Neural Network Applications for Connect Four

ECE457B Final Project

April 8, 2013

Thomas Blight, 20316512
Weiliang Chen, 20322350
Gabriella Grandilli, 20294003
Noah Sugarman, 20295111

Abstract

The development of artificial intelligence for games often requires a large amount of time and effort. We investigated the development of an AI based on artificial neural networks that trains only with game states and their expected results. This type of AI would be capable of playing complex games without the difficulty of programming in prior knowledge of the game and its rules. We compared two networks: one that trained, with game states we generated ourselves and another that trained on a public database of possible game states after the 8th move.

The resulting AI performed at a beginner level of skill, beating an intermediate player 27% of the time and a random AI 87% of the time. Deciding which move to take took 0.15 seconds, no matter where in the game the decision was being made. Training on the 8-ply data resulted in the best AI. This approach was found to be scalable to larger input sizes, meaning it has the potential to be used for more complex games.

It was concluded that using this approach to develop an AI was viable when a low level of skill was acceptable. The approach excels over other approaches when a game has complex, difficult to code rules and when fast decisions are preferred.

Table of Contents

1 Introduction

1.1 Problem

The development of artificial intelligence for games often requires deep knowledge of the game, which means a large amount of time and effort. What if a game AI could instead be developed by just recording the game's state and the eventual result of the game? We set out to develop a competent Connect Four artificial intelligence player that learns only from game states and game outcomes to investigate the viability of this technique and the performance of the artificial intelligence that it produced.

We chose to implement it for Connect Four because it's a simple game to demonstrate, has a reasonable and finite number of game states, and is a solved game. This allowed us to easily verify how well our implementation performed. With Connect Four as a base-case, we discussed the use of neural networks towards other games and problems that can be modeled as a series of states.

1.2 Connect Four Gameplay

The typical game of Connect Four involves a game board of 7 columns by 6 rows, for a total of 42 positions. Each position can hold one game piece. A player starts off with 21 checker pieces, of opposite color to their opponent. To goal of the game is to place four of your pieces consecutively, either vertically, horizontally, or diagonally, such that they "connect". The players alternate turns in placing their pieces. A piece is inserted at the top of a column and falls to the bottom-most empty position.

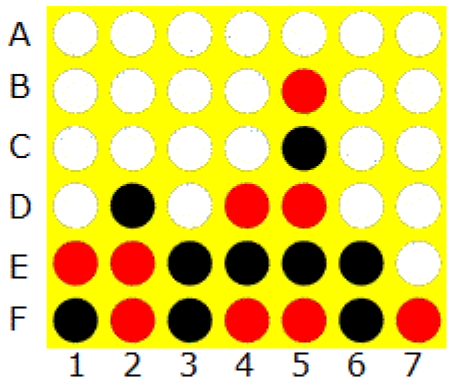

Figure 1: Connect Four Board

The game mechanics are simple to pick up but there is more strategy to the game than simply spotting three pieces in-a-row and trying to add on a fourth. Some aspects of more advanced strategy are:

- Building simultaneous threats by creating multiple rows of two or three.
- Pre-emptively blocking your opponent's potential wins.
- Forcing an opponent to play in a position they do not want to play in via "Control of Zugzwang" [4].
- Forcing an opponent to allow you to play in a position they want to play in ("Baseinverse") [4].
- Claiming even squares when controlling Zugzwang ("Claimeven") [4].

1.3 Motivation for neural network-based solution

We chose to implement an artificial neural network solution because was the best fit for the problem in a number of ways.

Since we want to be able to take the current game state as input, we need to take a minimum of 42 inputs. This is a slightly larger number of inputs that might be ideal for a neural network, but we expect the neural network to be able to handle this number of inputs with a reasonable training time. In addition, it is easy for the neural network to train on these inputs since there is no preprocessing that needs to be done. A fuzzy logic system would need extra processing to be done on the game state to determine its "fuzzy" status.

Since Connect Four has a maximum finite game length of 42 turns -- the number of positions on the board -- we expected a neural network to be able to give us the next move quickly, with an ideal time of 0.5 seconds per decision or less. By comparison, an adaptive algorithm approach like Alpha-Beta pruning or A* algorithms would be looking ahead several turns and would take significantly longer.

1.4 Background

Connect Four was created by Dr. Howard Wexler and introduced to the world in 1974 [5]. It was weakly solved in 1988 independently by both James D. Allen and Victor Allis. That is, it was demonstrated that there is an algorithm that will produce the optimal result for each player [4].

A popular area of research is the development of artificial intelligence that can play common board and card games. The author of [1] lists a few reasons for this work: board games are examples of problems in real-life; it's a way for understanding and furthering techniques in artificial intelligence.

2 Method

2.1 Training Data

Training data for the neural network had 42 inputs and one output. The inputs represented the game state, or current state of pieces on the Connect Four board. Each value represented a single slot in the board. The first value represented the slot at the bottom left corner of the board. The proceeding slots represented the slots found by moving up each column from left to right. A value of ‘0’ represented no player occupied the slot. A value of ‘1’ represented the player who went first occupied the slot. A value of ‘-1’ represented the player who went second occupied the slot.

The output represented the expected result of playing out the game to its end from the given game state. A value in (0, 1] represented a high confidence of winning, with one being the highest confidence. A value in [-1, 0) represented a high confidence of losing, with -1 being the highest confidence. A value of 0 meant the game could go either way or end in a draw.

The figure below shows an example training data set showing that moving in the middle column for the first move results in a high probability of winning the game.

```
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

Figure 2: Sample Training Data

2.1.1 8-Ply Data

Once source of training data was the 8-Ply game results that were accessed from [3]. Each entry of this data contained the game state of a Connect Four game after 8 moves -- four by each player -- and the winner of the game if both players played optimally. A sample of the 8-play data is displayed in figure 3.

```
...
b,b,b,b,b,b,b,b,b,b,b,b,o,b,b,b,b,b,x,o,x,o,x,x,o,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,win
b,b,b,b,b,b,o,b,b,b,b,b,o,b,b,b,b,b,x,o,x,o,x,x,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,win
b,b,b,b,b,b,b,b,b,b,b,b,o,b,b,b,b,b,x,o,x,o,x,x,b,b,b,b,b,b,o,b,b,b,b,b,b,b,b,b,b,b,win
o,b,b,b,b,b,b,b,b,b,b,b,o,b,b,b,b,b,x,o,x,o,x,x,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,win
b,b,b,b,b,b,b,b,b,b,b,b,o,b,b,b,b,b,x,o,x,o,x,x,b,b,b,b,b,b,b,b,b,b,b,b,o,b,b,b,b,b,draw
...
```

Figure 3: 8-Ply Data from [3]

Each line contains 43 comma-separated values. The first 42 represent the board positions and the game piece in each, which is used as the input. The last value is the result of the game if both players play optimally, which is used as the output. In this data, *b* represents a blank position, *x* is the starting player, and *o* is the second player. The result of the game is with respect to the win or loss of the starting player *x*.

2.1.2 Heuristic AI Game Data

Another source of training data came from two AIs play 100,000 games of Connect Four against each other and recording which game states lead to what result. Each time a move was made, the resulting game state was recorded. After the game was finished, the game states were iterated through and marked as either a win, lose or draw, depending on the end result of the player who moved to achieve the game state. The game state was then added to a persistent data structure that mapped each seen game state to its total number of wins, losses and draws.

Once all 100,000 games have been played, the output value of each game state was calculated according to the following formula:

$$\text{output} = (w-l) / (w + l + d)$$

where...

w= # of wins from the game state
l = # of losses from the game state
d = # of draws from the game state

For example, in a hypothetical simulation of 100 games, half of them had a first move of playing in the middle column. Out of those games, 40 resulted in a win, 9 resulted in a loss and 1 resulted in a tie. The output for this example would be: $(40\text{-}9) / (50) = 0.62$. Figure 4 shows the resulting training data from this simulation for moving in the middle column for the first move.

