UTSA Journal of Undergraduate Research and Scholarly Works
Volume 7 December 2020

Application of Neuroevolution in Blackjack

Iraz K. Tejani
University of Texas at San Antonio
School of Business

ABSTRACT

Blackjack is one of the few casino games with an extremely low house edge. In the past, many brute force simulations have been done to derive basic strategy. Classical brute force methods are tedious, time consuming, and often require hundreds of millions of games played to achieve results. In this project, I use reinforcement learning, specifically neuroevolution (NE), which is an attempt to simulate biological evolution, to see if an artificial neural net (ANN) can evolve to learn basic strategy and achieve the theoretical maxima provided by a basic strategy simulation. Two main simulations are run in this project, one using basic strategy charts and the other using the evolved ANN. These are then compared to see how effective the ANN was in learning strategy as well as how quickly it was able to learn.

Keywords: Neuroevolution of Augmenting Topologies, Blackjack, Reinforcement Learning, Artificial Neural Net

1. Introduction

Neuroevolution (NE) has been widely researched and studied for the past thirty years. While in the beginning NE algorithms were primarily focused on evolving only the weights of connections in an artificial neural network (ANN), more recently in 2001 with the publication of the Neuroevolution of Augmenting Topologies (NEAT) paper, it was found that evolving the network topologies alongside the weights outperformed traditional fixed-topology networks [1]. In comparison to a NE system with a fixed topology, Enforced Subpopulations (ESP), NEAT performed five times faster. Due to this discovery, NE was able to become much more viable in use for reinforcement learning tasks since it requires drastically less computing power. In my project, I assess the ability of NEAT to learn strategy in blackjack in comparison to earlier brute force simulations. The first brute force simulations arose in the 1980s where Julian Braun of IBM had written a program which would go through each possible combination of cards trying every possible move and state the probabilities of the various outcomes [2]. Needless to say, this is very computationally expensive and requires hundreds of millions of games played for optimal results.

1.1 Motivation

The primary motivation for this project was to evaluate the effectiveness of using NE to derive strategy in blackjack. Blackjack was chosen since it has one of the lowest house edges out of your standard selection of casino games which basically means you still lose money, just a lot slower than other games [3]. The house edge varies between each casino where it is common to have different rules and even different rule sets for specific tables. For example, one table can have a single deck shoe (device which holds dealer's cards) and will allow the player to double after a split. While another table in the same casino may have a six deck shoe, which is known to lower the house edge, but not allow the player to double after a split. Some casinos have side bets as well, allowing you to bet on the values of cards such as a flush or a straight however I will not be addressing these since they are not standardized in any set of rules.

1.2 NEAT Overview

This section provides a high level understanding of how NEAT works. For this project, I use the library NEAT-Python which is highly documented and has been extensively used for training games in the past [4]. At first a population of default genomes is created with minimal structure. Each genome contains genetic encoding which details the connection nodes and their associated details such as weight, innovation number (IN), and the in/out nodes (see Figure 1).

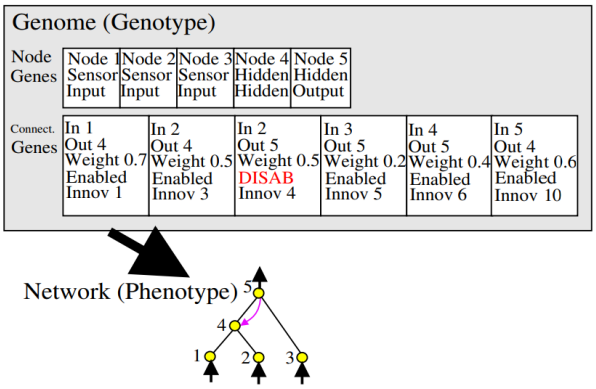

Figure 1. Illustration of genome translated to a network [1].

The main method of evolution of these genomes is through two types of mutations. One is the add connection mutation, and the other is an add node mutation. In the add connection mutation, two unconnected nodes will form a connection and in the add node mutation a connection is split in two adding a node in between [1]. In the event of a crossover you will need to reference the IN. The IN is a number assigned to each

gene in a so that when performing a crossover, you are able to determine the historical context of the gene and match accordingly. The last main concept NEAT uses is speciation. To preserve certain network structures and their mutations, the population is split into separate species so that they can compete with others in their own species. The performance of the population is judged by a fitness function which returns a single value that determines the effectiveness of a genome. The most fit can be the genome with either the maximum, minimum, or mean fitness value.

