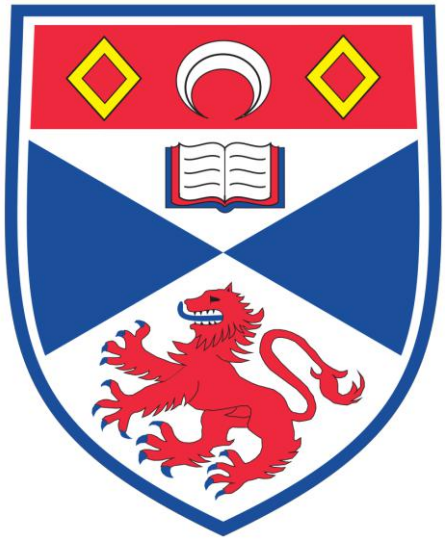
Word Count: 3526

ASSIGNMENT 1:
'GUESS WHO' ARTIFICIAL NEURAL NETWORK

CS5011 - Artificial Intelligence Practice



170009629

pmh20@st-andrews.ac.uk

OCTOBER 10, 2017

UNIVERSITY OF ST ANDREWS

Table of Contents

List of Abbreviations

Artificial Intelligence	AI
Machine Learning	ML
Artificial Neural Networks	ANN
Deep Neural Networks	DNN
Natural Language Processing	NLP
Finite-State Automaton	FSA
Reinforcement Learning	RL
Multi-Layer Perceptron	MLP
Genetic Algorithm	GA

1.0 Introduction

Within the field of Artificial Intelligence (AI), Machine Learning plays a very large role in some of the most notable achievements to date. A popular subset of Machine Learning, that has brought a lot of attention towards AI, is Artificial Neural Networks (ANNs). In recent years, ANNs (in particular Deep Learning Neural Nets) have made some incredible advances in classification tasks, Speech Recognition/Natural Language Processing (NLP), and complex Pattern Recognition. For this reason, ANNs are a predominantly important area of AI to have an understanding and working knowledge of.

For this assignment, a piece of software containing a neural network was created, and then used to play the game 'Guess Who'. This involved designing a fairly small shallow network in order to classify between characters, based on the users answers.

This particular classification task may not necessarily require a neural network, as it could potentially be solved by a sequential set of logical steps. However, it gives a very clear representation of the process and functionality of an ANN. Additionally, the task allows crucial experience to be gained in the setting of network parameters, giving a better understanding of the effects that architecture manipulation may have on the learnability.

2.0 List of Parts

As these details are covered in the design section, please see Appendix A for a list of parts.

3.0 Literature Review

Machine Learning (ML) is a section of AI that is based on the computer's ability to learn from data/experience, without the need to be programmed precisely for every possible scenario. Within ML, there a many various techniques that can be employed, but this assignment (and ergo, this literature review) will focus on Neural Networks. Covered in this section are some relevant research contributions that will give better understanding of ANNs.

One application that neural networks have recently helped progress, is modelling language processing. As human language capabilities come from processes in the human brain, ANNs (also a connectionist model, based on the brain) prove to have an essential starting point for NLP **[1]**. A common technique that works very well in NLP is the use of Finite-State Automaton (FSA). For this, simplistic recurrent neural networks have proven their abilities in being trained to implement FSAs **[1]**.

Even though ANNs have only been introduced to speech processing problems in recent years, they already match the effectiveness of many of the conventional NLP techniques, in some cases even outperforming them **[2]**.

Another large space that ANNs are growing in, and have a lot of potential scope in, is the medical field. Here they can be used for such tasks as patient diagnoses, urological feature extraction, cervical smear screening, and analysis of sleeping disorders **[3]**.

An important problem in this field that ANNs are being researched in attempting to solve, is the task of classifying cancers into diagnostic procedures, based on gene expression signatures **[4]**. This indicates with a complicated example, how the ANN in this assignment will be used to solve the task at hand, by means of classification.

Design

4.1 Part 1

4.1.1 Point 1

To avoid too much manual effort, all the data used in the training table were automatically read in to arrays, in order to give easy access when training later in the code.
To do this, a fairly simple class was from CS5001 – Practical 1 (written by Jonathan Lewis) was utilised. The *readFile*() method in this class reads the lines from a text file in to an Array List, and returns these strings in to an array of ‘paragraphs’.
Splitting this array (using a comma as the token delimiter) allowed each cell from the original spreadsheet to be stored in its own element location.