```
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.62]
```

Figure 4: Example Generated Training Data

2.1.2.1 Game Playing Algorithm

The AI that played the 100,000 games of Connect Four was implemented using a potential win heuristic combined with a random move selector. 50% of the time the AI would randomly move and 50% of the time the AI would follow the potential win heuristic. Whenever there was a forced move, the AI would always make that move. A forced move was defined to be a move that immediately resulted in a win for the current player or a move that blocked a win the opponent could take on their next turn.

The reason for using the combination of random movement and a heuristic was to generate a diversity of game states, yet still promote good play. If only the heuristic was followed then each game played would be exactly the same. If only random moves were selected then the data would be too diversified and poor play would be promoted.

The potential win heuristic favored moves that were part of a large number of potential victories. A potential victory is defined as a line of four slots in a row containing no pieces from the opponent. Figure 5 shows that moving in the middle column for the first move results in the being part of seven potential victories.

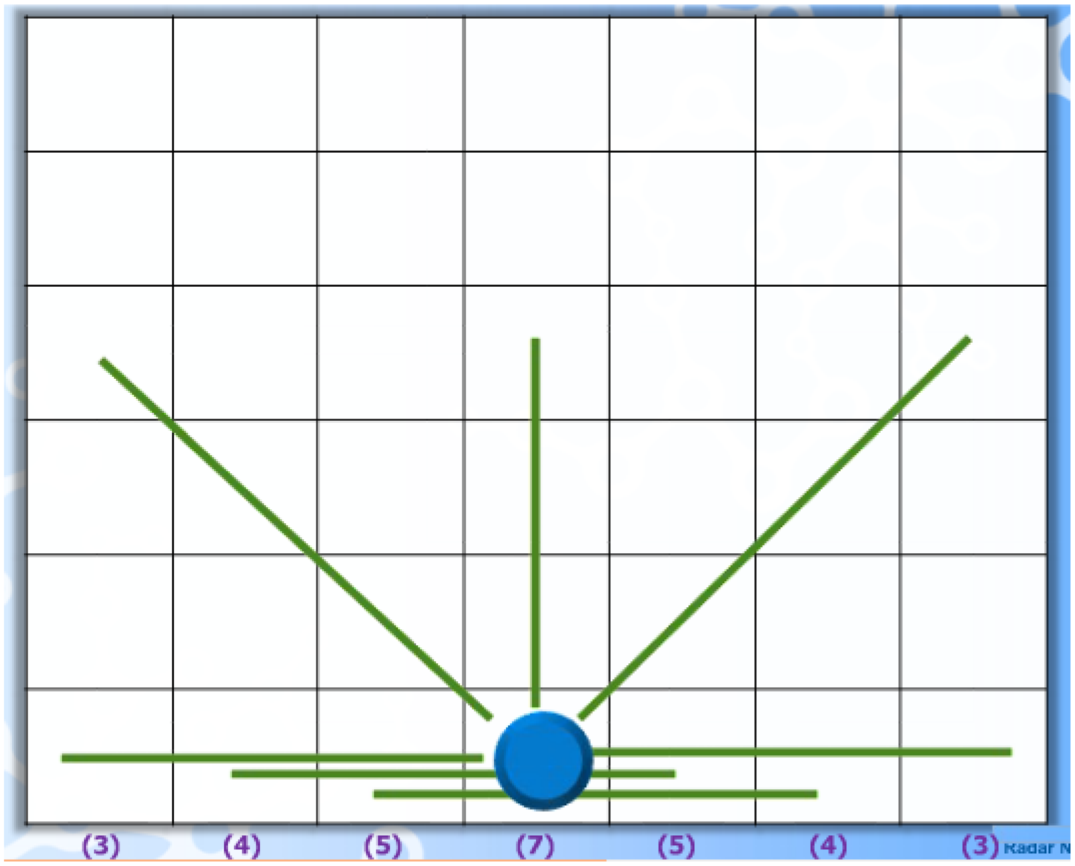

Figure 5: Potential Win Heuristic from [6]

The heuristic also accounts for how close a potential victory is to completion. For example, being part of a potential victory that already has two pieces in it means only one piece is left to win the game and is therefore much more valuable than a potential victory with only one piece. To select the next move using the potential win heuristic, all possible next moves are generated and the one with the highest score is selected. The formula used to calculate the score for the heuristic is as follows:

$$\text{score} = v_0 + 2v_1^2 + 3v_2^3$$

Where v_i= the number of potential victories the piece will be part of with i of the current player's pieces already present.

This equation was discovered experimentally by varying the parameters and playing games against modified versions of the same heuristic. The heuristic that resulted in the most wins on average was selected.

2.2 Neural Networks

We used a back propagation artificial neural network with supervised learning with scaled conjugate gradient backpropagation training function. Input space for these neural networks are the game state of connect-4, which is a representation of the 6 by 7 board. To make it easier for calculation, we flatten this input into a size 42 array, following the same method used by the 8-ply data to provide consistency.