1.3 Blackjack Rules

For this version of blackjack, the following rules are used:

- Single deck, cards are shuffled every round.
- Dealer will stand on soft 17 hands.
- Payouts are: 3-2 for hands of Blackjack (natural), 1-1 for all other wins.
- Split up to two hands every round.
- Double after split is allowed.
- Upon splitting Aces, re-splitting is not allowed. Only one more card will be received. If a split Ace receives a 10-value card, that hand is not considered Blackjack.
- It is allowed to surrender the initial two-card hand returning half of the original bet.

This rule set provides a house edge of about 0.46% with a standard deviation of 1.14, which seems low compared to other variants but is still found common in casinos regardless [3].

2. Basic Strategy Simulation

To assess the effectiveness of the trained net, there is a need to have a baseline to compare our results with. To do this I first create a simulation environment which plays using premade strategy charts to determine the theoretical maximum rate of return, as well as the percent of games won. These strategy charts were derived from the brute force methods mentioned earlier. A player can either have a hard, soft, or splittable hand. If the hand contains no ace then it is labeled hard, however if it does then it will be considered soft. This is because an ace card has a dynamic value where it can be either 11 or 1 depending on whether the player has bust. There is a different chart for each of the three scenarios.

Player	Dealer's card									
hard	2	3	4	5	6	7	8	9	10	A
4-8	H	H	H	H	H	H	H	H	H	H
9	H	Dh	Dh	Dh	Dh	H	H	H	H	H
10	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	H	H
11	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	H
12	H	H	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	Rh	H
16	S	S	S	S	S	H	H	Rh	Rh	Rh
17+	S	S	S	S	S	S	S	S	S	S
soft	2	3	4	5	6	7	8	9	10	A
13	H	H	H	Dh	Dh	H	H	H	H	H
14	H	H	H	Dh	Dh	H	H	H	H	H
15	H	H	Dh	Dh	Dh	H	H	H	H	H
16	H	H	Dh	Dh	Dh	H	H	H	H	H
17	H	Dh	Dh	Dh	Dh	H	H	H	H	H
18	S	Ds	Ds	Ds	Ds	S	S	H	H	H
19+	S	S	S	S	S	S	S	S	S	S
splits	2	3	4	5	6	7	8	9	10	A
2,2	Ph	Ph	P	P	P	P	H	H	H	H
3,3	Ph	Ph	P	P	P	P	H	H	H	H
4,4	H	H	H	Ph	Ph	H	H	H	H	H
6,6	Ph	P	P	P	P	H	H	H	H	H
7,7	P	P	P	P	P	P	H	H	H	H
8,8	P	P	P	P	P	P	P	P	P	P
9,9	P	P	P	P	P	S	P	P	S	S
A,A	P	P	P	P	P	P	P	P	P	P

Figure 2. Strategy chart used for the simulation. H(hit), S(Stand), Dh/Ds(Double), Rh(Surrender if allowed otherwise hit), P/Ph (Split) [3].

To use this chart in the simulator, I simply save it as a matrix, or a list of lists in python, and pull values from it depending on the value of the hand, dealer shown card, and whether the hand is hard soft or splittable. This simulation was run one million times with a runtime of about 66

seconds while saving all game data. It could be run further, but with saving the game data, the file size quickly adds up with about one megabyte per second ran.

Results (Percent)	
Wins	42.54
Dealer Wins	45.06
Push	7.99
Surrendered	4.41
Player Blackjack	4.57
Dealer Blackjack	4.57
Hard	86.43
Soft	14.63

Table 1. Game Results from a simulation of one million games using basic strategy.

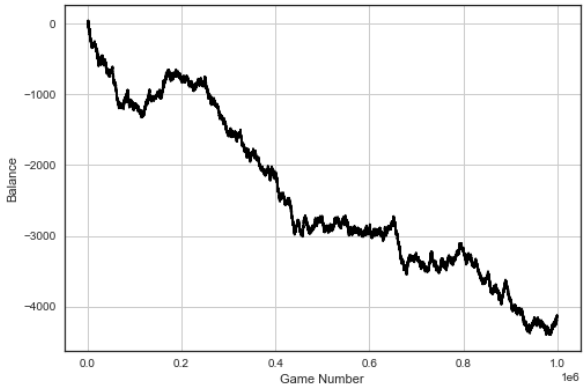

Figure 3. Results from one million games of blackjack using basic strategy and the balance of the player throughout the games.

Results from the simulation show that a player can expect a theoretical maximum win rate of around 42.6% and a rate of return of about 99.6% (see Table 1 and Figure 3). Each hand was only given one dollar to bet not allowing any type of dynamic betting.