Doing it this way made it easy to access the features (questions), and also made it simple to pair each output from the long list of names, with its corresponding input matrix.

One key mechanism included in this block of code (*createDataset*() method), was that is was designed such that it would work with any text file dataset that is provided. This allowed for scalability when increasing or replacing the dataset and/or questions later in the assignment, with no alteration needed in the method. This gives the method an element of abstraction.

4.1.2 Point 2

To make the data supplied usable, the network required it to be in numerical form. At this stage, the task only involved yes/no answers in the data inputs, so for simplicity these were mapped to **1** for “yes”, and **0** for “no”.

Another reason why 1 and 0 were used is because the Sigmoidal Logistic function (the activation function used in this ANN) does not work well if the numbers are extreme. For example, if the stimulation for a particular neuron is calculated to be above 10 (or below -10), the result will always tend towards 1 (or 0). This lack of distinguishability makes it hard for the network to learn, and therefore is best practice to keep the inputs low, which reduces the need to normalise in these simple networks.

The output had to also be in a numerical format, instead of the ‘name’ strings provided. For this, the ‘one-hot’ encoding method was chosen. This was selected over binary encoding, because for classification tasks, the neural net tends to perform better when the classes are represented by completely separable neurons. The result is that the ANN does not get mixed up when predicting classes as easily as it might with binary (although both methods would still work well). Binary encoding may be a good option when the number of varying classes is enormous, and the programme would find it too costly to have a single output per class. However, the size of training data in this task is nowhere near the scale where that would become an issue. In addition to this there is bonus advantage of using ‘one-hot’, to do with probabilistic certainty prediction, but this will be discussed further in a later section.

The specific encoding for the task in **Part 1** and **2** is depicted in ***figure 1*** below.

	Target Output Values				
Class (Name)	**Output neuron 1**	**Output neuron 2**	**Output neuron 3**	**Output neuron 4**	**Output neuron 5**
"Alex"	**1**	0	0	0	0
"Alfred"	0	**1**	0	0	0
"Anita"	0	0	**1**	0	0
"Anne"	0	0	0	**1**	0
"Bernard"	0	0	0	0	**1**

Figure 1 – Encoded names

4.1.3 Point 3

Setting up the network involved invoking premade methods from libraries within the 'Encog' framework. This meant that the complicated backpropagation calculus and matrix multiplication could be abstracted away from, leaving the focus to be put on understanding the neural network as a whole.

This point did not require many decisions, as the number of layers and activation type were specified in the coursework, and the number of input and output neurons are determined from the number of features and classes respectively.

It was selected that both the hidden and output layer would be connected to biases. Although they are not always needed, it seems that it is always safer to have them, as the cost is so low for what could potentially make a non-converging network work perfectly. It does this by allowing lateral motion in the resolving of the sigmoidal logistic function. In this sense, the bias acts in a manner analogous the constant 'C' in the formula for a line:

$$y = mx + C.$$

The only disadvantage to using biases unnecessarily, is that additional weights are being used when perhaps not needed. However, in the scheme of the entire software, this will have absolutely negligible effect on the training of the ANN.

4.1.4 Point 4

Deciding upon the number of hidden units in an ANN is extremely important, as it directly relates to the level of overfitting/underfitting that the trained network will experience. This is often hard to detect, and can only be truly determined after conducting tests (including noise test) and measuring the error rate of predictions. To combat this negative effect, certain techniques were employed to determine the optimal number of hidden neurons, and will be discussed further in **Point 8**.

As a general rule-of-thumb, the following formula was used to calculate that 10 hidden nodes could be a rough starting point. See calculation below:

$$\#\ Hidden\ units = \frac{2}{3} \times \#\ Input\ units + \#\ Output\ units$$

$$\#\ Hidden\ units = \left(\frac{2}{3} \times 7\right) + 5 = \mathbf{10\ \boldsymbol{hidden\ units}}$$

4.1.5 Point 5

For training the network in this section, the backpropagation algorithm was implemented. Much like creating the network, ‘Encog’ provide classes to carry out the training behind the scenes.

The main choices that were made here include the momentum, the learning rate, and the end-point error. These are very important parameters, as they not only alter the time taken to learn, but also the stability of the trained network in making consistent predictions. A particular example of this is to set the convergence error too low (0 for example), which would also cause overtraining if the ANN were able to converge. This is because the function solution within ‘black box’ would be too specific to the training data, and would unlikely be able to recognise ‘noisy’ inputs, or a slightly skewed pattern.