The result accuracy of neural networks are affected by structure and size of neural network, as well as training function used for neural network. In order to get the optimal result, a combination of different parameters are tested. The analysis for these tests are in following analysis section of this report. Exact parameter used for each training set is explained below. The rule of thumb for parameter tuning is that resulting neural network should have high accuracy, good performance, as well as time efficient.

For training and generating neural network, we used the neural network toolbox that is part of Matlab. This toolbox provides simple supervised learning back propagation neural network training.

2.2.1 8-Ply Neural Network

The first neural network was trained on the 8-ply data. We expected this to be the worse of the two neural networks.

Since 8-ply data assumes that the player plays optimally and the result is already deterministic, we can use classification neural network training. For win/draw/lose, we treat each result as one class. The target for training data is adapted to a size three vector, where one element is 1, other two elements are 0, indicating which class this input belongs to.

The resulting neural network has one hidden layer with 20 nodes. Its structure is shown as the following graph.

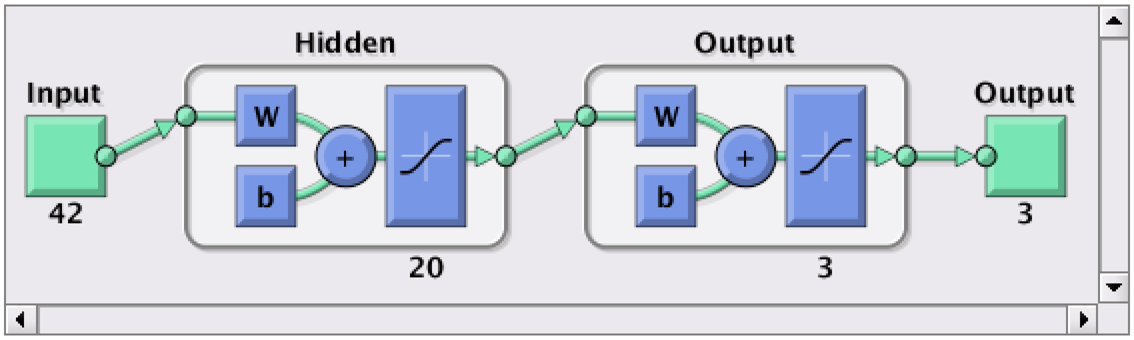

Figure 6: Diagram of 8-Ply Neural Network Structure

For training, 70% of the 8-ply data is randomly selected as input and output. With above set-up, the confusion graph and performance graph is listed below.

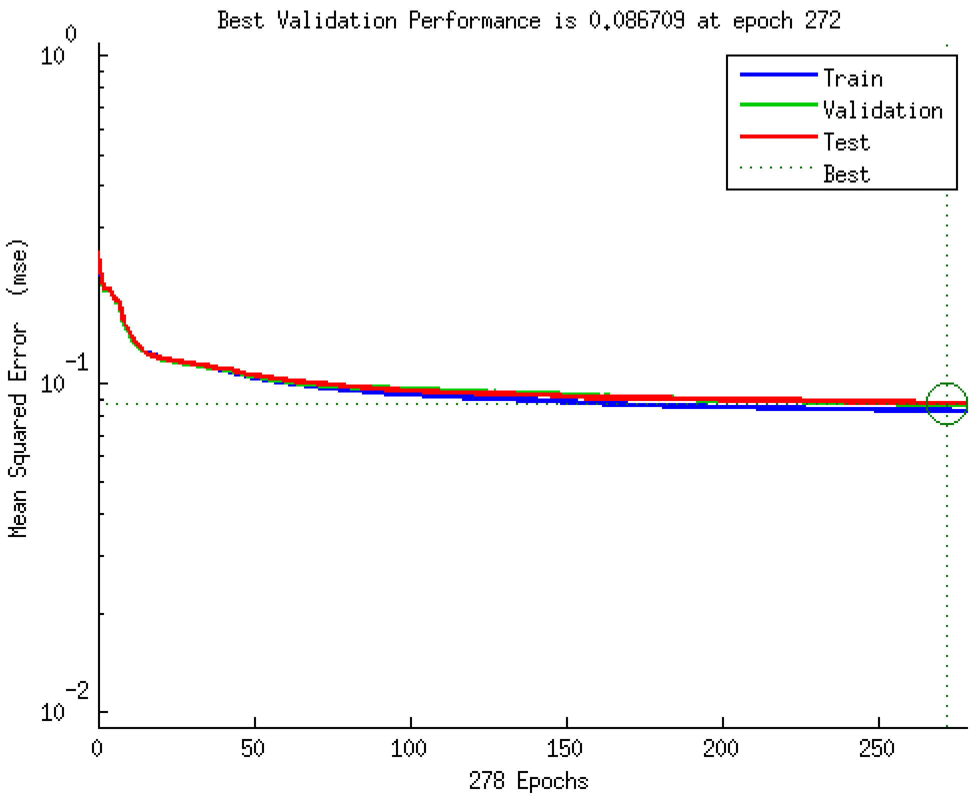

Figure 7: Performance graph of 8-ply neural network

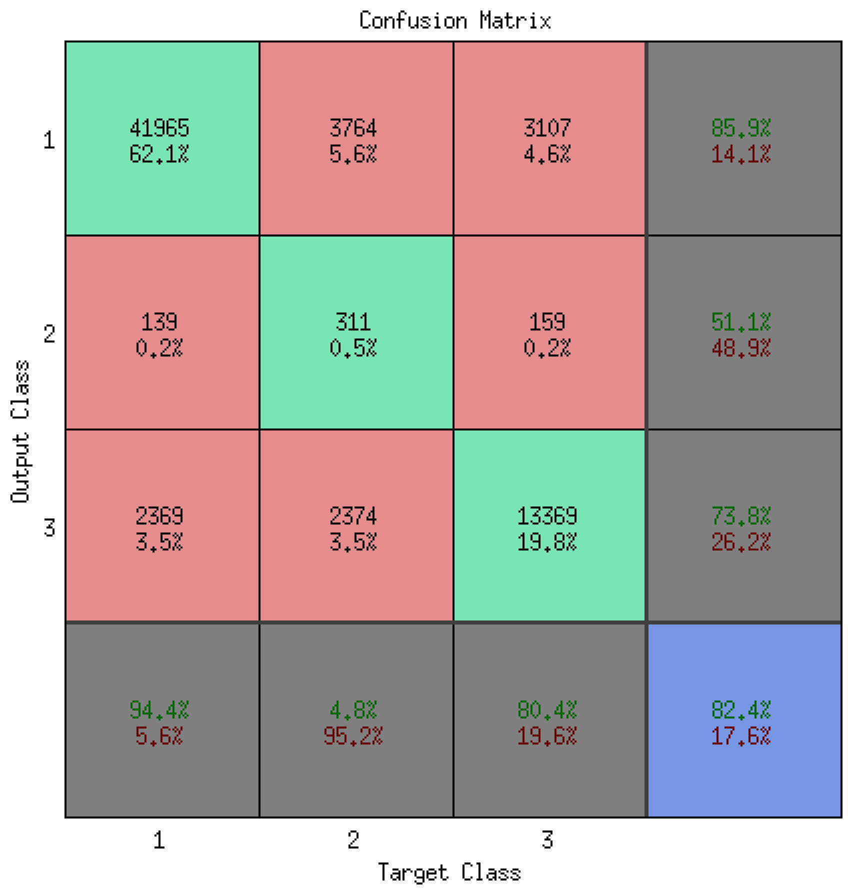

Figure 8: Confusion graph of 8-ply neural network

2.2.2 Heuristic Neural Network

The second neural network was trained on the data generated from the heuristic AI playing 100,000 games. We expected this to be the better of the two neural networks.

Since heuristic training data has output in term of possibilities, this neural network is trained as regression instead of classification.

The resulting neural network ends up being used is a regression neural network with one hidden layer of 15 hidden nodes, as shown below.

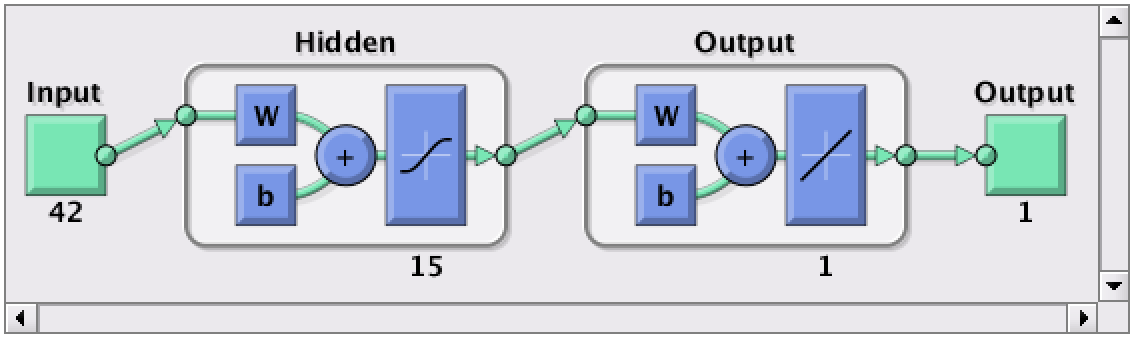

Figure 9: Diagram of Heuristic Neural Network Structure

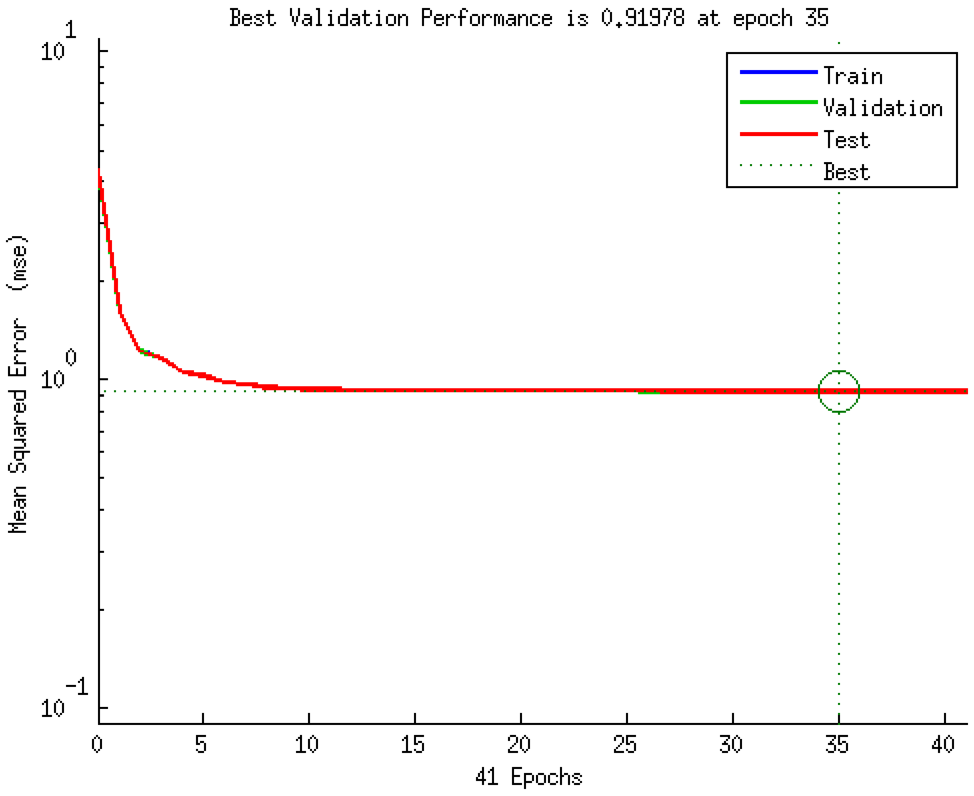

Figure 10: Performance graph of Heuristic neural network

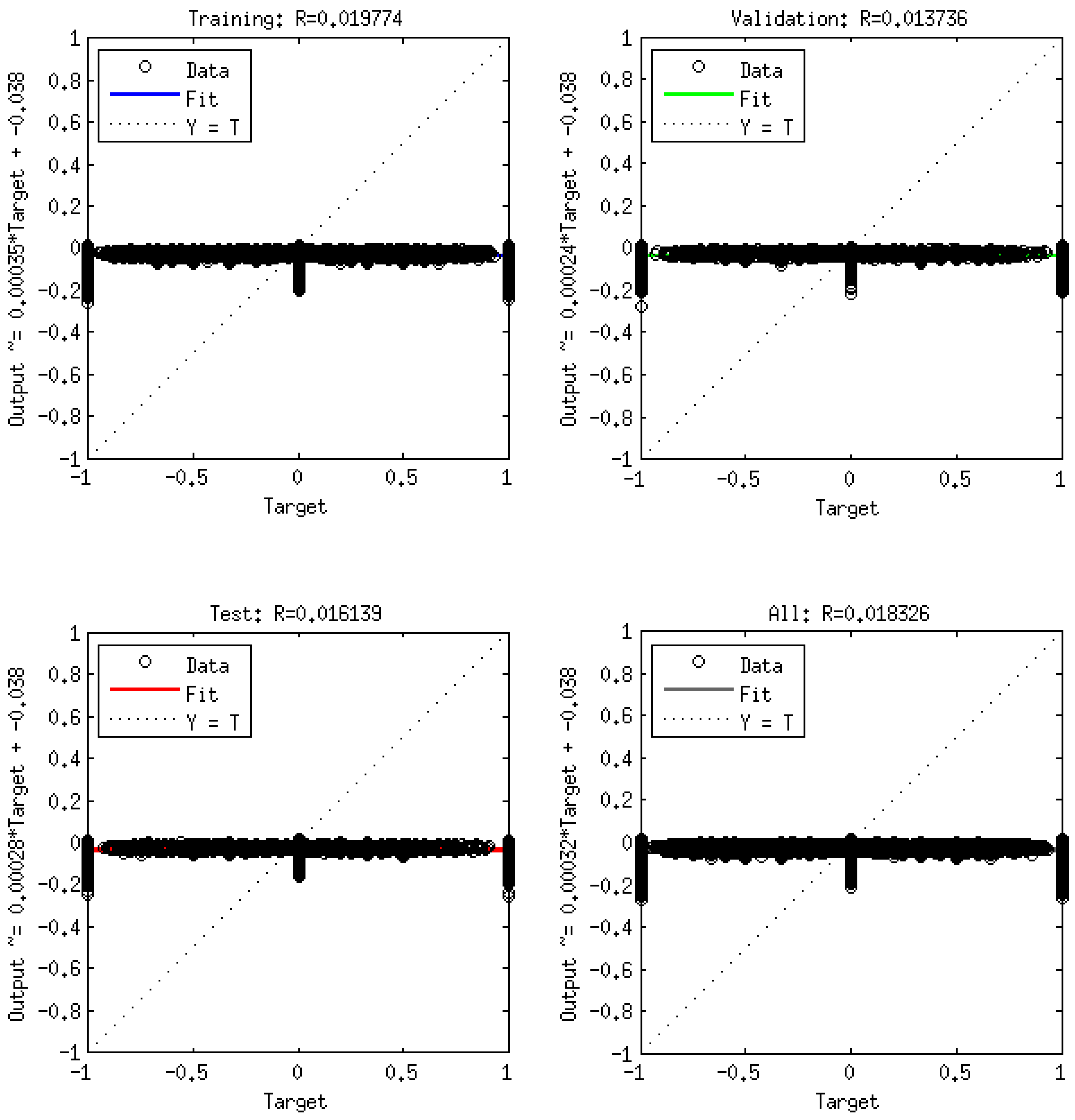

Figure 11: Regression graph of Heuristic neural network

3 Analysis and Results

3.1 Gameplay Results

We chose the following scenarios to analyze the success of our method:

- Human player versus Heuristic NN
- Human player versus 8-ply NN
- Heuristic NN versus 8-ply NN
- Random move AI vs Heuristic NN