3. Applying NEAT

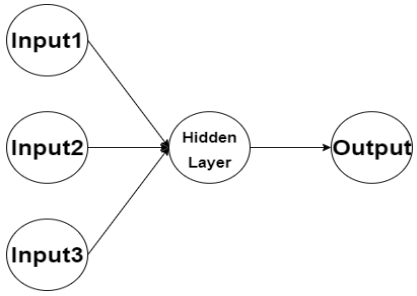

Figure 4. Beginning minimal structure of each genome.

3.1 Inputs and Outputs

For the nets' inputs, I will be using 3 different values. Input 1 will be a binary value indicating whether the deck is a hard of soft deck.

$$f(v) = \begin{cases} 1 & \text{if } 11 \in \{v_1, \dots, v_n\} \\ 0 & \text{otherwise} \end{cases}$$

Where v is a vector of the player's current hand. This is an important input because strategy can greatly differ depending if the player has a usable ace or not. Having an ace allows the player to take greater risk on higher hand totals knowing they have a cushion preventing them from a bust. The second input is a sum of the players hand.

$$\sum_{i=1}^{n} v_i$$

Rather than inputting each individual card, I input the sum of cards instead because the single values are irrelevant. A case in which there is a need to know the individual values would be if the player was keeping a count (a method players use to gain an advantage by keeping track of cards played). The third input is the value of the dealer card that is shown. For the outputs, I have one single node as the output

split into five actions. Since I am using the tanh activation function, the value of the output node will be between -1 and 1 therefore each action will be assigned a range of values between the given limits.

$$f(o) = \begin{cases} 1 & if\ o < -0.6 \\ 2 & if -0.6 \leq o < -0.2 \\ 3 & if -0.2 \leq o < 0.2 \\ 4 & if\ 0.2 \leq o < 0.6 \\ 5 & if\ 0.6 \leq o \end{cases}$$

Where o is an output value returned from the net between -1 and 1. These values correspond to the following moves: 1(surrender), 2(stand), 3(hit), 4(double), 5(split). If the net attempts to make an illegal move such as splitting when it does not hold two matching cards, it will be forced to hit.

3.2 Fitness Functions

Two different fitness functions were used. The first one attempts to maximize the rate of return. This fitness function was essentially just the sum of the total amount of money gained or lost over n amount of games. Usually it would be 30,000 games, however if a net achieves the highest fitness result then it will be run again and the average between the two results will be the final fitness. This function ended up not working too well compared to the second one. The second one maximized the number of games won which also worked better to maximize the rate of return as well. However, a natural (blackjack) does not count towards the number of wins since it is a random occurrence and not one caused by the player. A net can also completely fail and have its fitness equal to zero. This would be if the net has gone through twenty percent of the number of games to be played and has not won more than a fifth of them. After repeated training sessions it was found that nets who did not meet this bottom line only stagnated and soon became extinct.

3.3 Results

The best evolved net came from the second fitness function where I start off with a fully connected net with all input nodes connected to the hidden layer and all those connected to the output. The results came close to the theoretical maxima, falling short on the rate of return since it was not able to learn a few things such as surrendering.

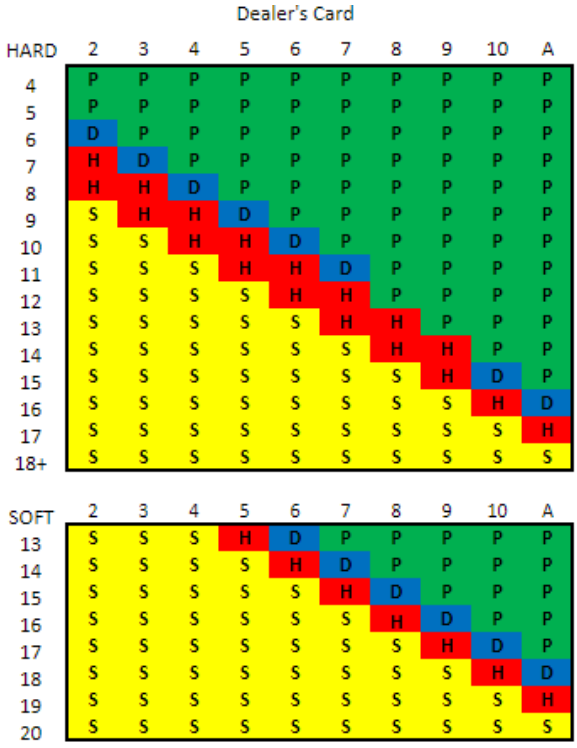

Figure 5. Strategy chart from the most fit genome. H(hit), S(Stand), D(Double), P (Split if possible otherwise hit).

