Frederick Hendrik Snyman
13028741

COS314: Project 3

Dataset

Afrikaans
The Afrikaans dataset contains 50 chunks of text, averaging to about 496 characters per chunk.
English
The English dataset also contains 50 chunks of text, averaging to about 344 characters per chunk.

These averages stated above include spaces, as well as special characters, that are contained within the chunk of text.

Frequencies

The frequencies of each character were calculated by iterating through the chunk of text. The chunk of text is firstly converted to lowercase. For each character, the ASCII value of that character is used to determine whether the character is eligible to be counted, in order to ignore all whitespace and other special characters. Because all characters are read as lowercase, 97 (the ASCII value of 'a') is subtracted from the character's ASCII value, and then the character's integer representation in an array is increased.

```
private int[] determineFrequencies(String string){
    int[] returnThis = new int[26];
    string = string.toLowerCase();
    for (int i = 0; i < string.length(); ++i){
        char c = string.charAt(i);
        if ((c > 96 && c < 123)){
            returnThis[c-97]++;
        }
    }
    return returnThis;
}
```

Figure 1: Code to determine character frequency

Special Characters

Special were deemed to be unimportant, as they do not greatly influence the decision to be made whether a chunk of text is English or Afrikaans, seeing as both languages borrow words which contain special characters from other languages. Any special characters found within a chunk of text were thus ignored.

Format of Dataset

Each language is stored in a separate file, in the following format

```
chunk of text\n
char of language
```

where `chunk of text` is the chunk of text of the language, and `char of language` is either 'a' or 'e' for Afrikaans or English respectively.
This format allows both languages to be present in a single file, but for separation of concerns, and convenience, the two languages were placed in separate files.

Pre-processing

The data were pre-processed by manipulating the frequencies of the characters according to the following formula:

$$t_s = \frac{t_u - t_{u,min}}{t_{u,max} - t_{u,min}}(t_{s,max} - t_{s,min}) + t_{s,min}$$

where:

- $t_s \Rightarrow$ *scaled target*
- $t_u \Rightarrow$ *unscaled value*
- $t_{u,min} \Rightarrow$ *minimum of unscaled values*
- $t_{u,max} \Rightarrow$ *maximum of unscaled values*
- $t_{s,min} \Rightarrow$ *scaled minimum*
- $t_{s,max} \Rightarrow$ *scaled maximum*

Because the Sigmoid function is used as the activation function for the neurons in the Neural Network, the frequencies were scaled to be within the range $[-\sqrt{3}, \sqrt{3}]$, seeing as the Sigmoid function undergoes the most change in gradient between these x-values.

```
private Double[] scale(int[] inputs){
    Double[] scaledInputs = new Double[inputs.length];

    Double targetMax = Math.sqrt(3);
    Double targetMin = -targetMax;
    int min = inputs[0];
    int max = inputs[0];

    for (int i = 1; i < inputs.length; ++i){
        if (inputs[i] < min){
            min = inputs[i];
        } else if (inputs[i] > max){
            max = inputs[i];
        }
    }

    for (int i = 0; i < inputs.length; ++i){
        scaledInputs[i] = ((((double)inputs[i] - (double)min)/((double)max - (double)min))*(targetMax - targetMin))+(targetMin);
    }

    return scaledInputs;
}
```

Figure 2: Function to scale frequencies

These scaled inputs were then used as the pre-processed data for input to the Neural Network.

Experimentation

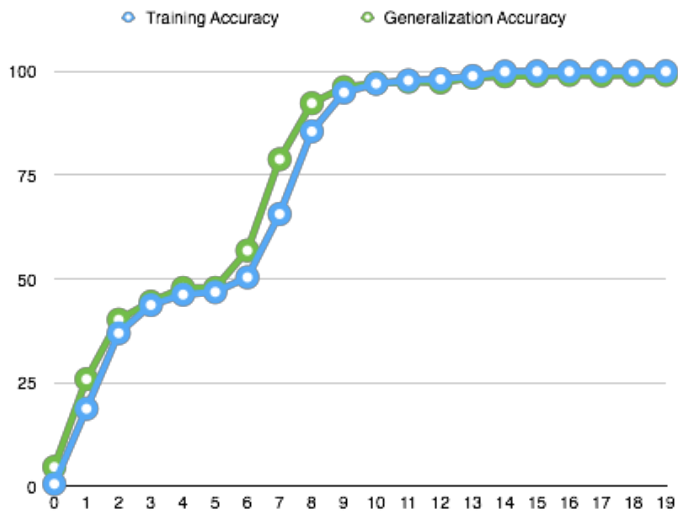
NB For all graphs: x-axis is the number of epochs; y-axis is the percentage

Hidden units

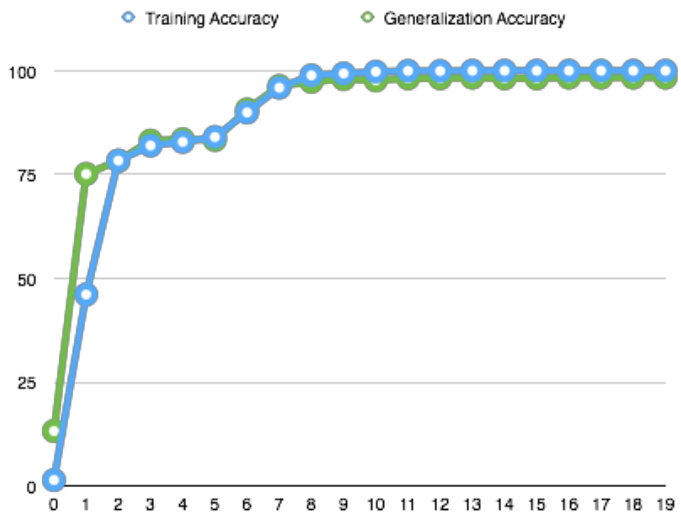
To determine the optimal amount of hidden units, the Neural Network was ran for 30 times using the following parameters:

Learning rate: 0.7
Momentum: 0.1
Max number of epochs: 20
Training set size: 0.8

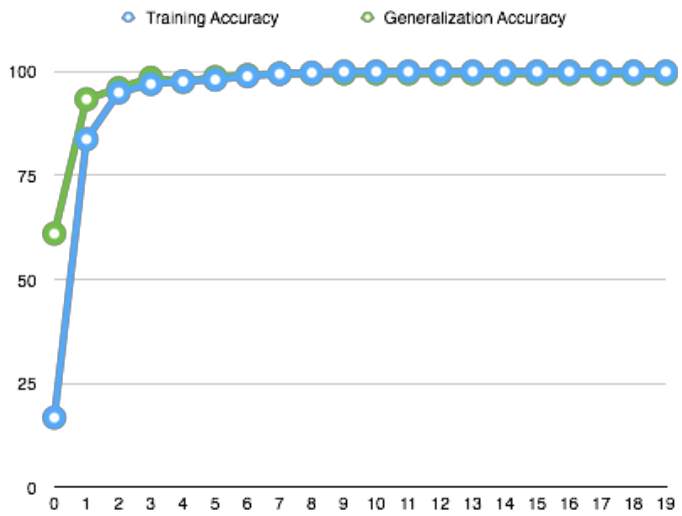
and varying hidden units. The following results were produced:



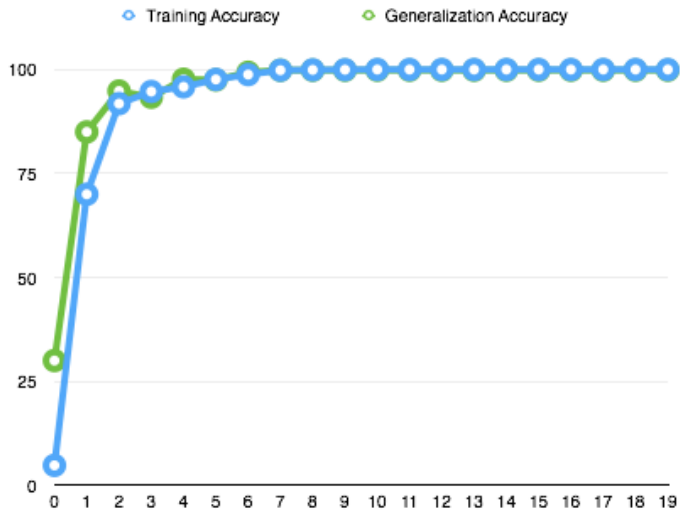
Hidden nodes = 1



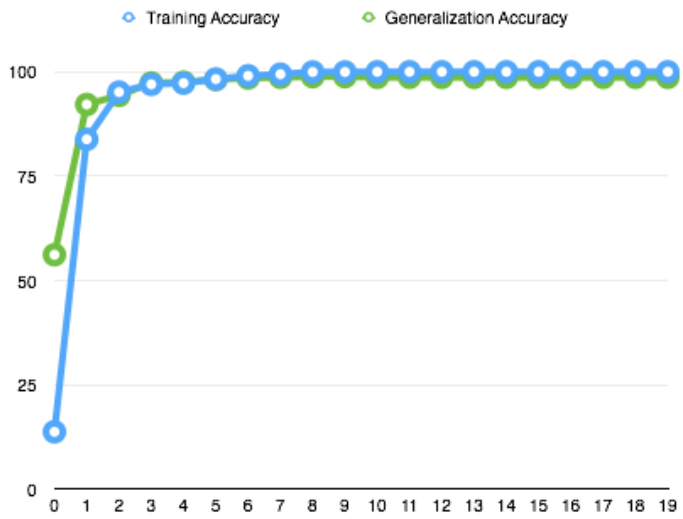
Hidden nodes = 2



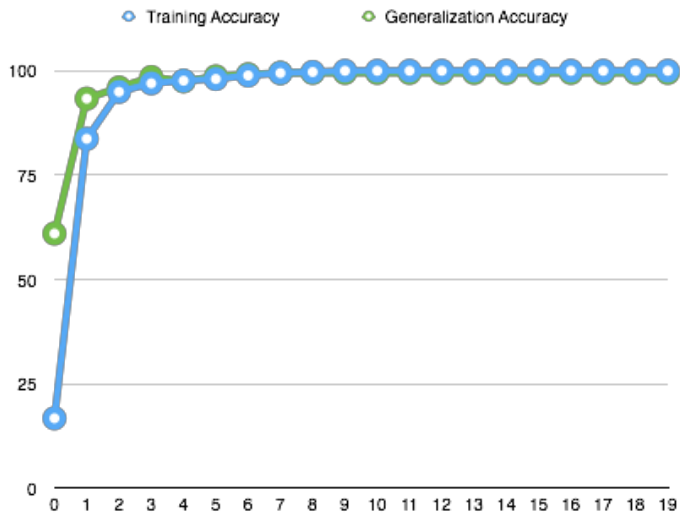
Hidden nodes = 3



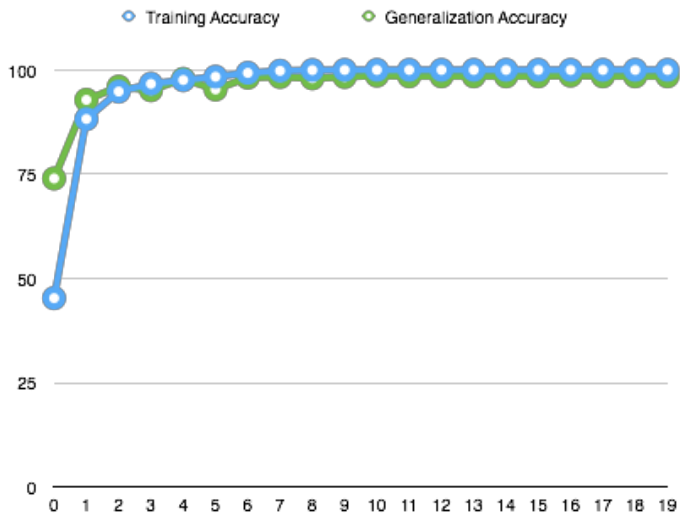
Hidden nodes = 5



Hidden nodes = 10



Hidden nodes = 15



Hidden nodes = 100

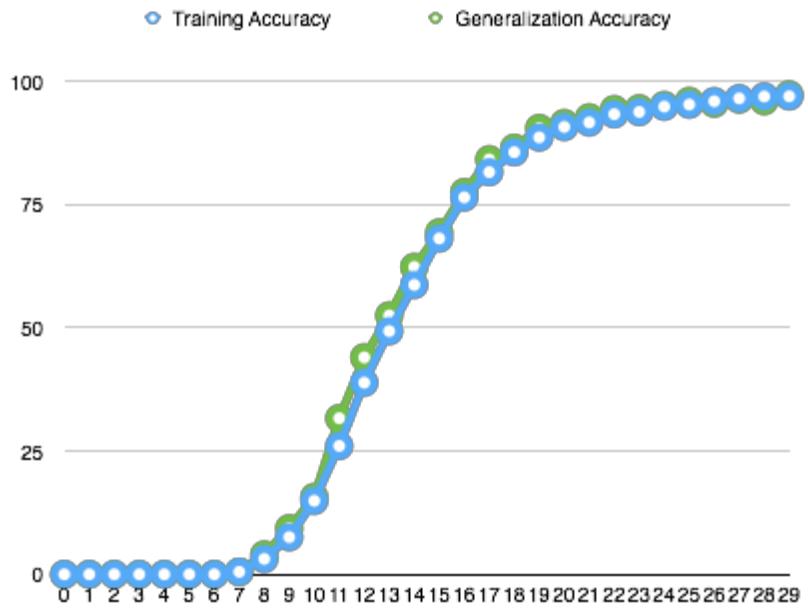
From the above results, it is clear that the more hidden nodes there are, the quicker the neural network trains on the training set.
The higher number of hidden nodes also increases the rate at which an accurate generalization is achieved.
A further observation to be made is with as the number of hidden nodes increases, the accuracy on the first epoch also increases, for both the training accuracy and generalization accuracy.
Overfitting tends to occur as the number of hidden nodes increases.
Thus, we can deduce that there seems to exist a certain sweet spot for the amount of hidden nodes, which from the results seem to be when `hidden nodes = 10`.

Learning rate

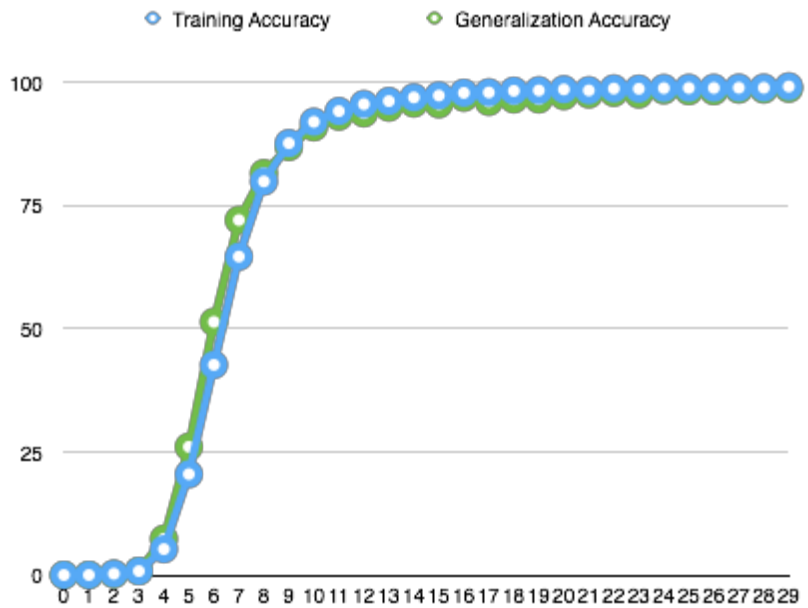
To determine the optimal learning rate, the Neural Network was ran for 30 times using the following parameters:

Hidden units: 5
Momentum: 0.1
Max number of epochs: 30
Training set size: 0.8

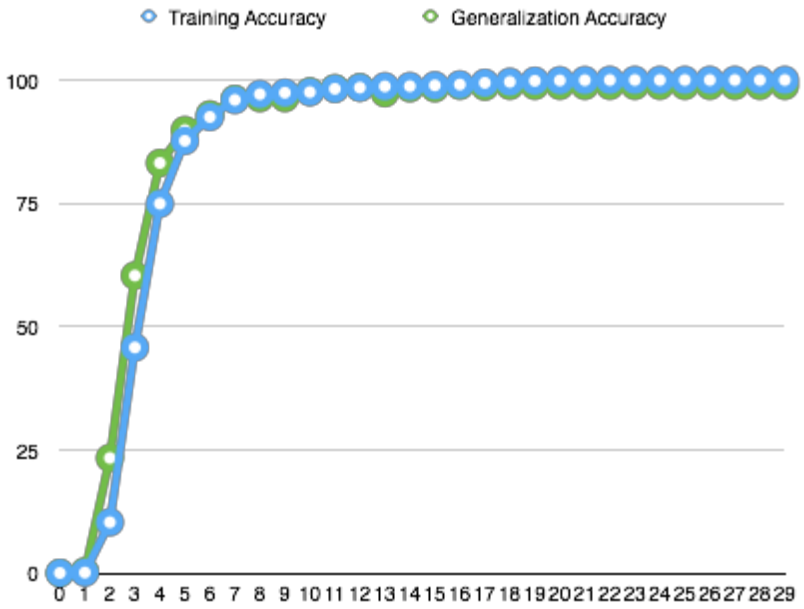
and a varying learning rate. The following results were produced:



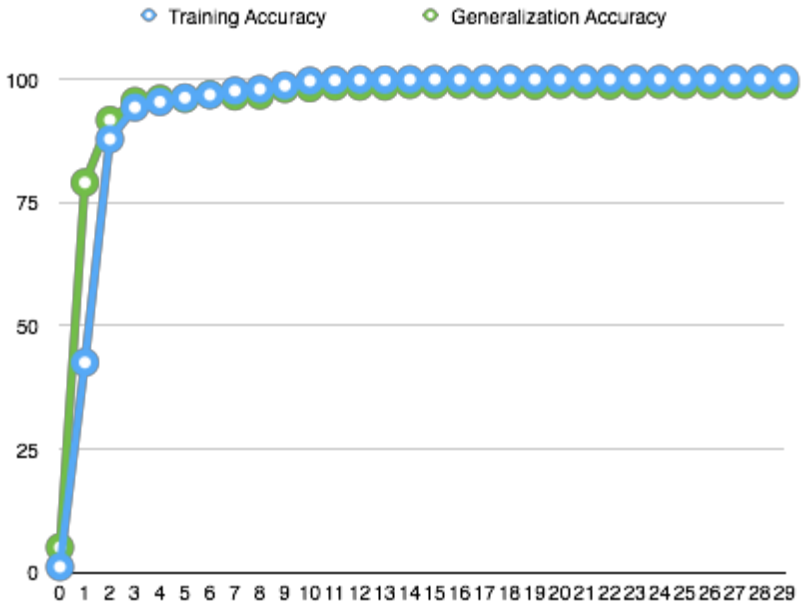
Learning rate = 0.05



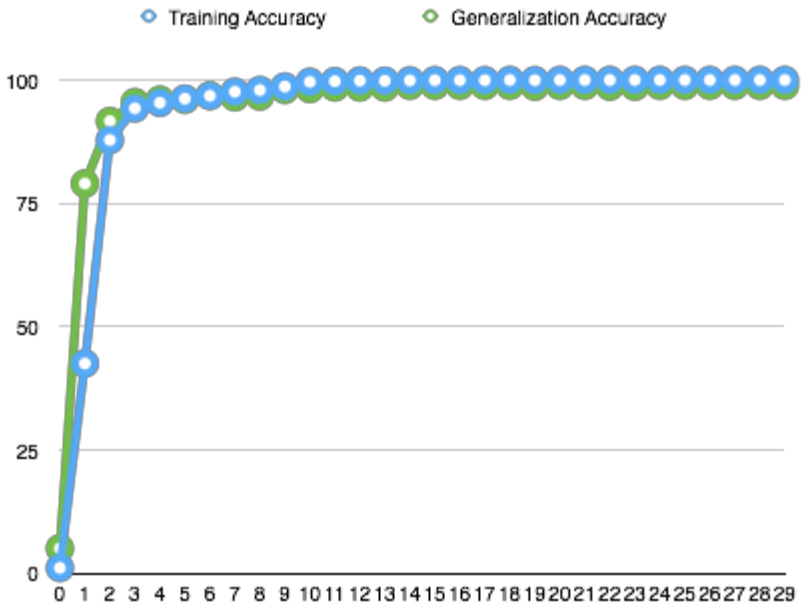
Learning rate = 0.1



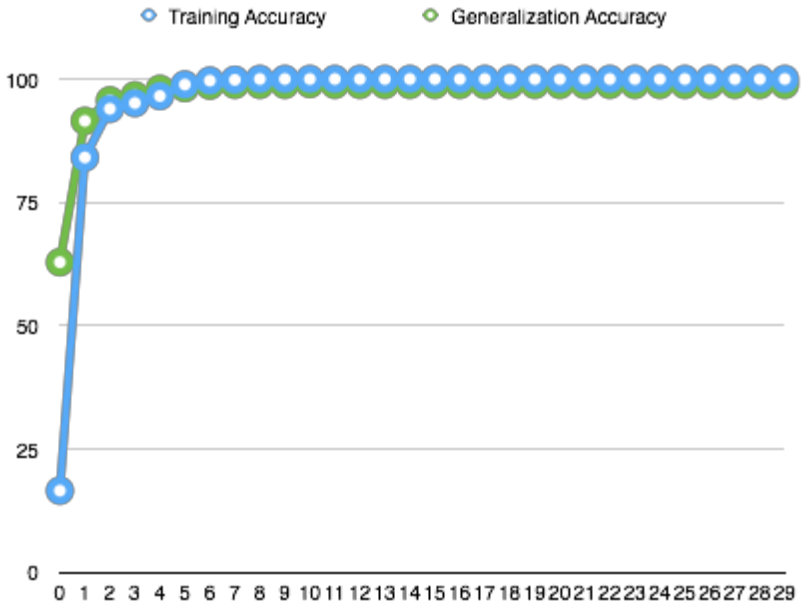
Learning rate = 0.2



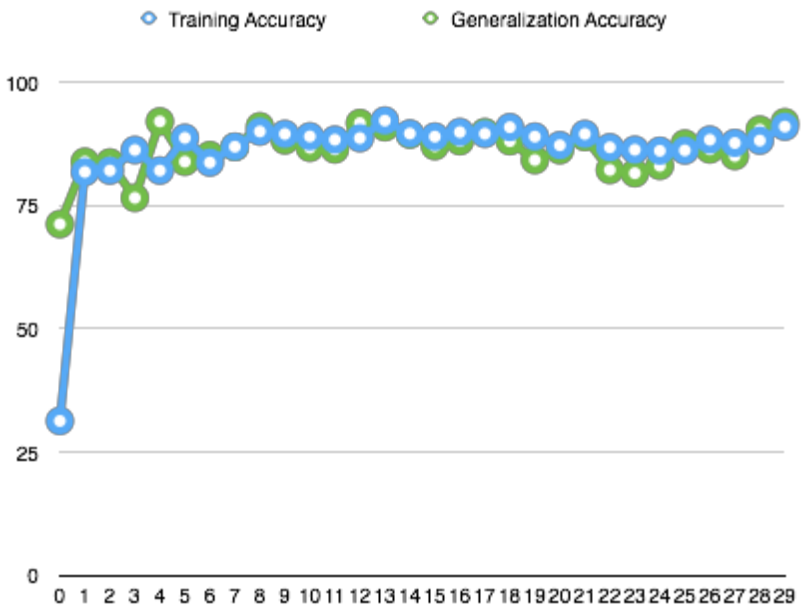
Learning rate = 0.5



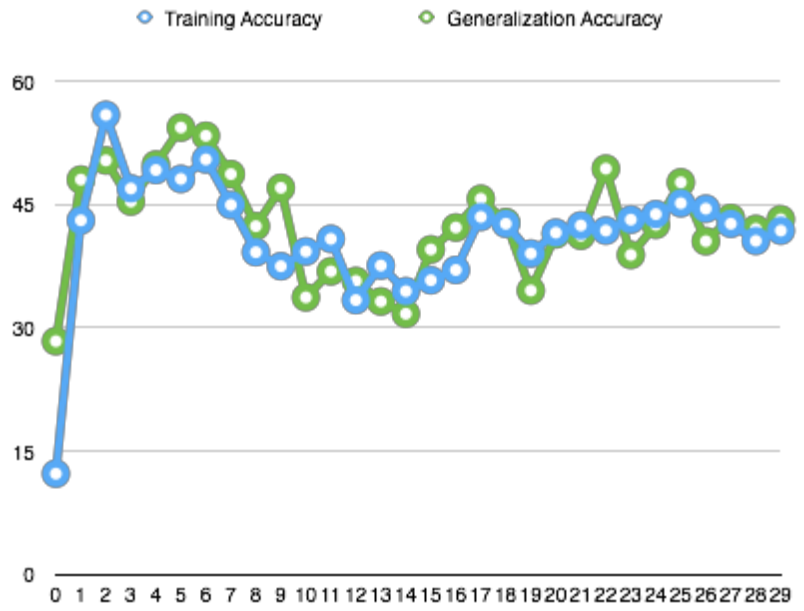
Learning rate = 0.7



Learning rate = 1.0



Learning rate = 2.5



Learning rate = 5.0

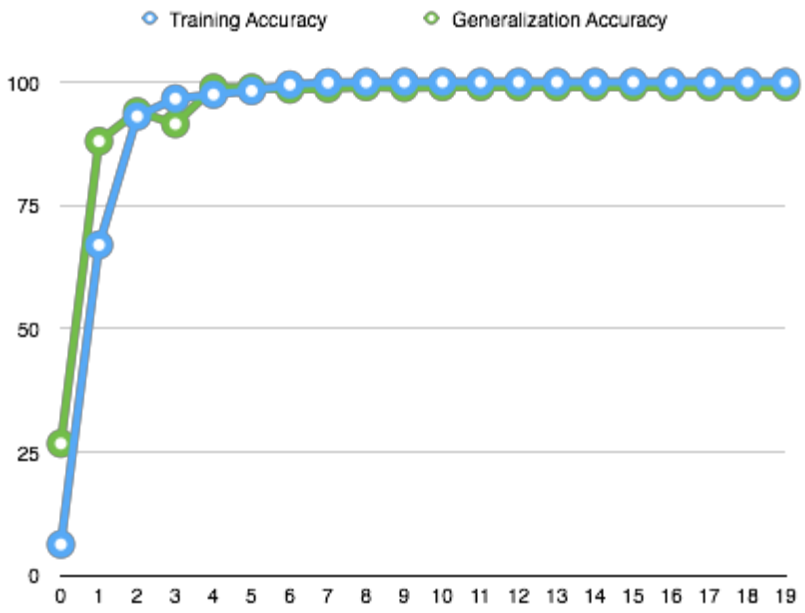
From the above results, it is clear that the learning rate has a significant influence on the rate at which the training accuracy and generalization accuracy increases.
A higher learning rate results in quicker stabilization of the accuracies at a high accuracy.