- Random move AI vs 8-ply NN

Figure 12 shows the results of these evaluations. It's clear that in all cases, the 8-ply-trained neural network performed better than the heuristic-trained neural network. This confirms the assertion that small, well-chosen data sets provide better training than large, random data sets [2]. We believe this is due to the training eventually contradicting itself and causing mistakes rather than always improving over each epoch of training.

Both AIs performed poorly against an intermediate level human player and the 8-ply-trained neural network performed well against a random AI. The details of these results can be seen in the figure below. Overall, both AIs performed at the level of a beginner Connect Four player.

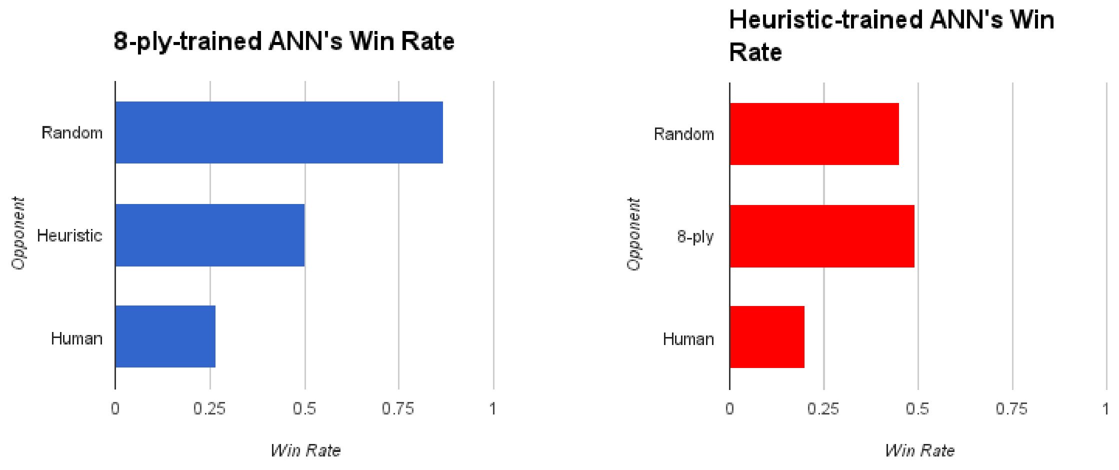

Figure 12: Win Rates of the two neural networks.

3.1.1 Human vs Computer

Evaluating the win-loss record of the neural networks against the human player shows us how much progress we have made in our goal of creating a "competent" neural network. Ideally, the neural networks would be able to win against an intermediate human player about half of the time. However, this was not the case. Instead, our neural networks were able to win 20 - 26% of games.

It’s possible this indicates that our players were better than intermediate human players. Determining the skill level of human players in an objective manner is difficult, but we do not believe our players to be exceptional. Prior to our play evaluation, they were instructed to avoid playing according to high-level theory, such as control of zugzwang, and were instead to play quickly, rather than scrutinizing each move.

3.1.2 Computer vs Computer

We hypothesized that having the two neural networks play against each other would show the relative progress each neural network has made in its training. Interestingly, during the evaluation they played nearly equally, with the Heuristic-trained ANN lagging behind by just 1% of games played, which seemingly contradicts the hypothesis.

The alternate conclusion to draw from this would be that they are equally skilled, but this seems unlikely as their performance against both the random player and human player are not very similar.

So then we must ask ourselves how the match between the two neural networks would be so even, especially considering the fact that the heuristic-trained neural network performed its best against the 8-ply-trained neural network. It’s possible that the training data provided biased the 8-ply-trained network to play in a way to which the heuristic-trained network was able to counter somewhat easily.

3.1.3 Computer vs Random

Having the neural networks play against an AI that makes a random move shows a minimum bar of success to measure the neural networks against. Ideally, the neural networks would be able to win against the random AI the majority of the time, as we believed it would demonstrate whether or not the training is having a positive effect.

When playing against random, the neural network trained with the dataset of 8-ply combinations played much more effectively than the Heuristic-trained network. One possible reason for this is that the Heuristic training data contained bias toward a certain sequence. For instance, the heuristic used was biased toward playing the first piece in the middle (4th) column. However, this should have been mitigated by the partially random choice of moves.

It is interesting to note that not only was the Heuristic-trained network worse against random than the other neural network, but it was also winning less than half of the games versus random. This doesn’t indicate that random play should be better than it against humans. After all, a human of just about any skill should be able to easily play around a random opponent most of the time. However, it is rather concerning that it is so vulnerable to random play.

3.2 Analysis of Neural Network Approach

The following analysis used the 8-ply neural network. The analysis was done by changing one of the many parameters that determine a neural network and documenting the resulting confusion and performance.

Through testing the neural network and collecting data , it was shown that when using a neural network, the most time consuming step was loading an already trained neural network into memory. On average, it took 0.15 seconds for a neural network to generate a result. Loading an already trained neural network, on the other hand, took 1.2 seconds on average. These results were consistent across all neural networks we tested with.

Training a neural network took much longer than loading a neural network. Even the simplest neural network with 8-ply training data took around 1 minute to compute. Aso, parameter tuning had a large impact on training timing.

When talking about timing issues for the neural network problem, we focused on the how parameters affect timing for trainings. Another focus for using the neural network was how accurate its results were. This alongside timing issue were the two direction for optimization through parameter tweaking.

The parameters we identified for neural networks were the training algorithm, size of the neural network, structure of the neural network, and the input space of the neural network.

3.2.1 Training Algorithm of Neural Network

The training algorithm was a determining factor in how the neural network performed and how much time was required to train.

The following table used a neural network of 10 hidden nodes with size 42 input space. There’s a clear trade off between timing and performance.

Function Name	**Algorithm**	**Timing (s)**	**Performance**
trainscg	Scaled conjugate gradient	30	0.1323
trainlm	Levenberg-Marquardt	1200+(*)	-
trainoss	One step secant	230	0.1045
trainbfg	BFGS quasi-Newton	715	0.0952

(*) exceeds CPU time limit

3.2.2 Size of Neural Network

Neural networks have an optimal size relative to their input size. In this section we try to find the optimal number of hidden neurons. We started with a small neural network and increased its size until the performance stopped improving. Comparing the following result with our actual neural network, the performance and accuracy were about the same level. On the other hand, neural network with only 20 neurons took less time to train. Thus we concluded that 20 nodes in one hidden layer was an optimal size to use for our neural network.

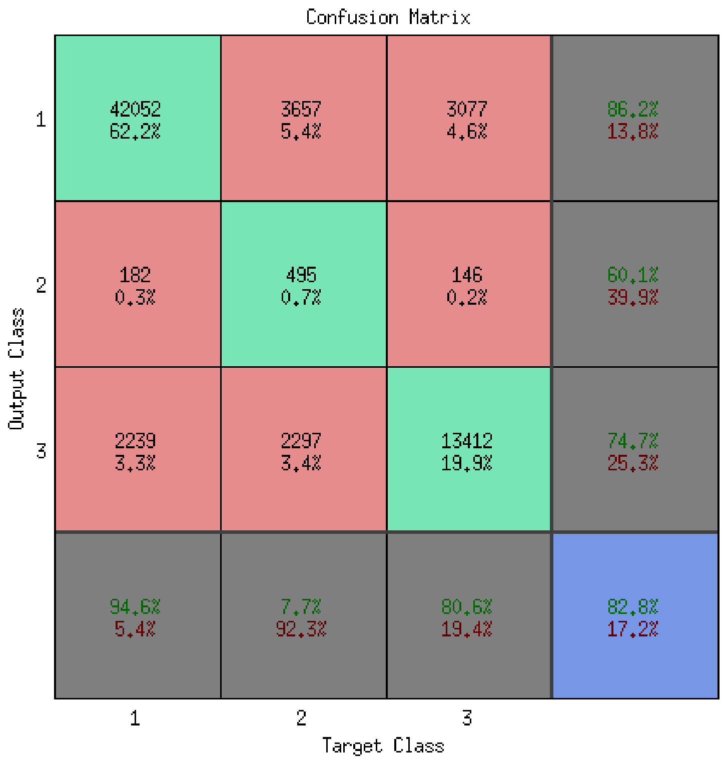

Figure 13: Confusion of one hidden layer with 30 nodes

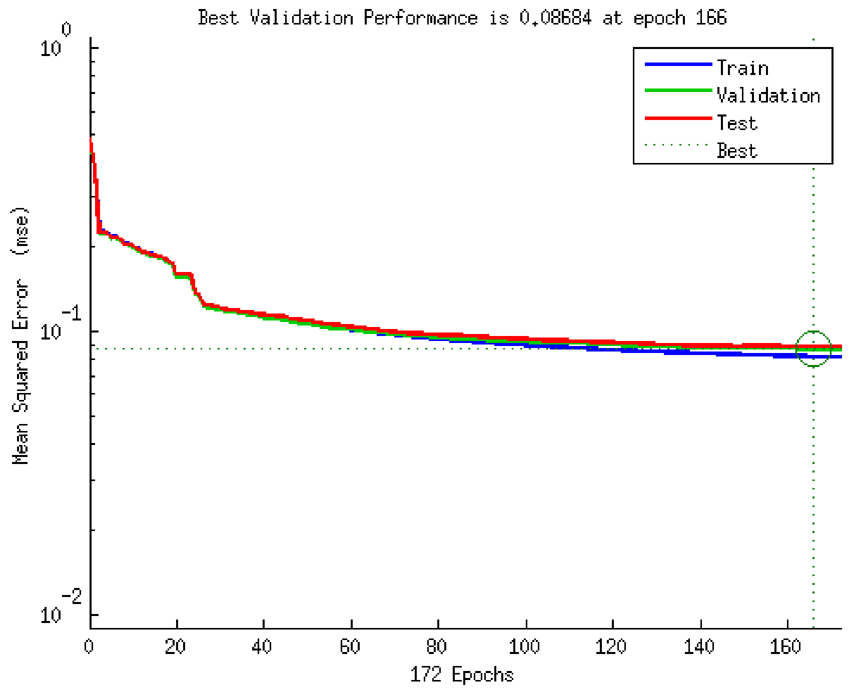

Figure 14: Performance graph of one hidden layer of 30 nodes

3.2.3 Structure of Neural Network

To determine the structure of the neural network, in other words, how many hidden layers the neural network should have, a similar method was used. We started with a simple neural network of one hidden layer and 10 nodes. We increased the number of hidden layers and observed how it affected the performance of the neural network.

As the following charts show, a neural network with two hidden layers had worse performance than one with one hidden layer. For our size of input data, anything more than one hidden layer was way too many neurons to be trained properly. Thus we used one hidden layer for our neural network.

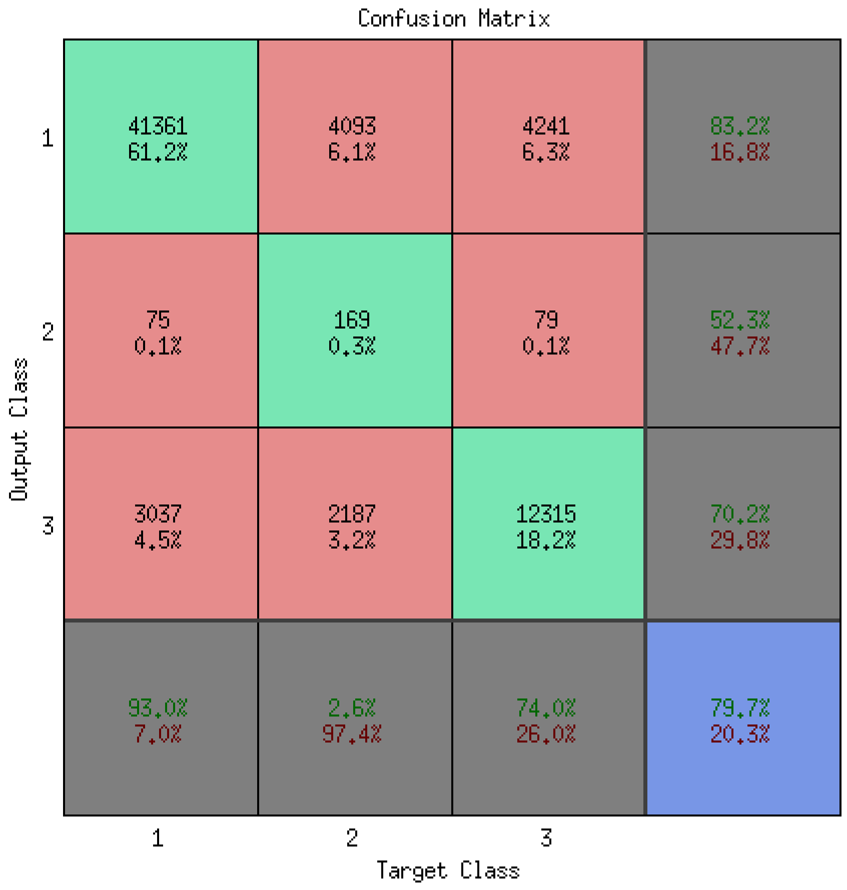

Figure 15: Confusion graph of one hidden layer of 10 nodes

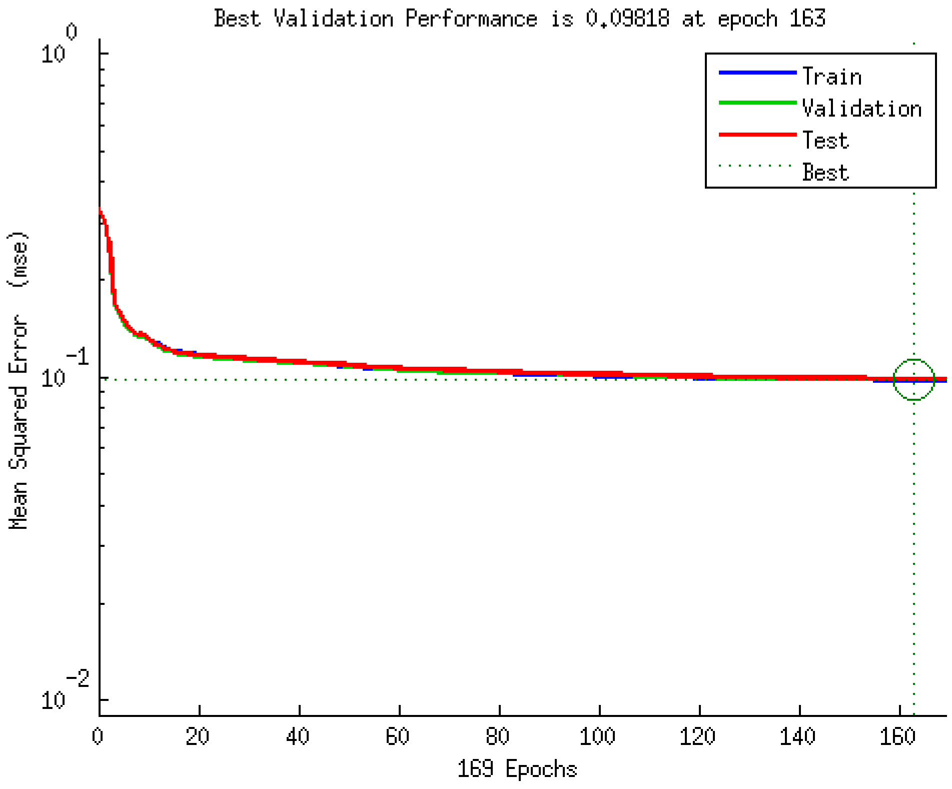

Figure 16: Performance graph of one hidden layer with 10 nodes

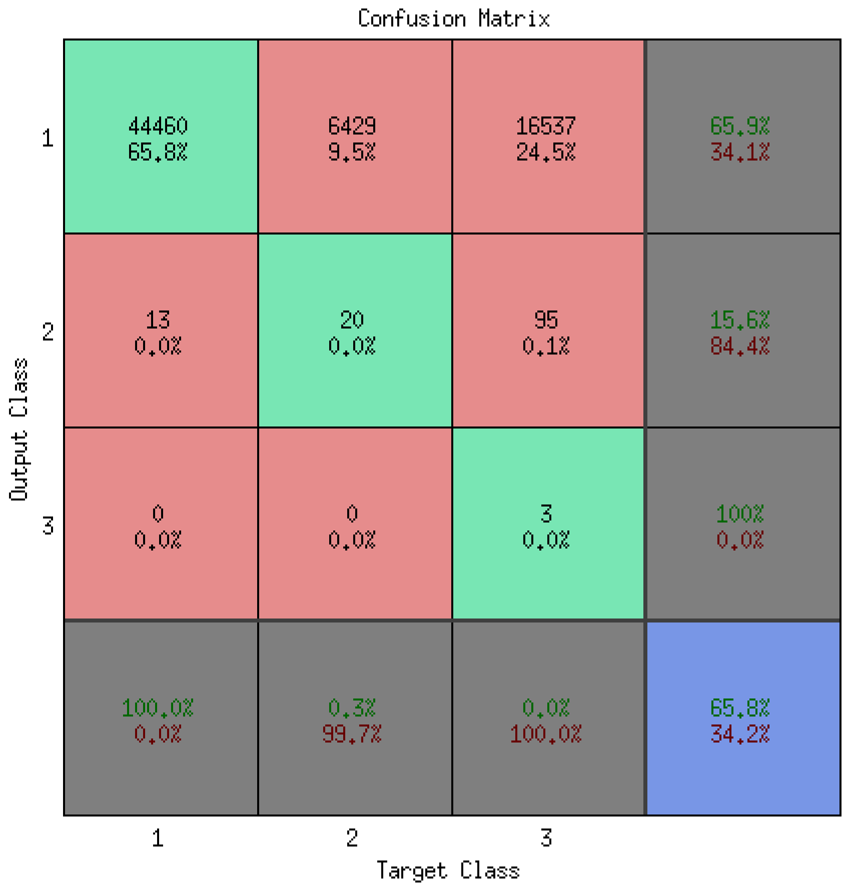

Figure 17: Confusion graph of two hidden layer of [10 5] nodes

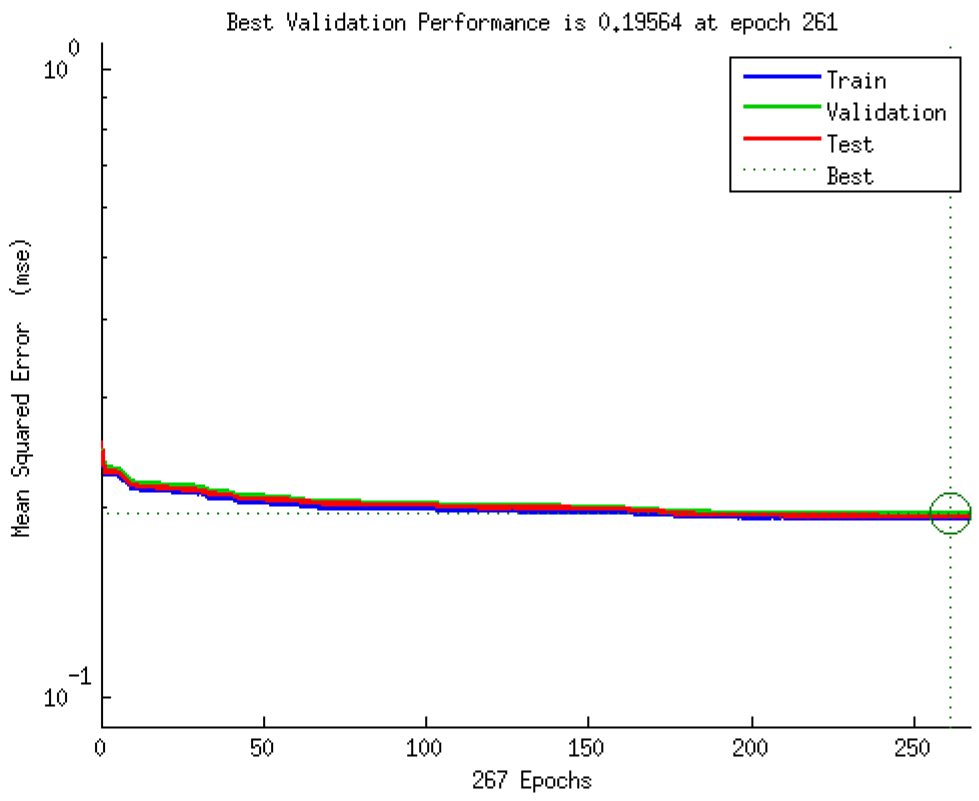

Figure 18: Performance graph of two hidden layer [10 5] nodes

3.2.4 Input Space of Neural Network

The following analysis focused on the scalability of our neural network. Since the Connect Four game board was a fixed sized, it did not make sense for the input space to be anything other than a size 42 array; however, other games might. One of the advantages of using game states to train the neural networks was that the neural network does not require a priori knowledge of game rules. This makes it highly applicable for other games. It was a possibility that the input size of another game would be very large, making scalability of input space a potential problem.

In most neural network applications, the input space is rather small. A size 42 array is considerably large by comparison. Also, our group had limited access to computer resources, so the CPU as well as memory space was very sparse. A complicated neural network had the risk of exceeding our processing limit.

This problem introduced by limited resources was manageable with Connect Four. In this section we explored the relationship between input space size and training time in order to discover how scalable our solution was with respect to input space size.

Since we already had 8-ply data available, we used reduced subsets of its input to test how long training would take. However, there were several problems with this approach. Firstly, our training algorithm contained certain randomness, meaning the result was not entirely deterministic. Secondly, by taking only part of the input, the training data was no longer guaranteed to be a precise representation. Thirdly, and most importantly, computation time was highly dependant on resources located on remote server, which was not consistent over time. Despite this, the approach stilled portray a relatively accurate relationship between input space size and training time. The results can be seen below.

Input space size	**Timing (min:sec)**
8	1:24
10	1:25
16	1:35
20	1:55
30	2:07
32	2:14
40	2:37

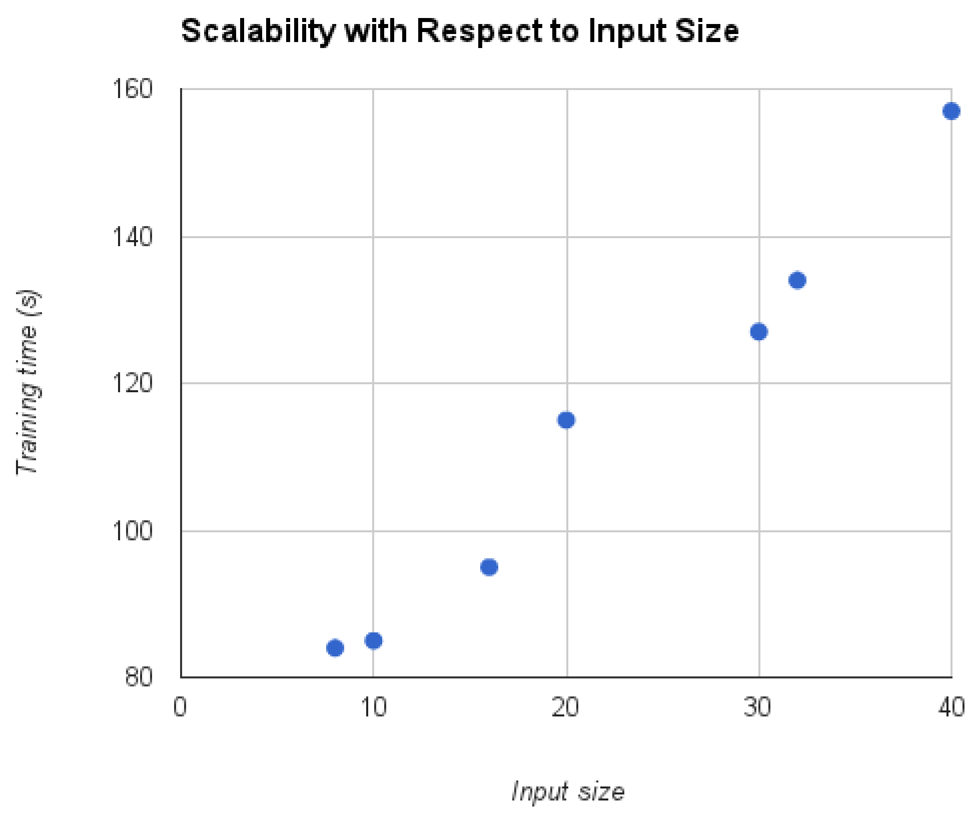

Figure 19: Input space compared to training time

The results from figure 19 show that the training time grows linearly with input size and is therefore able to scale to more complex games with larger input sizes. Based on the observed trend, training a game with 2000 input (about 50 times larger than the input for Connect Four) should take about one hour, which is a manageable amount of time. These results show that this method of learning may be an effective technique for developing AIs for games that have highly complex rules which would be hard to program into an AI.

4 Conclusion

Developing an AI that learns only from game states without knowing the rules of the game is a viable approach, especially for games where quick decision making is required without needing a high level of skill and games where other types of AI would be difficult to implement due to the complexity of the games rules.

The learning approach has the advantage of being simple to implement even for games with highly complex rules. This advantage comes from the fact that the AI only needs a large set of game states to learn how to play the game, and not any other knowledge about the game. Training time scales linearly with the number of inputs, so complex games with large game states can be supported.

The AI makes decisions very quickly, even if the game is complex in nature. The AI can achieve this because it only has to ask the neural network to evaluate each of the next possible game states. The complex decision making that other AIs have to do has been already done when the neural network was trained.

The 8-ply-trained AI performed at a level similar to a beginner Connect Four player, beating an intermediate level human player only 27% of the time. The AI was notably better than a purely random algorithm, beating it 87% of the time. In comparison, the heuristic-trained AI performed worse in all categories.

4.1 Future Improvements

There were several improvements that could be made for future analysis. One of these improvements is to do a better job generating testing data. This can be done by playing more games of Connect Four and using more in depth AI such as using game tree searching.

Another improvement would be to always take forced moves when evaluating the two neural network AIs. This would improve performance against human players and would test the harder decisions the AI had to make, rather than those that can be easily implemented.

Lastly, performance evaluation could be performed on more games then just Connect Four. We determined that this approach could be used for other games, but having actual performance results would make a stronger argument. Other games with different levels of complexity may have different level of skill then the Connect Four AI.

References

[1] I. Ghory, “Reinforcement Learning in Board Games”. University of Bristol, 2004. Retrieved from CiteSeerX: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.7712.

[2] M. O. Schneider and J. L. Rosa, “Neural Connect Four - A Connectionist Approach to the Game”. Pontifical Catholic University of Campinas, 2005. Retrieved from CiteSeerX: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.8621.

[3] J. Tromp, “Connect-4 Data Set”. University of California, 1995. Retrieved from UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets/Connect-4.

[4] V. Allis, “A Knowledge-based Approach of Connect-Four”. Master’s thesis, Vrije Universiteit, 1988. Retrieved from CiteSeerX: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.2778.

[5] Anon., “A City College Toy Story”. CUNY Matters. City University of New York, 2001. Retrieved from City University of New York: http://www1.cuny.edu/events/cunymatters/2001_winter/backmatter.htm

[6] M. Yasir., “Artificial Intelligence - Connect Four Game”. Retrieved from Authorstream: http://www.authorstream.com/Presentation/mysyasir-149155-artificial-intelligence-connect-4-game-ai-four-heuristic-model-presentation1-science-technology-ppt-powerpoint/