Results(Percent)	
Wins	42.68
Dealer Wins	49.54
Push	7.77
Surrendered	0.00

Table 2. Game Results from a simulation of one million games using the most fit genome.

While comparing the basic strategy and NEAT results, the net was unable to learn one important thing which was the ability to

surrender (see Figure 5). Surrendering in games where it is highly probable the player will lose, results in around a 4% decrease in games won for the dealer. While the NEAT simulation resulted in 49.54% dealer wins, the basic strategy results had a smaller 45.06% (see Table 2). Even though the net was able to reach the theoretical maximum win rate it was not able to maximize the rate of return coming out to 90.8%. Interestingly the rate of return was higher on the second fitness function versus the first which emphasized a higher return.

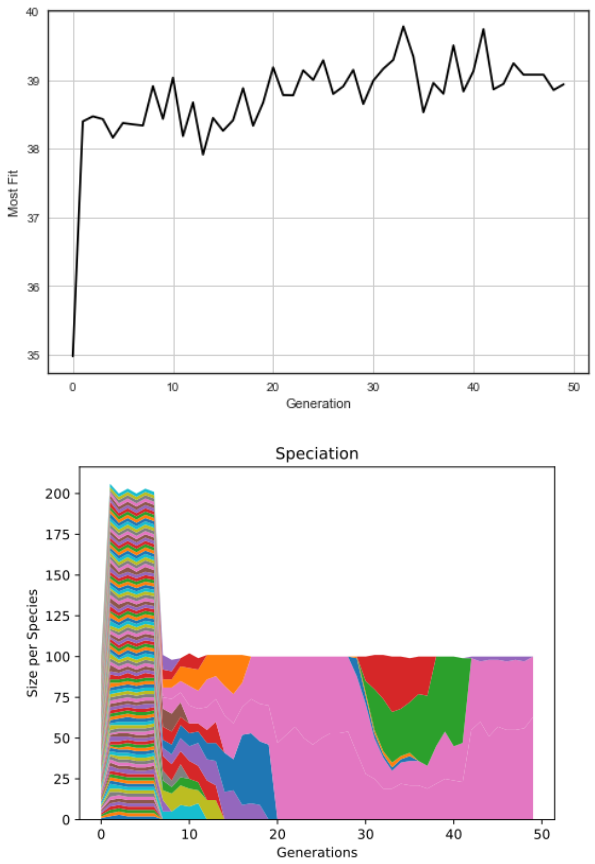

Figure 6. Evolution of fitness and species using NEAT's visualize function.

The net was quickly able to evolve within 5 generations achieving a high fitness right off the bat, which could simply be an anomaly since most of the other simulations took around 20 generations to reach this point.

4. Conclusion

In this paper, I run simulations in blackjack using predetermined strategy and compare it to the results from an ANN trained with NEAT. Although NEAT was able to formulate a good solution relatively quickly, it was not an optimal one and could have used a small amount of improvement. Some changes in the ANN's configuration or possibly a change in the fitness function might be able to achieve this goal in the future. Another method which may work better in this case would be Q-learning which is also a type of reinforcement learning.

References

[1]Stanley, Kenneth O., and Risto Miikkulainen. "Evolving Neural Networks through Augmenting Topologies." *Efficient Evolution of Neural Network Topologies*, 2002, pp. 99–127.

[2]Snyder, Arnold. "History of Blackjack: Julian Braun, Blackjack Pioneer." *History of Blackjack: An Interview with Julian Braun*, 2005, www.blackjackforumonline.com/content/brauninterview.htm.

[3]Shackleford, Michael. "Wizard of Odds." *Wizard Of Odds*, wizardofodds.com/.

[4]"NEAT Overview." *NEAT*, CodeReclaimers, 2015, neat-python.readthedocs.io/en/latest/neat_overview.html.

[5]"Blackjack House Edge Calculator." *Blackjack House Edge & Standard Deviation Calculator*, Beatingbonuses.com, 2006, www.beatingbonuses.com/houseedge.htm.

[6]Lehman, Joel, and Risto Miikkulainen. "Neuroevolution." *Scholarpedia*, 2013, www.scholarpedia.org/article/Neuroevolution.

[7]Winston, Patrick Henry. "Artificial Intelligence." *MIT OpenCourseWare*, 2010, ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/.