# Evolving Connect Four

## Using Neural Networks & Genetic Algorithms

Connect Four is a grid based game that requires two opposing players to place discs into columns with the goal of having four discs in a row or diagonally. It is also a solved game with perfect information where the first player can always win with perfect play. While not overly complex, Connect Four is a great game to use to being understanding the science of Artificial Intelligence.

A Neural Network can be represented in a similar manner to how the human body works. As infants, we do not know exactly how our body works. It takes months to learn to crawl, then walk, and then run. As adults we do not have to think about walking; we just walk. The activity is wired into our brain. The same is similar in a computerized neural network. A network of neurons that interconnect with one another takes some input and determines the type of output to produce. It is trained by receiving predetermined inputs and then propagated until the desired outputs are achieved.

Previously we had used the Alpha Beta Pruning Algorithm (AB AI) to play the game Connect Four. This algorithm provides weighted values to each decision in order for the AI to chose the best move to make. It does this by mimicking a played game versus a real player and attempts to evaluate every move that it and its opponent can make. The AI delves a predetermined number of moves deep to make a decision.

The problem with using the AB AI to play Connect Four is that the smarter the AI becomes, the longer it takes for the AI to make a decision on what move to make. For Connect Four, the AB AI has to chose one of seven possible moves each turn. For each possible move, it has to evaluate the decision of each possible counter move that its opponent could make. Two levels deep and the AI has forty nine combinations to evaluate. Three levels deep will take even longer with three hundred and forty three different combinations! In reality, since the AI will be ignoring possible combinations that do not lead to a good outcome it won't be quite that many, but the deeper the AI decides to go the more possible combinations that it may have to evaluate and the longer it takes for an AI to determine which branches could be considered taking and which it should completely ignore. Since it is not possible to win the game in less than seven moves (four moves for player one, three moves for player two), the AI would have to go at least seven levels deep in order to know that its opponent will not let them win in seven moves.

On the other hand, a Neural Network does not need to use any extra computational power to make these decisions; it only takes a number of inputs and provides some sort of output. The difference between the two approaches is that the Neural Network does all the extra calculations before it is used. It is trained to provide output based on the input that it receives. Using the previous analogies of children, a child may point to a shoe and then a sock and call both a shoe. If not corrected then the child will learn that both are the same type of object.

Normally, a Neural Network is trained using a set of predetermined inputs and outputs. The Network starts off by feed forwarding an input and generates an output. The output is compared to the predetermined output and determines how close the network was in achieving the answer. Then, through back propagation, the Network rebalances the weights connecting each node. In a sense, this is the Network "practicing" a particular skill. The Network continues this until it reaches a predetermined stopping point. This could be once the network converges to a particular error percentage or if the network has gone through all of the training data. This allows the Network to learn.

The problem with using this method for a game, such as Connect Four, is that it may be difficult to determine the correct data set with which to train the network. For example, in a different game, it may be possible to create a data set with all the best options when interacting between two objects out of many. It may also be possible to create a data set for a combination of three different objects out of many. But what about four? What if the order the objects came in mattered? What if the correct decision is not only based on your past and possible moves, but also your opponents past and possible moves? The more complicated the game, the more difficult it becomes to create a data set that could train all the best possible decisions.

Instead of using the normal method of utilizing training data and back propagation, I decided to use a genetic approach to train the network. This would work similar to normal Genetic Algorithms where each member of the generation would be given a fitness value, a rank based on their fitness value, and then randomly selected based on their rank in order to create the next generation. I used the same Utility function that the AB AI uses to evaluate its choices.

In order to cross two networks into a new network, I used the method of converting each network into a string as described in the paper "The Encoding Problem"1. Each node is given an identifier, its parent node with associated weight, and additional relevant information. I used the following template for each layer:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Previous Index | Weight | Node | Index | Additional Info |
| Input Layer |  |  | Node | 1 | Coordinates |
| Hidden Layer | 1 | .27 | Node | 2 |  |
| Output Layer | 2 | .71 | Node | 3 | OutputMethod |

The encoded string is pipe delimited ('|') in order to know where the string could be crossed with other networks. Once the new encoded string was generated, it could be used to create the actual network.

For training, I started off with 100 randomly generated neural networks. Each network had forty two input nodes to represent each position on the board fifty hidden nodes, and seven output nodes that would determine the column that the AI would chose as its next move. Each layer would propagate completely to the next with a random weight for each connection. I would then have each network in the current generation play against the AB AI that only looks at possible moves a few levels deep. Once each AI has played versus the algorithm, their fitness is evaluated based on the same Utility function that the AB AI uses to evaluate each move.

I kept the training running until each AI in a generation could beat the AI that it was paired against. I then swapped the AB AI that was used in my Connect Four program with a randomly selected member of the generation as my opponent. Unfortunately, I did not have a difficult time defeating the AI. I realized that the network was only using the AB AI as a limited training set and was only learning how to defeat one opponent. In reality the network should have been trained against either different opponents (difficulty would have to roughly stay the same) or would need a completely different method to determine the fitness of the network.

In the future, instead of using the same Utility function that the AB algorithm uses, I will use a technique used in the paper "Playing Tic-Tac-Toe Using Genetic Neural Network with Double Transfer Functions"2. The method evaluates each move as it is being placed. This evaluation would work as a more accurate fitness function which will allow the AI to train and evolve using the best possible move instead of learning based on how a different AI plays. This will allow the network AI to not only learn the correct way to play but to also learn how out smart its opponent. It will not have to be taught individual techniques for playing the game.

## References

1 Playing Tic-Tac-Toe Using Genetic Neural Network with Double Transfer Functions (<http://file.scirp.org/pdf/JILSA20110100005_14407830.pdf>)

2 The Encoding Problem (<http://homepages.inf.ed.ac.uk/pkoehn/publications/gann94.pdf>)