Backpropagation Blackjack

Michael Alsbergas
5104112
ma11xr@brocku.ca

ABSTRACT
This paper investigates the effectiveness and efficiency of training an artificial neural network to play the card game blackjack using basic backpropagation. Its decision making skills are tested by making a choice based on the known state of the game's board and are then evaluated based on the number of games it can win.

1 INTRODUCTION

Most games cannot be played, let alone mastered, with sheer technique and skill. Many games involve an element of randomness or uncertainty, particularly card games. It is because of this element of randomness that no perfect strategy can be developed.

Many AI's that are taught to play games often use a game tree to search all possible moves to find the next optimal move. This method, however, can be computationally expensive and very consuming in physical memory. For example; the game of chess has an average branching factor of 35, meaning that there are on average 35 different possible moves a player can make on their turn. Yet such a large tree can be pruned and reduced by adding in additional algorithms such as alpha-beta pruning.

In the case of card games and other games of chance, such reductions are simply not possible. In the case of chess, a decent AI would create a tree with a depth of 4, and using alpha-beta pruning could reduce the overall size of the tree by almost half! Yet, to find the optimal decision to make in a game of blackjack, 2 trees would need to be created, each with a branching factor of 13 and a depth ranging from 2 to 7, possibly more depending on the specific set of rules being used to play! While the size of a game tree for chess may still be larger than a tree for blackjack, it still requires a significant amount of computational power to perform. The worst part is, even if every optimal decision is made, the player can still lose due to chance!

That is where the advantage of a neural network comes into play. Neural Networks are ideal for making decisions and/or categorizing certain states with limited information given. They can also be very robust or adaptive to noise or unwanted data within their training data. This should make them very effective when making decisions with games of chance where limited information is given and there's often no definitive correct answer.

2 PROBLEM DEFINITON & BACKGROUND

2.1 Rules of Blackjack

Blackjack is a 2 player game between the actual player and the dealer. The goal of the game is to get a hand with a value that's closer to 21 than your opponent without going over. If the value of a hand is over 21, that player will automatically lose the game so long as their opponent's hand's value is less than or equal to 21. A hand exceeding the value of 21 is known as a *bust*.

The value of cards is based on their face regardless of their suit. For cards 2 to 10, their value to a hand is their face value. For example, a hand with only a 7 and an 8 in it will have a value of 15. Jacks, queens, and kings each also have a value of 10. The ace can have the value of either an 11 or a 1 depending on the value of a hand. A hand that contains an ace using a value of 11 that is still below or equal to 21 is known as a *soft hand*. However, if using the ace with a value of 11 would cause the value of the hand to exceed 21, the ace must be used with a value of 1. This is known as a *hard hand*. If a player has a hand that contains only a jack, queen, or king and a single ace, this hand is known as *blackjack* and the player automatically wins regardless of their opponent's hand and receives a higher payout in the case of gambling.

The game begins with the dealer dealing 2 face up cards to the player and 2 cards to themselves, one face up and the other face down. Of course, the player then only knows the values of his/her own hand and the value of the dealer's one face up card. With only those pieces of information, the player must make a decision; should they *hit* and take another card or *stay* and end their turn there. If the player hits, they are dealt another card to add to their total and the same decision must be made again. If the player chooses to stay, they're keeping the hand with the value they currently have and end their turn. It then becomes the dealers turn where they must repeat the same process of hitting or staying. However, most dealers are bound by a heuristic typically along the lines of hit on a 16 or lower, and stay on anything higher.

However, after the 4 initial cards are dealt at the start of a game, the player has 2 other choices in addition to hit and stay. The additional choices are *split* and *double down*. These choices were not be included in the experiments for multiple reasons. These reasons include simplicity in programming, that they'd only influence the amount of money earned/lost not the number of games won, and that they're unique decisions that would need separate networks of their own.

2.2 Problem Definition

As described earlier, finding the optimal decision to a given state in blackjack is a problem that has a complexity of 13^n. The n, in this case, is the total number of cards that both players could have at the end of game, excluding the 2 revealed cards of the player and the 1 revealed card of the dealer. The maximum n possible varies depending on which set of rules is used. American rules state that a player wins if they manage to reach 5 cards in their hand without busting, thus putting the maximum n at 7. The UK set of rules does not have that 5 card limit, allowing the maximum hand size to be 15 without busting. Thus, a decision tree using UK rules could have a maximum n of 27! While still being possible to compute using modern day computers, it is still an expensive operation to perform.

The next problem is that blackjack is a game of chance and uncertainty, thus rendering almost all in-depth strategies inadequate. Even Kendall and Smith [1] claim that Thorpe's strategy is the best and still does not have a 100% win rate. (I believe it should have a lower rate than stated after trying it for myself.) Ruvolo and Spritke claim that the dealer's strategy has a 90% win rate. This shows that a fairly simple heuristic can be used to play effectively, yet still have no perfect strategy.

So, the goal is ultimately develop a strategy that performs better than a heuristic yet can make a decision faster than the large computational time of finding the optimal solution.

3 METHODOLGY

A neural network fits this task because its robustness to noise allows it to deal with the large elements of uncertainty in training. Once training is complete, it is very quick and efficient to use and does not need to repeat expensive calculations or use a look-up table.

Most similar papers reviewed used reinforcement learning where a specific state and action were trained based on the outcome of the game. The result ended up being more of a well developed look-up table with odds of winning using a given board state and action rather than a well trained network. Yet the equations are interesting as they calculate the odds of the previous states and the previous actions, not just the final decision as is being done here.

The structure of the network used is 3 hidden nodes in 1 hidden layer and a single output node. While most reviewed papers used a large number of input nodes, such as 16 (one for each possible card in hand and the dealer's up card), or 28 (18 for each possible value of the player's hand and 10 for each possible value of the dealer's up card). The 3 inputs used are the value of the dealer's up card, the current total of the player's hand, and whether or not the player's hand is a soft or hard hand. Using the smaller number of inputs and the smaller network size should make it faster and easier to converge as there will be fewer weights to optimize.

The training examples will be randomly generated games using a seed value. After each game, the number generator will be reset with a new seed value; this is done for a number of reasons. The first reason is to make each game

repeatable as if they're using the same deck shuffled the same way each epoch. The second is to make each game independent from each other, so the actions from the previous game will not affect the deck next game.

For each game, the initial 4 cards will be dealt to each player. Then, the information is passed forward through the network with the output being to hit or stay. The output node should fire 0 to hit and 1 to stay. If the network decides to hit, the information will be updated with a new card added to the network's hand and then passed through the network again. This process is repeated until either the network decides to stay or busts.

Once the network's turn is done, the dealer's turn is done following the heuristic of hitting on any value below 17. This heuristic was used instead of others for simplicity and realism as well as being easier for the network to train on.

After a winner is determined, the network or the dealer, the network is trained based on the last decision it made and whether it won. If it won, the last decision it made will be reinforced. If it lost, the last decision made will have a large error as the other decision will be reinforced. The last decision will be reinforced rather than having all previous decisions reinforced because there's no real way to tell if those choices were optimal or not.

Simple backpropagation using a gradient sum is used to help account for the large amount of noise in the training examples. Such a source of noise is the existence of unwinnable games. From the perspective of a player, the game is non-deterministic with its random elements. Yet, from a deterministic, all-knowing point of view, there a games that can never be won by the player. For example; the player has a hand of 12 against an up-card of 10. The optimal decision would be to hit, but the next card on the deck is another 10, so the hit would result in a bust. So, the network would be trained to stay. Yet, the dealer's face down card is a 7, so when the network decides to stay, it will still lose. No matter what, the network will be trained to lose in that situation.

With this method simulating real games, the network should train fairly efficiently. The expectation is that the network will constantly hit till they bust and will learn to stay. Then, it will constantly stay on the first hand it gets, where it will begin to lose with low value hands. From there, it will learn to hit on low values and stay on high values, and continue learning the game from there.

There will be 500 randomly generated games played each epoch. Each epoch, 10 of these games will be used for training and updating the network. There will then be 500 epochs, so at the end of the 500 epochs, each game will be trained on almost 10 times. This process of 500 epochs will be repeated 50 times, thus the network will be trained and validated on 12,725,450 games!

4 EXPERIMENTS & DESIGN ISSUES

Initially, once the creation of the network was complete, it was trained using the structure and training described above. However, the results showed that the network was not learning at all. In fact, it was converging after the first epoch. A variety of modifications where made, such as using 4 hidden nodes instead of 3, or changing the dealer's heuristic. I even changed the activation function and the outputs of the network. Still, the network would immediately converge after the first epoch, but the overall performance did increase as these alterations were made. Originally, the network was not going to use a summed error when updating, it was at this point that that aspect of the designed was changed in hopes of promoting proper learning.

Upon further investigation of the output, it was found that the network was either constantly hitting till bust or constantly staying regardless of the value of the hand. Its decision was also being constantly reinforced despite facing a large error in the opposite direction. Then a run was performed with momentum removed from the update equation. The results were that it learned but not very quickly and failed to converge. Upon further investigation, it was realized that the momentum equation was faulty as after a number of increases to the value, the value would no longer decrease.

The momentum equations were then changed, but still did not converge as desired. They were changed again to formulas a lot simpler than the previous ones. After that change, the results were then considered adequate to demonstrate learning happening in the network.

With the network performing as intended, the final experiment was run with an initial learning rate of 0.1 that decreases after each epoch, and an initial momentum of 0.1 as well. The rest of the experiment was performed as described above.

5 RESULTS & ANALYSIS

Over all, the results were rather lack luster and somewhat disappointing. After 25,450 epochs, the best result was 227 wins out of the 500 games played. That's a win rate of 45.4%, which is almost half that of the average casino player [1].

While an in-depth examination of all the decisions the network made over these games is extremely difficult, simply due to the overwhelming number of them, a cursory examination has been made. This examination showed that the network was performing somewhat as expected but slower and over more cycles. That is, it did show that it would constantly hit for a time, and then constantly stay for a time, as predicted. However, it would repeat this flip-flop several times over rather than learning right away. Of course, ideally the decisions should be about 1-3 hits followed by a stay. Fortunately, this pattern does begin to emerge later on in the experiment. The emergence of this pattern does help demonstrate that the network is in fact learning, just at an incredibly slow rate.

While Figure A is rather condensed due to the large number of epochs used, the results are still rather visible. It appears that the number of wins begins as being largely erratic with occasional spikes in performance and frequent dives as well. Yet, as the number of epochs continues on and the training continues, the number of won games tends to stick closer to the higher values and the dives in performance become less frequent and less intense. This shows that the network demonstrates some level of learning, at least at when learning what not to do.

There appeared to be a faint upward increase in the average number of wins over time. This increase is really difficult to see in Figure A, so it may not actually exist. It may just appear that way as more values cling to the upper bound because less downward dives occur.

To help visualize this possible increasing curve, Figure B was created. Figure B shows the average outcomes of every 1000 epochs. This better represents trends in the data that Figure A displays.

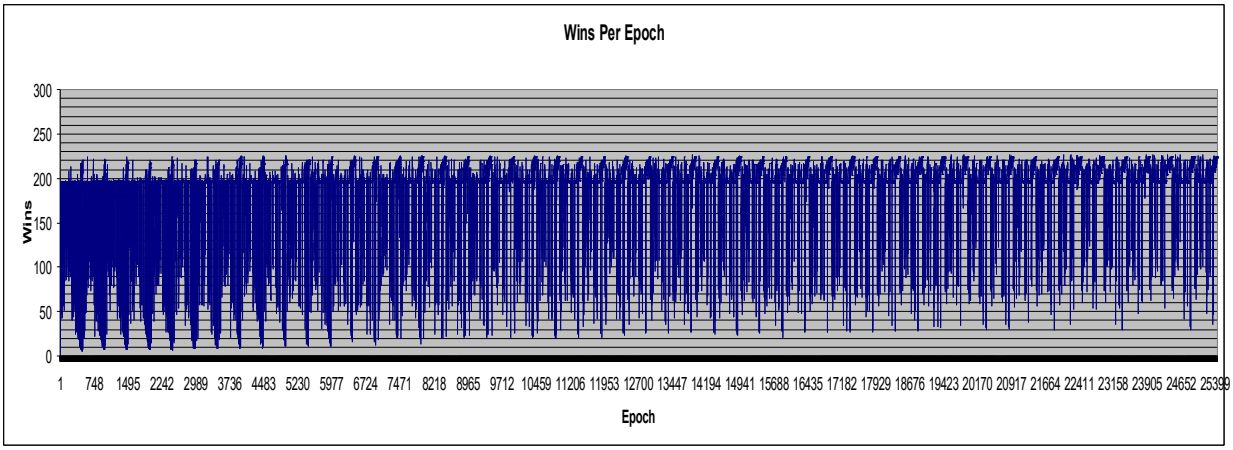

Figure A: Results of the Network over 25,450 Epochs

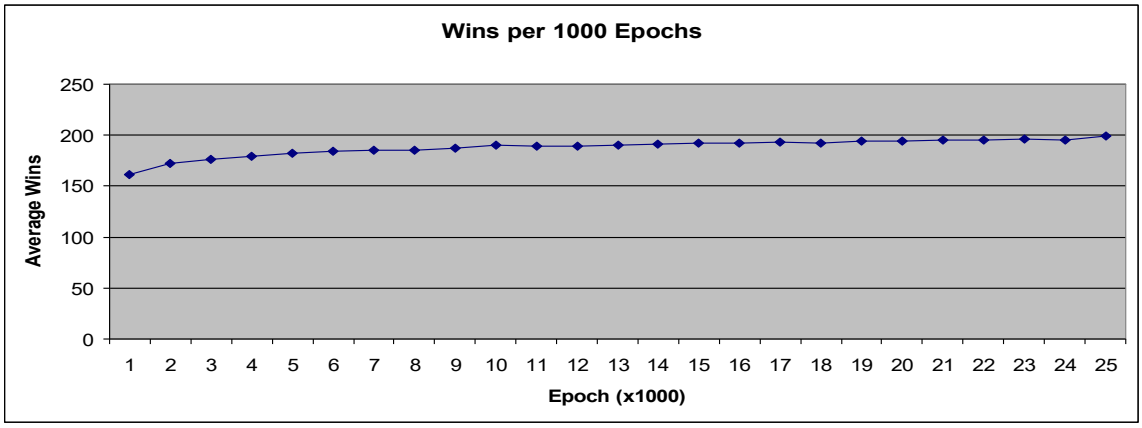

Figure B: Averages of Wins of 1000 Epochs

As visible in Figure B, the average number of won games is in fact increase as training progresses. However, this can mean one of two things. It could be showing that the overall performance and efficiency is increasing as training continues, or it could just mean that there are less dives and less low values are occurring as training continues.

Either way, it still shows that in at least some manner, the network is definitely learning how to play the game. At the very least, it demonstrates that the network could be converging. However, Figure B alone cannot support that claim on its own because after over 25,000 epochs it still hasn't converged! Which begs the question if it ever will, at least will it converge in less time than it would take to calculate an optimal decision? Personally, I hope it is not converging because that would mean that it is converging to a really low quality win rate.

6 CONCLUSIONS & FUTURE WORK

6.1 Conclusions

Given the immense amount of training provided to the network and the low level of performance in return, a neural network trained using basic back propagation is not an efficient or effective means of playing blackjack. However, this does not mean that this is overall a bad approach to take. The network still demonstrated progress towards learning the game and further improvements to the network and aspects of it could yield better performance.

6.2 Future Work

Further experiments on the network could be using a more efficient learning algorithm such as r-prop or quick-prop to help aid in convergence of the network weights. This type of adjustment is recommended over simply increasing the number of hands and epochs used.

Changes in the structure such as adding more hidden nodes or using 2 output nodes instead of 1 may yield better performance. Ruvolo and Spritke suggested in their future work using a second hidden layer in the network, but they were not very optimistic about the outcome of that change [2]. Neither am I as only 3 inputs were used here compared to their 28.

The next recommendation for further experimentation is changing the training data. Rather than using randomly generated hands to train the network on, perhaps it could train individual hands reinforcing on Thorpe's strategy rather than winning or losing games. A comparative study could then be done on the effectiveness of learning and executing Thorpe's strategy compared to having a network learn using more realistic examples.

(In fact, having students decide on their own set of training examples could make them more interested and invested in the outcome of their program)

7 REFERENCES

[1]G. Kendall and C. Smith, "The Evolution of Blackjack Strategies".

[2]P. Ruvolo and C. Spritke, "The Pentium Goes To Vegas".

[3]R. Vikramaditya, "Blackjack Game: Artificial Neural Network Implementation of the Blackjack Dealer with a Reward mechanism.". [Online]. Available: http://eecs.wsu.edu/~vjakkula/BlackJackProjectReport.pdf. [Accessed: 29- Apr- 2016].

[4]R. Grayson, "Roger Grayson's ePortfolio - Neural Network BlackJack", *Efolio.digitalgrayson.com*, 2016. [Online]. Available: http://efolio.digitalgrayson.com/nnblackjack.php. [Accessed: 29- Apr- 2016].