputs – full cascade as described in the report.

TrainCascade, TestCascade are the functions which train and test individual cascaded ELM machines. The functions are called by TrainNetwork and TestNetwork respectively as each new cascade is added or tested.

6.5 NN.m

This script is a wrapper for the Matlab fitnet() function so that it can be called using data which is presented in the same format as that for the ELM and cascade ELM scripts and the results compared.

6.6 runExperiments.m

This allows all the experiments described in this report to be easily run. To use the script select the appropriate experiment by setting ExperimentNumber to an integer between 1 and 10.

6.7 Development and Testing

The code was developed over a twelve week period. During that period many different versions of each routine were produced, tested and refined. The scripts provided are the refined and optimised versions. Where possible Matlab matrix operations are used so as to improve efficiency.

Each separate routine in the scripts was tested using simple unit tests to ensure that each individual stage worked as expected. This is important when working with a complex algorithm such as CASCELM.

In total ~600 lines of Matlab script were produced.

6.8 C Code

The directory C_Casper contains the solution and source files required to build the CASCOR code. As set up the compiled executable can be either be run from visual studio or executed as a stand-alone application. If it is run from visual studio then the debug menu needs to be set up to pass the correct parameters to the executable.
Parameters are:

Classification mode
Decay
Training Length
Seed
Adecay
Max_tower
Max_neuron
Description
Net_file

Refer to the paper on CASCOR/CASPER for further information what these parameters are for.

The Visual studio project is currently set up to pass in the following parameters:

```
Original_C_Casper.exe 1 0.1 200 0 1 12 1 doubleSpiral
```

Doublespiral.net contains the training, test and validation data for the double spiral data set coded into a .net file.
7 Conclusion and further work

7.1 ELM Conclusion

When compared to Neural Networks trained with back-propagation ELM provides a significant improvement in training times. Results are dependent on the data set used but it is not unusual for ELM to reach the same level of accuracy in a hundredth of the training time required for back-propagation. Further in the case of the experiments described in this report the Matlab fitnet implementation was used and this is a sophisticated version of back-propagation so these improvements are even more impressive.

CASCOR offers several advantages over back-propagation. In the experiments discussed in this report Cascade performed ~4 times as fast as back-prop but these results might be influenced somewhat by the different implementation platforms used. Assuming that we can compare these training times then ELM training times are approximately 25 times faster than CASCOR with the data sets used in these experiments. CASCOR does offer the advantage that it is constructive and hence can reach an optimal topology in less time. Experiments were not conducted into any of the variants of CASCOR, e.g. CASPER but it is known that these offer further advantages over CASCOR. Referring to the data reported in other papers, ELM is still the clear winner in terms of training time.

The results from the ELM experiments clearly show that this algorithm works. It can achieve very fast training times. Because the hidden layer is randomly created the performance of the network is very variable. By increasing the number of nodes in the hidden layer the networks performs more reliably but becomes prone to over fitting of training data. Finding the best size for the hidden layer is often a matter of experimentation but the fast training times make this less of a problem than with back-propagation.

The extremely simple nature of the ELM algorithm is another of its big advantages and makes implementation on a variety of different hardware, including GPUs [8], very straightforward.

7.2 CASCELM Conclusion

The main contribution of this project is the development of a new type of neural network which combines ELM with Cascade. After several experiments a viable architecture was arrived at in which ELM machines are cascaded. The report details the architecture.

The addition of a single cascade in the shallow cascade architecture gives a small advantage when there are only a few weights in the hidden layer. As the number of weights
increases so the advantage reduces. But this architecture allows the ELM to be cascaded together into bigger machines

With the type one cascade mentioned in the report we see that performance of the network is worse from an accuracy POV than a simple ELM when we compare both training time and number of weights required.

With type two cascade the performance of the network is slightly worse than that of a simple ELM with comparable numbers of weights and achieves about 75% of the accuracy. Training times are ~10 times longer. On the face of it these experiments suggest that CASCELM is worse than ELM and not worth investigating further. However there are possible advantages to using the CASCELM

In conclusion whilst it is true that CASCELM does not perform as well as ELM it does work as an algorithm. Further research is required to understand how and where it could be improved.

7.3 Further work

CASCELM was only implemented in a basic way for this project. Experiments showed that the theory works but many refinements could be made. In the implementation provided here the number of cascades generated is pre-decided, in a more sophisticated implementation the algorithm would be modified to add a cascade, test the machine with testing data and then stop if required accuracy was reached. In this way the network would avoid the problem of over-fitting identified earlier.

7.3.1 Improving the performance of ELM

As identified the main factor which determines the performance of an ELM network is the random values which are placed in the hidden layer. Finding optimal weights for the hidden layer can currently only be achieved by generating a selection of networks and comparing their performance. Clearly this is not ideal and it would be better if some sort of algorithm could be used to generate the ideal weights for a particular data set. Whilst this might not be completely possible it is probable that a better solution is available than simply randomly creating a matrix each time and discarding any that doesn’t work well. In this report it was shown that scaling the matrix weights down improves performance and it seems likely that clusters of large weights in neurons can cause problems. Better distribution of weight values over all neurons which avoid clusters of large and small values seem likely to improve performance particularly with relatively small hidden layers.
Further investigation is required to understand better what constitutes a good or bad set or weights for the hidden layer and if techniques exist which might produce better sets of weights or at least reduce the chances of weights being generated which are bad. This is a problem with the algorithm at the moment and a major stumbling block to its adoption.

7.3.2 Improving the performance of CASCELM

As each new cascade is added the change to accuracy when tested with testing data should be evaluated. In this way it would be possible to check if the new random layer which was added improved or worsened the accuracy of the network. If the performance did not improve with the addition of the new cascade, or only slightly improved, then there would probably be an advantage in recreating the random layer for the new cascade and retesting. In this way the problem of adding “bad layers” to the network could be avoided and because the layers are relatively small a much more efficient set of layers could be created than with the traditional ELM structure. It might even be possible to re-use previous layers if they proved to be very good. If after trying to add several cascades performance had not improved significantly then training would halt. Changes to the code required to do this would not be too major but are beyond the scope of this project.

7.3.3 Use of CASCELM for classification of infinite data

The existing ELM architecture can only be used with finite sets of data. With the single cascade architecture the algorithm could potentially make use of infinite data. The algorithm would work thus:

1. Read the data into a suitable buffer.
2. Train the first cascade on the data which is in the buffer
3. Read the data into a suitable buffer.
4. Train the second cascade on the data which is in the buffer
5. Discard the first cascade
6. Repeat...

Each subsequent cascade would retain some of the knowledge of the previous cascade. As data is presented to train the network it is hoped that the network would gradually adapt to any changes, acting as a kind of filter. It might be possible to refine this approach with more than two cascades, combining the output from the previous three cascades for example. There is a great deal of possible refinement which could be achieved by experimenting with the topology of the network.
References


Appendix A
INDEPENDENT STUDY CONTRACT

Note: Enrolment is subject to approval by the projects coordinator.

SECTION A (Students and Supervisors)

UnilD: u4750194

SURNAME: Oakden FIRST NAMES: Anthony

PROJECT SUPERVISOR (may be external): Tom Gedeon

COURSE SUPERVISOR (an RSCS academic):

COURSE CODE, TITLE AND UNIT: COMP8740

SEMESTER □ S1 x S2 YEAR: 2014

PROJECT TITLE:
Cascade networks and Extreme Learning Machines

LEARNING OBJECTIVES:
- Familiarity with Cascade neural networks (Casper)
- Familiarity with Extreme Learning Machines (ELM)

PROJECT DESCRIPTION:
- Brief initial literature survey of cascade neural networks and extreme learning machines.
- Implement ELM in Matlab.
- Modify existing Matlab code for Casper to better approximate simple layered topologies.
- Experiment with layered Casper with ELM layers.
- Testing with previous benchmarks.
- If time permits implement a full Cascade ELM structure.
- Write report.

Research School of Computer Science

Form updated Feb-14
ASSESSMENT (as per course’s project rules web page; with the differences noted below):

<table>
<thead>
<tr>
<th>Assessed project components:</th>
<th>% of mark</th>
<th>Due date:</th>
<th>Evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report: name style:</td>
<td>45</td>
<td></td>
<td>(60%)</td>
</tr>
<tr>
<td>(e.g. research report, software description, …)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artefact: name kind: Matlab software, test results</td>
<td>45</td>
<td></td>
<td>(30%)</td>
</tr>
<tr>
<td>(e.g. software, user interface, robot, …)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation:</td>
<td>95</td>
<td></td>
<td>(10%)</td>
</tr>
</tbody>
</table>

MEETING DATES (IF KNOWN):

...ekly ...

STUDENT DECLARATION: I agree to fulfil the above defined contract:

Anthony Oakden

Signature

14/7/2014

Date

SECTION B (Supervisor)

I am willing to supervise and support this project. I have checked the student’s academic record and believe this student can complete the project:

...Cedea

Signature

24/7/14

Date

REQUIRED SCHOOL RESOURCES (IF KNOWN):

SECTION C (Course coordinator approval)

Signature

25/7/14

Date

SECTION D (Projects coordinator approval)

Signature

Date

Research School of Computer Science

Form updated Feb-14