These three parameters each have a trade-off between learning time and network accuracy, but the optimal combination of values can be incredibly hard to discover by trial-and-error, which is why neural nets are often described as an art rather than a science. However, there are some optimisation techniques that could be used such as Genetic Algorithms, which will be discussed further in **Part 3**, **Point 1**.

4.1.6 Point 6

The trained ANN was then saved to a file in the current ‘Learning1’ directory. This was then manually copied into the project directory of ‘Learning2’, which allowed the programmed game to use the best-trained network.
The file was transferred manually as the projects must be submitted as separate .jar files, each containing the relevant files. Also, writing directly to the ‘Learning2’ directory would have worked only before submitting, as when the assignment is being reviewed, the specified path to the correct folder would have changed.

4.1.7 Point 7

To ensure the ANN was been trained successfully, a number of validation tests were performed. These involved taking a set of input answers directly from the original data, passing it into the input neurons, conducting a forward pass, and comparing the output with the desired encoded output that corresponded to the selected input vector.

When the previous test proved successful, further tests were conducted, this time with the addition of some ‘noise’. This was simulated by altering one or two of the answers such that the inputs formed a sequence that the network had never seen during training. This was done to prove that the ANN could still guess the correct character when provided incorrect information, by recognising the pattern rather that the specific answers.

4.1.8 Point 8

As shown in **Point 4**, the number of hidden units selected (based on a rule-of-thumb calculation) was 10. However, to discover the optimal network, further alterations had to be tried and tested. This was done in a couple of ways.

The first method was to simply use intuition, combining both pruning and growing techniques in order to localise on the number of hidden neurons that result in the best training phase and network performance possible. This meant that for every new network topology tried, the resulting learning curve was analysed (for overall learning time, convergence, and smoothness), as well as how well the trained ANN could still predict. In ideal situations, this trade of can be represented by the graphical curves in ***figure 2***.

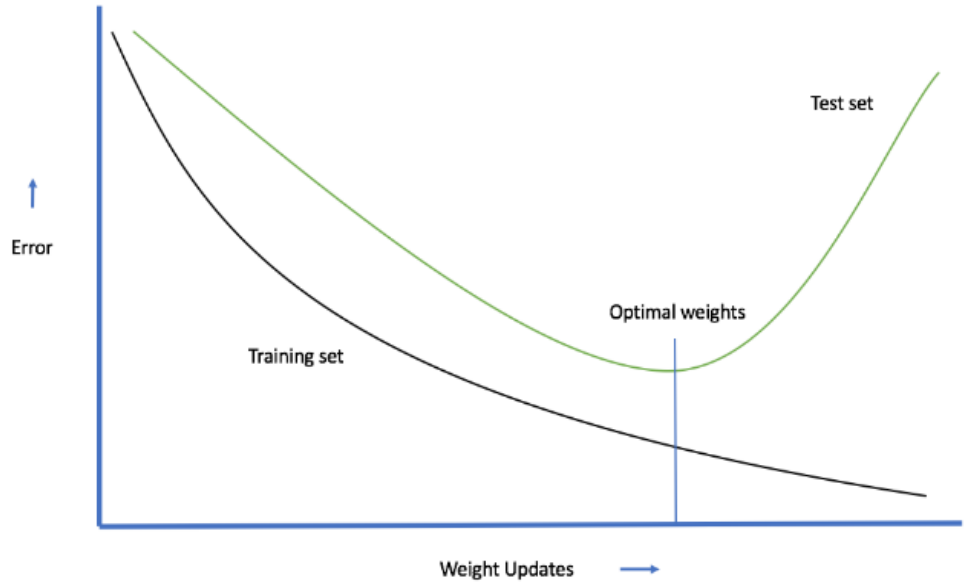


Figure 2 - Train/test data - error trade-off [5]

The second approach for finding an appropriate number of hidden units was to utilise more libraries provided by ‘Encog’. Using PruneIncremental, the program will automatically do the steps mentioned in the last (manual) approach. Although it does have some shortfalls, it does provide a fast and accurate result by testing the performance of the ANN through a specified range of hidden units. This was used to back-up the decision made based on the manual method.

4.2 Part 2

4.2.1 Point 1

An easy-to-use dialog interface was developed in order to have a simplistic experience when playing the game ‘Guess Who’ with the computer. The inputs were yes/no answers, but could be entered in a variety of ways. Loops were utilised so that if the user entered an invalid input, they would be asked to try again, as opposed to breaking the code. A game loop was also used to allow for repetitive games, if the user so wishes.

4.2.2 Point 2

The inputs from the user were stored in an array, and after all questions were asked, the ANN would be used to compute the output for the given input matrix. With this, an accurate classification (guess) could be made.
For being able to implement this step, the trained ANN was loaded from the file saved in the 'Learning2' directory.

4.2.3 Point 3

For mapping the output of the network to an appropriate guess, two techniques were used. The output values from the forward pass were in the range between 0 and 1 (because of the sigmoid/logistic function). These values could simply be rounded (**1** for greater than **0.5,** and otherwise **0**). The resulting values could be compared with the encoded names, and potentially makes a guess if it finds a match.

There is still the case in which the user provides answers that do not relate to any previously taught patterns. For this a second approach was used, as originally the game would simply say that it did not know the answer when it could not match the in-game neural outputs to a prior learnt output vector. This however was undesirable, as all it takes is one incorrect output being slightly on the wrong side of the threshold. Therefore, a safety-net was created, in that if the ANN failed to classify, it would instead simply guess the person with the greatest stimuli (highest output node).

One other product from the software, which was touched on earlier, is that the game expresses its certainty of its guess in the form of a percentage. This is because 'one-hot' encoding was used, meaning that each character has an individual node. Once activated, its sigmoidal output (between 0-1) can be used as a probability, which adds a nice feature to the game. An example of this is shown below:

	Curly hair	Blonde	Red Cheeks	Moustache	Beard	Ear rings	Female
Sum of Yes's	9	13	13	8	12	17	12
P(Yes)	0.2571	0.371	0.371	0.2285	0.342	0.4857	0.3428

4.2.4 Point 4

An addition for this point was to include a feature that allowed the game to guess early, whenever it felt confident enough. There are a few ways this can be done, but it was important to avoid simply neglecting data from the inputs, as missing inputs can cause very poor predictions, in some ways almost random.

One way is to train multiple networks (each capable to guess with a different number of features). These could be objects created from a completeANN class, each time giving the constructor a different number of inputs. The objects would come back fully trained, and could be used at each stage of the questioning. With this method, it is important to ensure that the reduced questions are ones that are still able to differentiate between the characters.

Another method would be to still use the original network, but make assumptions for any unanswered questions (after perhaps half the questions have been asked). A poor way to do this is to make all assumptions either 1 or 0. Take the table below for example, and assume

the user is thinking of Anita. If all questions are 0 as default, the game may make a very strong guess of Alex after answer 4, as the '0' assumptions make the network think that all 7 inputs are the exact ones of Alex, even though the next answers would all be 'Yes'.

Alex	1	0	1	1	0	0	0
Anita	1	0	1	1	1	1	1

A better option would be to make the default values 0.5. This gives an average, allowing the assumptions to never match perfectly with the remaining features of any character. This was implemented in the original software, and proved to work quite well.

However, the final code implemented in this assignment used slightly more accurate assumptions. It did this by using the training-data to calculate a probabilistic likelihood for every feature, and then using these more accurate probabilities for the corresponding feature input.

4.2.5 Point 5

As this particular point was very similar to the original task, it was decided to instead make an attempt to create an entirely new dataset from scratch, this time with the full 24 characters, and train the network with that. Along with this, character cards were provided so that the user could choose their answers based on visual inspection. The file of characters (from the TV series 'Game of Thrones'), and the corresponding training-set can be found in the 'Learning2' directory, and the game can be played. However, due to the more complicated nature, the ANN was not able to converge at a global minimum, and therefore gives faulty classifications.

4.3 Part 3

4.3.1 Point 1

There are many different optimisation/learning algorithms that can be used for such ML tasks, some of which working particularly well for certain types of neural networks. An example of these is Reinforcement Learning (RL), which works well when the exact input/output pairs may not be known, but the weighting decisions can still be evaluated as wither 'good' or 'bad'.

The reason backpropagation is so popular, is that it works very well with feed-forward Multi-Layer Perceptrons (MLPs), and the process can be understood mathematically with calculus. Along with this, there are also many libraries that implement backpropagation for the programmer. It is for all these reasons that it was used for this assignment.

One other important technique of note is Evolutionary/Genetic Algorithms (GAs). These can also be implemented easily in 'Encog', but for this particular classification task, would not provide any additional benefit. However, if the algorithm was being coded from scratch, other parameters (momentum/learning-rate/convergence-error) can be inserted into the chromosomes along with the weights, and over a number of generations, will evolve to an optimal solution. They also work very well when dealing with non-simplistic topologies and data-flow, as they do not require any 'backward-pass' to generate a solution.

4.3.2 Point 2

There was no time to implement the addition of a maybe answer into the game, however in theory it would work very well. This could be done altering the dataset to include maybes, and then altering how you numerically represent the answers. The top two methods for this would be to code the inputs as either:

Yes = 1, Maybe = 0.5, No = 0

Or:

Yes = 1, Maybe = 0, No = -1

Here the second method still works as even without using the hyperbolic tangent activation. This is because the inputs do not have activation functions, so the negative just denotes it as opposite to 'Yes'.

4.0 Testing Approaches and Examples

Testing the software is extremely important in finding run-time bugs. If the program appears to run correctly, these may be hard to detect, and therefore it is imperative to cover all possible scenarios.

The testing for this assignment included:

- Ensure program accepts dataset file.
- Check dataset has been stored/indexed correctly.
- ANN learns/converges (print error).
- Verification and noise tests are successful.

An example of this last one was to test Bernard by inserting the following answers:

Curly hair	Blonde	Red Cheeks	Moustache	Beard	Ear rings	Female
No	No	Yes	No	No	Yes	No

The output was then compared to the target encoding for Bernard's name:

Target Output	0	0	0	0	1
Real Output	0.0039096	0.1105735	0.0246655	0.0200836	0.93336

This shows how similar the response is. Noise can then be simulated by changing an answer, and the network still proves to guess the right person.

The game/user interaction was also tested meticulously. This was done by running the program many times, and:

- Changing answer syntax.
- Giving it inaccurate answers.
- Checking there are no unexpected breaks in the code.
- Consecutive games can be played.
- The question will loop if user gives invalid response.
- The network is imported correctly and works with game.
- The game makes correct guesses (even with wrong answers).
- The game can 'predict' and guess early (and is still correct).
- The user can actively refuse the game to guess early, and game resumes as normal.

5.0 Evaluation

When using the pruning (or rather growing) tool, it was specified to test between 1 and 15 hidden neurons. The results were as follows (***figure 3***):

1/15:	Current:	H1=5;	Best:	H1=5
2/15:	Current:	H1=1;	Best:	H1=1
3/15:	Current:	H1=4;	Best:	H1=1
4/15:	Current:	H1=8;	Best:	H1=1
5/15:	Current:	H1=3;	Best:	H1=1
6/15:	Current:	H1=7;	Best:	H1=1
7/15:	Current:	H1=6;	Best:	H1=1
8/15:	Current:	H1=10;	Best:	H1=1
9/15:	Current:	H1=11;	Best:	H1=1
10/15:	Current:	H1=12;	Best:	H1=1
11/15:	Current:	H1=15;	Best:	H1=3
12/15:	Current:	H1=14;	Best:	H1=3
13/15:	Current:	H1=13;	Best:	H1=3
14/15:	Current:	H1=9;	Best:	H1=3
15/15:	Current:	H1=2;	Best:	H1=3
Neural Network created (best): **07 - 03 - 05**				

Figure 3 - Automated pruning

From this, the 15 steps can be seen. The final result, showing the best network containing 3 hidden neurons, fully confirms what was found from the manual approach conducted prior to this automated test.

When altering the network parameters, the resulting learning curve can be observed in ***figures 4***, ***5*** and ***6***.

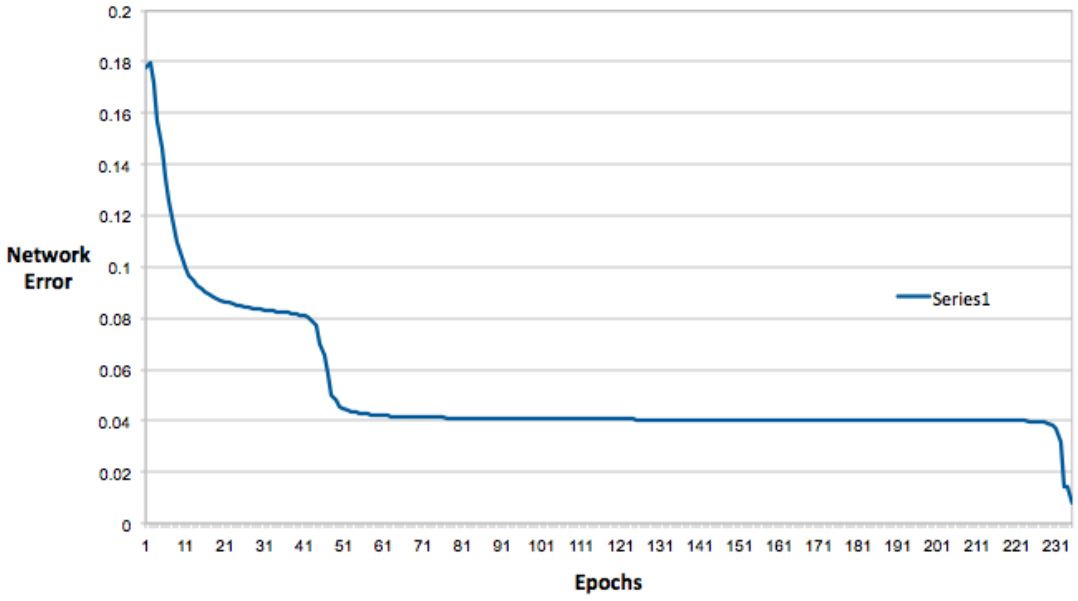


Figure 4 - LEARNING RATE = 0.8, MOMENTUM = 0.5

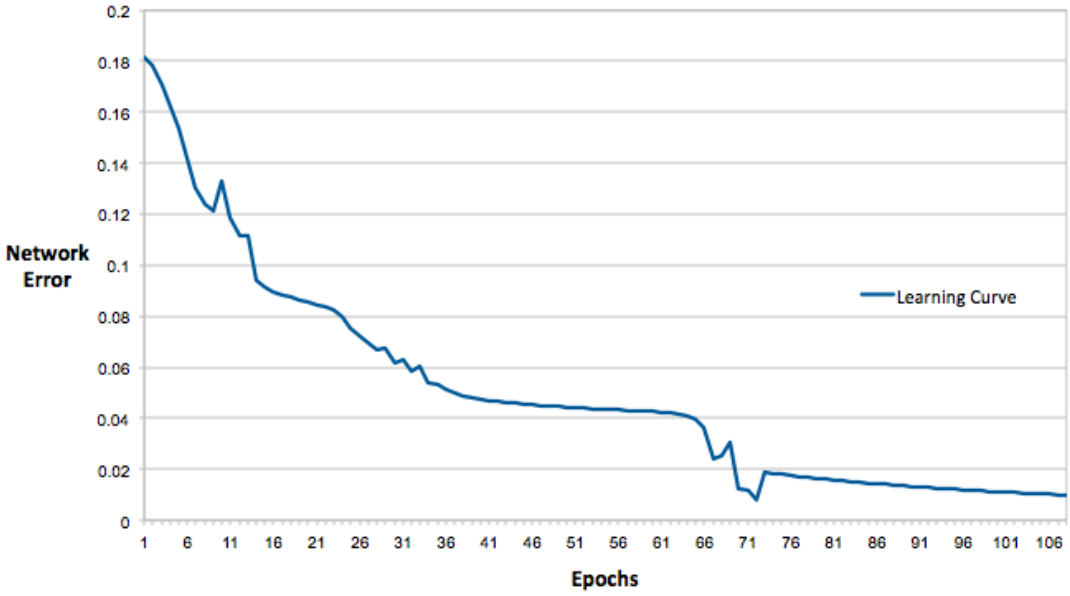


Figure 5 - LEARNING RATE = 0.9, MOMENTUM = 0.15

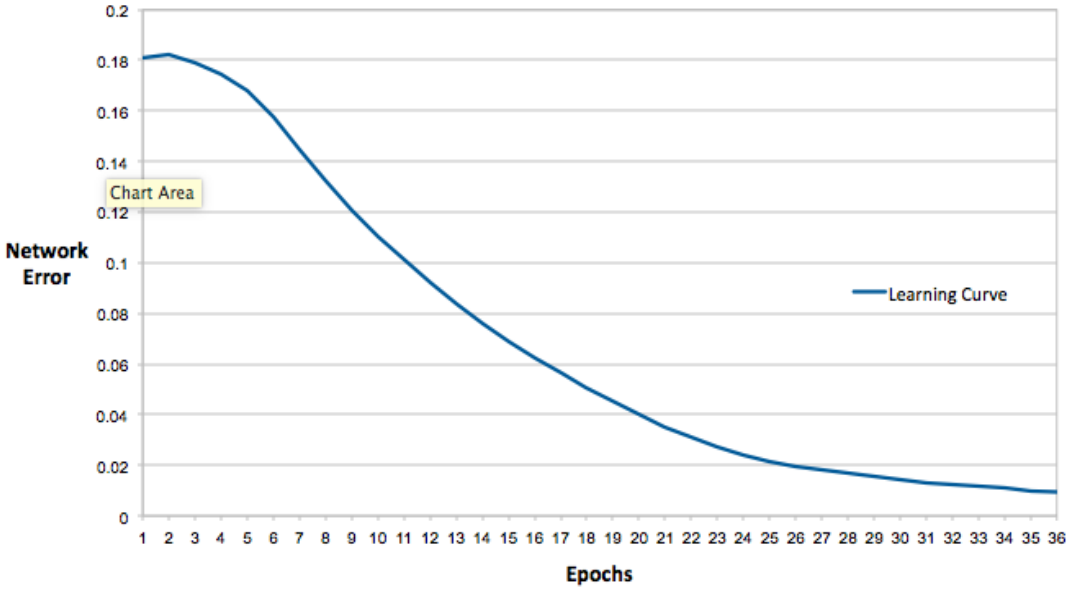


Figure 6 - LEARNING RATE = 0.6, MOMENTUM = 0.2

What these graphs demonstrate is that although a high learning rate causes a steep drop at the start, it often causes an unsettled curve. This is undesirable, with the ideal curve being smooth and short. This is particularly important when continued learning is needed, and an oscillatory learning curve would present many problems.

It would also appear that when the momentum is too high, it has a detrimental effect on the learning time.

Many more tests were conducted and analyses, but far too many to present, hence the subset of examples shown.

6.0 Appendix A

The following list contains all the parts that were implemented in the assignment, including the completed enhancements.

2.1 Part 1

- The training set was read in from a file.
- The input 'word' answers were converted into a numerical representation.
- The output 'name' classes were encoded in a numerical form. This finalised a usable dataset of input/output pairs.
- A multilayer feed-forward ANN was created, containing 1 hidden layer and sigmoidal activation functions.
- The size of the hidden layer was decided, as well as the learning rate, momentum, and use of biases.
- The network was trained using the Backpropagation learning algorithm.
- Verification tests and 'noise' tests were performed for confirmation.