As the learning rate gets larger, the training and generalization accuracies tend to be more unstable.
The learning rate does not appear to have an influence on the training accuracy of the initial epoch.
There thus also seems to be a sweet spot for the learning rate, and from the above results, this sweet spot appears to be when `Learning rate = 1.0`

Momentum

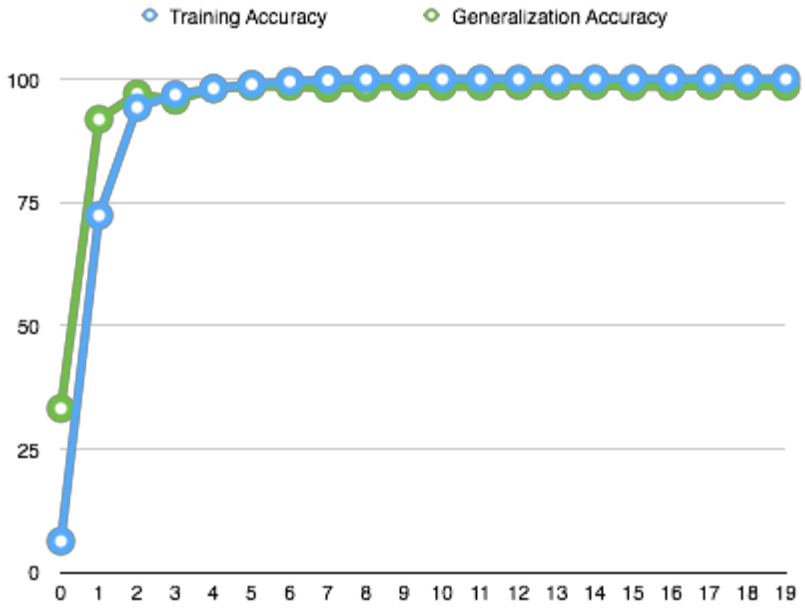
To determine the optimal learning rate, the Neural Network was ran for 30 times using the following parameters:

Hidden units: 5
Learning rate: 0.7
Max number of epochs: 20
Training set size: 0.8

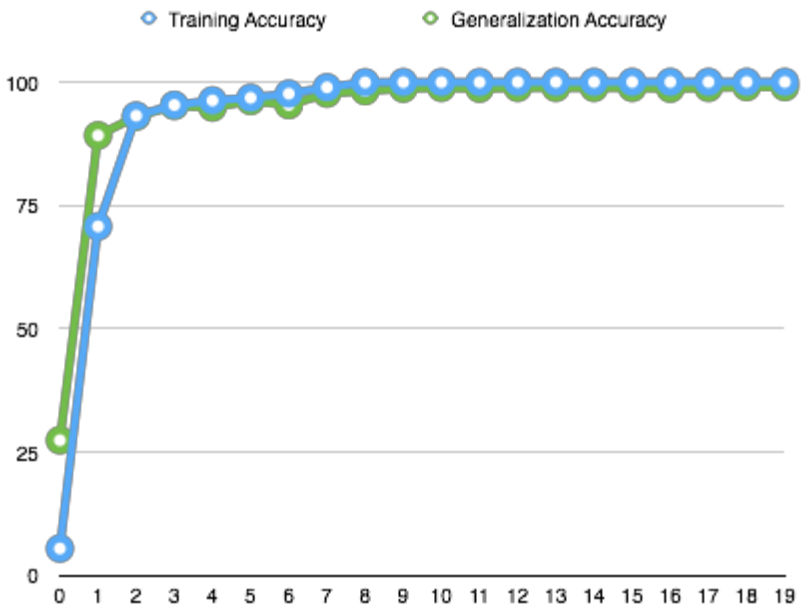
and a varying momentum. The following results were produced:



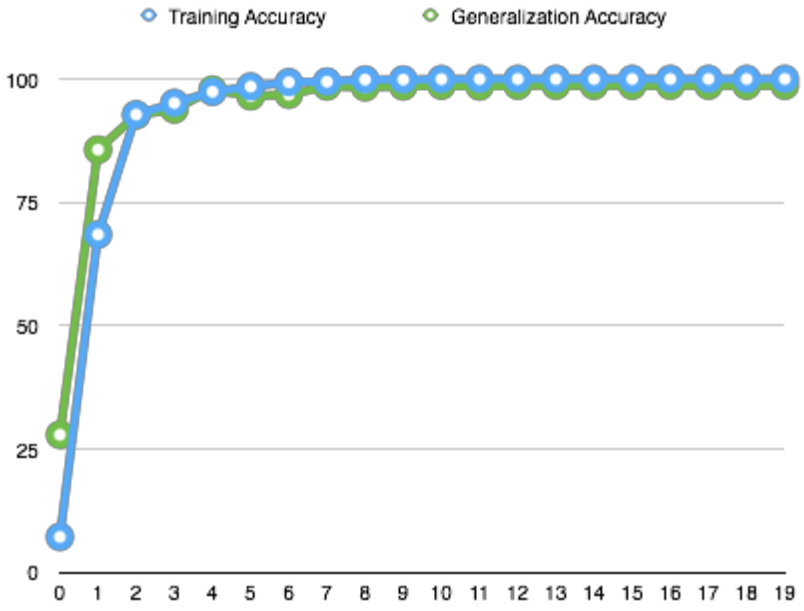
Momentum = 0.0



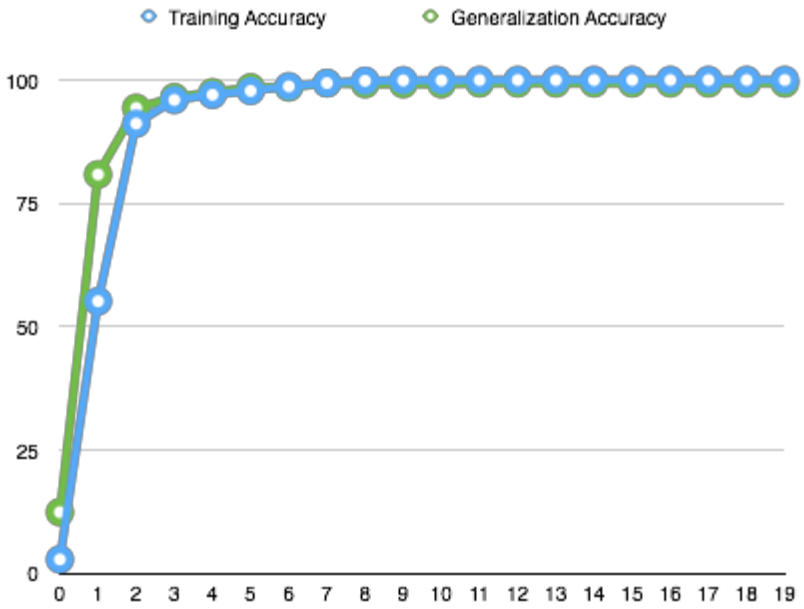
Momentum = 0.01



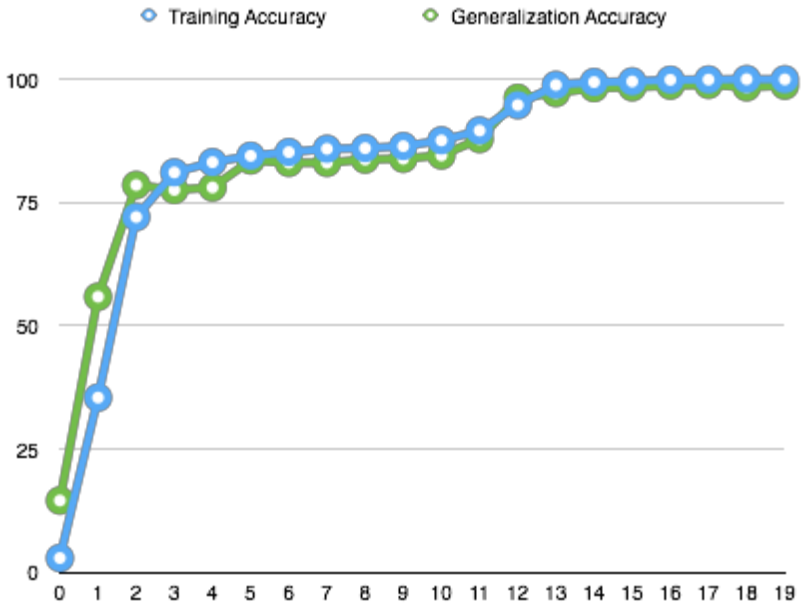
Momentum = 0.05



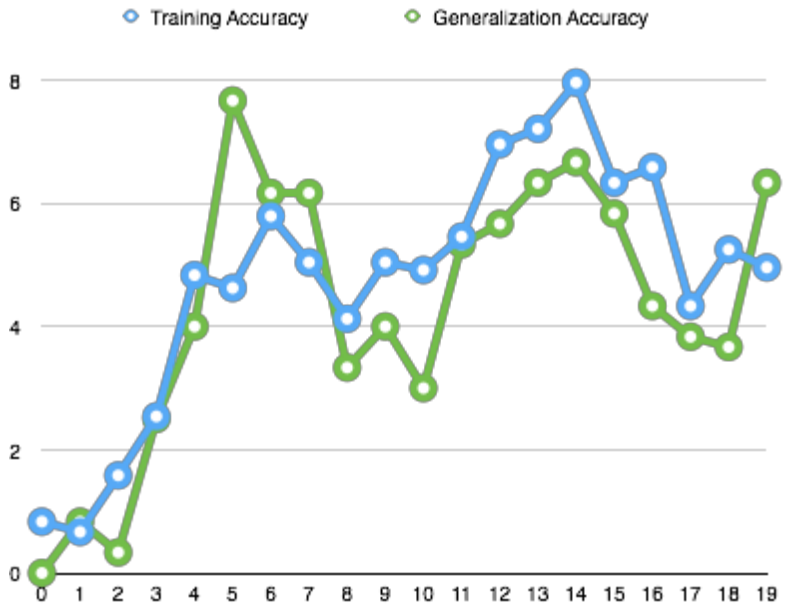
Momentum = 0.1



Momentum = 0.5



Momentum = 0.75


Training Accuracy
Generalization Accuracy
8
6
4
2
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Momentum = 1.0

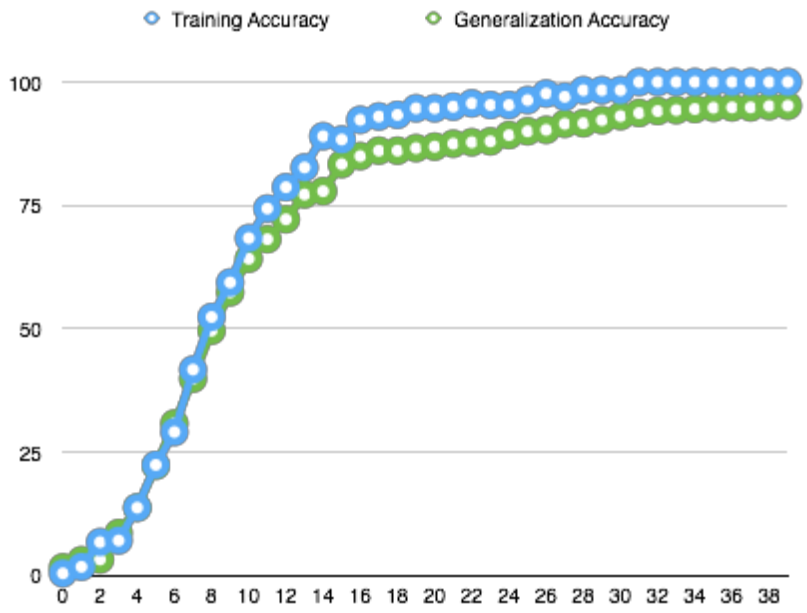
From the above results, we can see that the momentum has a small impact on the rate at which the training and generalization accuracies converge.
However, as the momentum gets too big, this convergence is more unstable. This is because a local minimum cannot be achieved due to the fact that the previous weight has too big of an influence on the next weight being calculated.
The optimal momentum seems to reside at 0.01, because the rate at which the training and generalization accuracies converge is the highest at this momentum. These accuracies are also stable as the number of epochs increase.