- In order to find the optimal ANN, various network settings were altered whilst observing the effect on the ANNs learning, and its accuracy in character classification.
- The final trained network was saved for use in **Part 2**.

2.2 Part 2

- A logical interface was created to provide clear interaction between the user and the neural network by means of dialog (text input/output).
- The trained network from **Part 1** was loaded into the game.
- The input features were mapped to the output names for classifications to be made.
- The feature of 'early guesses' was added. If the ANN ever thought it was sure of an answer, it would ask if it could attempt an early guess.
- Add an entirely new large dataset (attempted)

2.3 Part 3

- Training techniques discussion.
- Yes/no/maybe answers discussion.

7.0 Reference List

[1] *Dale R, Somers HL, and Moisl H. (Eds.) Handbook of Natural Language Processing. Marcel Dekker, Inc., New York, NY, USA; 2000. p.755 & 823*

[2] *Morgan DP, Scofield CL. Neural Networks and Speech Processing. In: Neural Networks and Speech Processing. The Springer International Series in Engineering and Computer Science (VLSI, Computer Architecture and Digital Signal Processing), vol 130. Springer, Boston, MA; 1991. p.329*

[3] *Dybowski R, Gant V, editors. Clinical applications of artificial neural networks. Cambridge University Press; 2001.*

[4] *Khan J, Wei JS, Ringner M, Saal LH, Ladanyi M, Westermann F, Berthold F, Schwab M, Antonescu CR, Peterson C, Meltzer PS. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. Nature medicine. 2001 June; 7(6): p673.*

[5] *Kelsey T. AI Principles: Neural Nets – Lecture 03. CS5010. 2017. p8*