Training set percentage

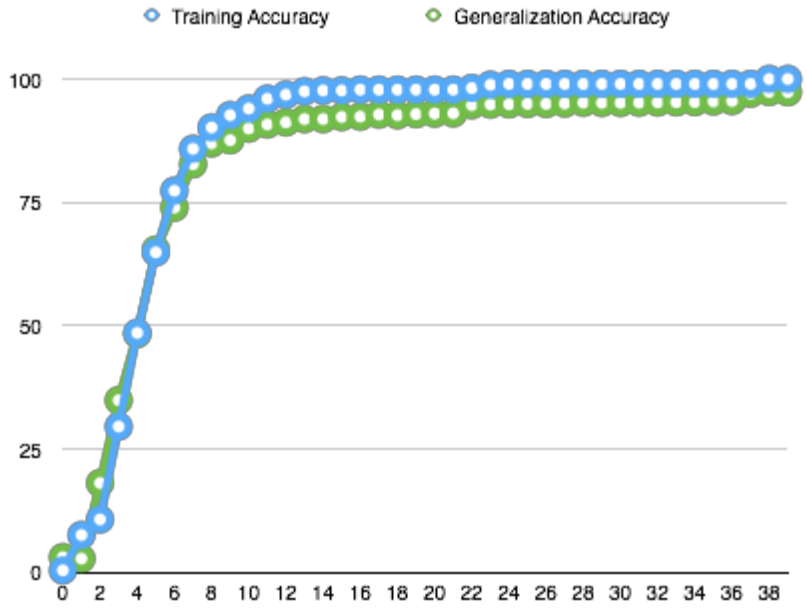
To determine the optimal training set percentage, the Neural Network was ran for 30 times using the following parameters:

Hidden units: 5
Learning rate: 0.7
Max number of epochs: 40
Momentum: 0.1

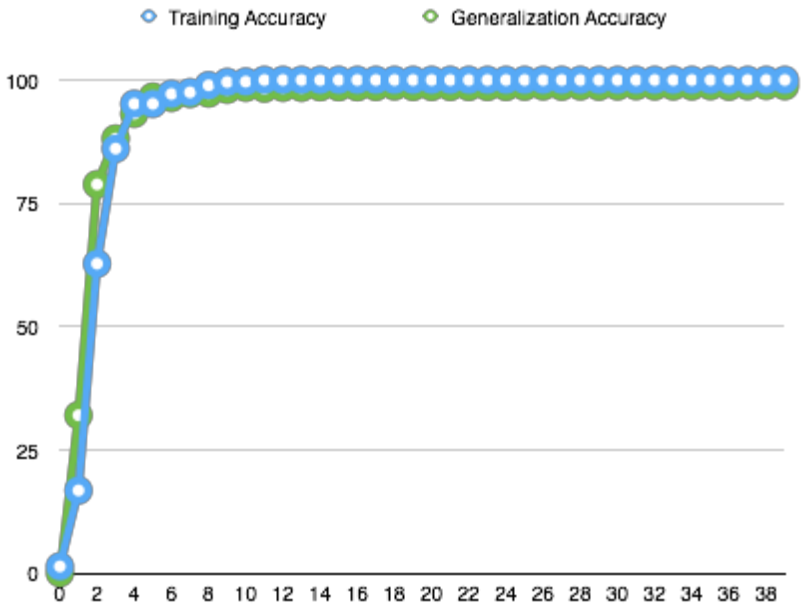
and a training set percentage. The following results were produced:



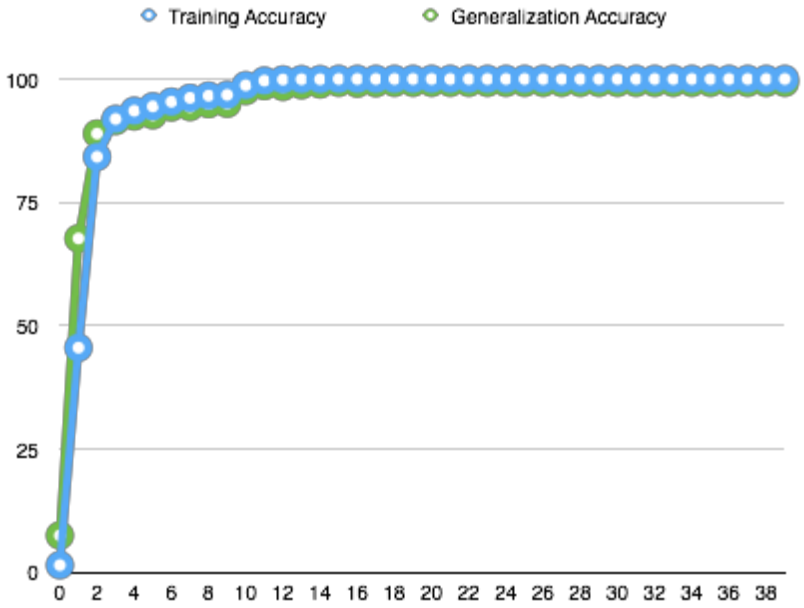
Training set size = 0.1



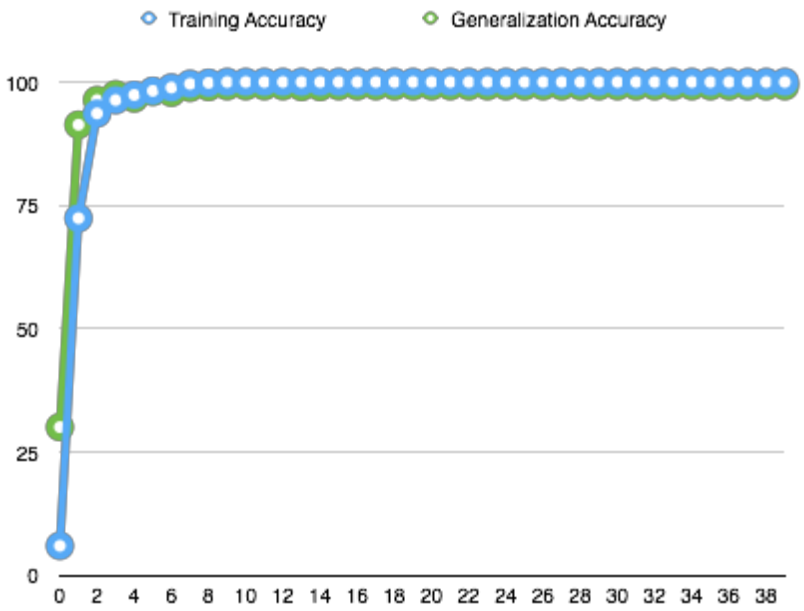
Training set size = 0.2



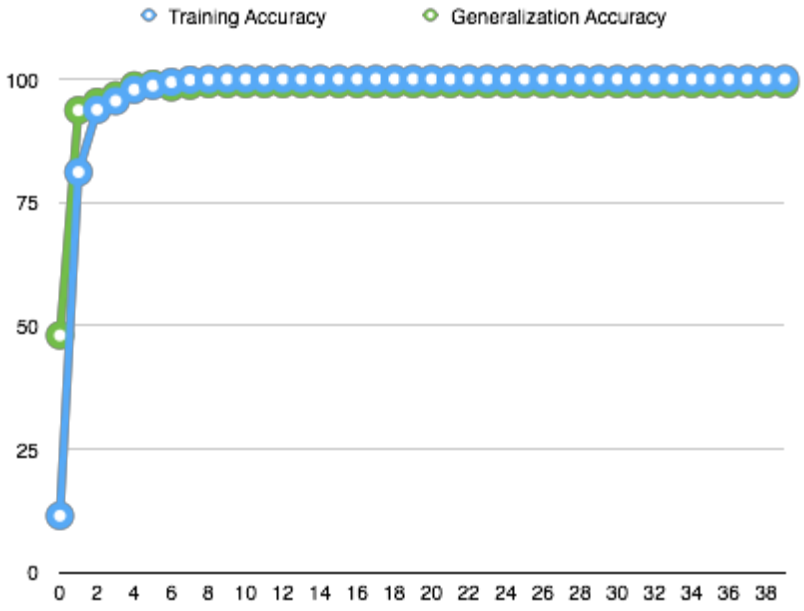
Training set size = 0.4



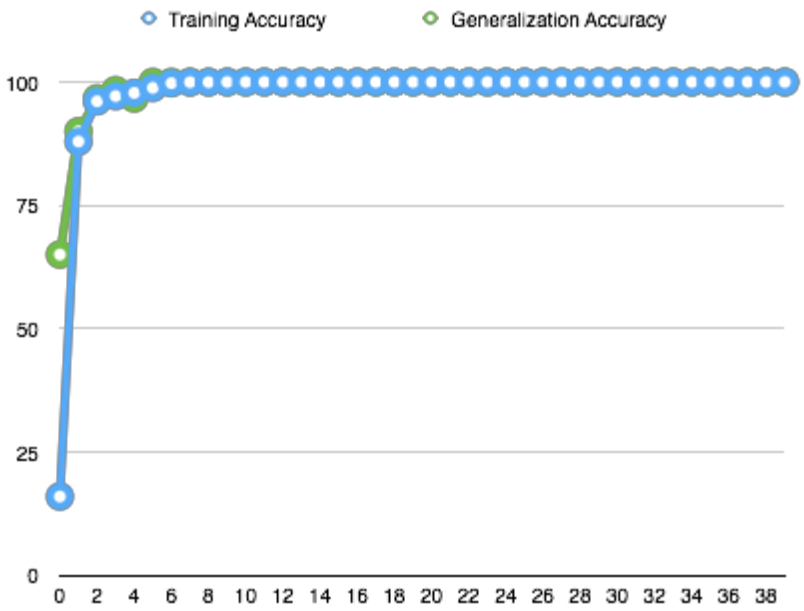
Training set size = 0.6



Training set size = 0.8



Training set size = 0.9



Training set size = 0.99

From the above results, it is clear that as the size of the training set increases, the rate at which the training accuracy and the generalization accuracy converges increases.
The convergence tend to be more stable as the size of the training set increases.
It is to be noted, however, that the generalization accuracy is less reliable as the training set size increases, seeing as the sample size of the generalization set is too small to produce reliable results to judge the generalization of the Neural Network.
With this in mind, we can deduce that the optimal convergence occur when `Training set size = 0.8`

Conclusion

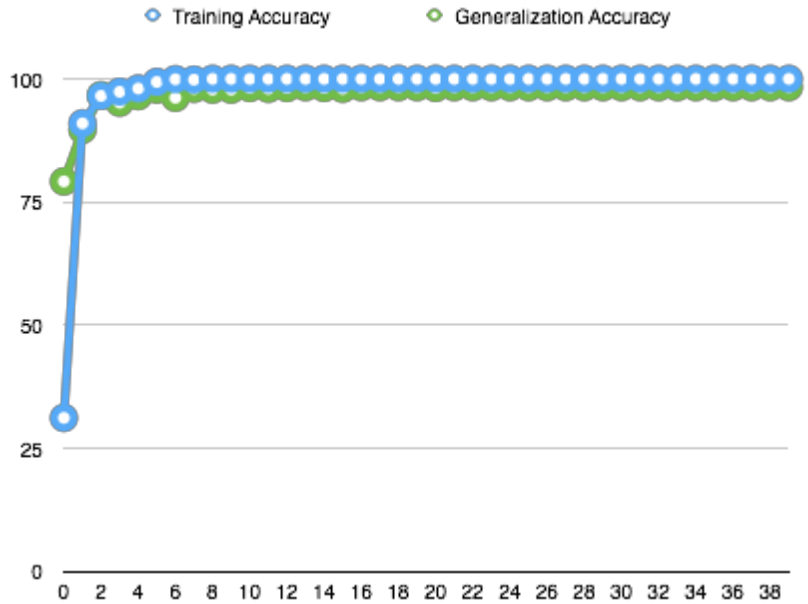
From the above results, we can conclude that the most optimal:

Hidden Node = 10
Learning rate = 1.0
Momentum = 0.01
Training set = 0.8

The least optimal:

Hidden Node = 1
Learning rate = 0.05
Momentum = 1.0
Training set = 0.1

Graph produced by most optimal:



Graph produced by least optimal: