Teaching a Neural Network How to Play Blackjack

Jeremy David & Kenny Groszman
Contribution: 50-50, respectively

Neural Machine Learning 502
Dr. Erszebet Merenyi

Statement of Problem:

Blackjack, sometimes called twenty-one, is the world's most popular casino game. It is also commonly cited as the casino game where the player has the best odds of winning, with the most widely accepted "optimal strategy" giving the player 49-51 odds of winning in the long-run. In this project, we teach two backpropagation neural networks how to play blackjack and see what strategies it suggests to maximize profitability. In the process of doing this, we observe how the backpropagation networks reacts to stochastic data, in which identical inputs can map to different outputs. We also look for strategies that mitigate problems that may arise as a result of this difficulty. After optimizing our backpropagation neural networks, we will simulate the performance of its decision-making strategies against the published blackjack optimal strategies, and hopefully use our network to generate simulated profits.

Description of Data:

We generated our data by simulating rounds of blackjack and picking out the optimal strategy. In order to understand this simulation, it is necessary to understand a bit about blackjack and the structure of our network.

Blackjack Background:

The objective of blackjack is to beat the dealer by drawing cards in order to increase the score of one's hand without exceeding 21. A score is calculated as the sum of the values of the cards in one's hand, where face cards are worth 10 and aces are worth the player's choice of either 1 (a hard sum) or 11 (a soft sum). The dealer also has the opportunity to draw cards, but they must follow the pre-specified rules below:

1. The dealer must hit if their sum is less than or equal to 16
2. The dealer must also hit on a sum of 17, provided that it is a soft sum

When playing blackjack, a player has two decision points:

1. At the beginning of the hand (when the player has two cards, and can only see a single one of the dealer's cards), a player must make a decision whether to double down, split, or stand. We term this the "betting decision".
    - *Doubling down* means the player doubles his bet, and commits to drawing a single card from the deck.

    - *Splitting* means the player splits his two cards into two separate hands and draws a new second card for each hand. The player can then play those two hands separately. Splitting is only allowed if the player's two original cards have the same value.
    - *Standing* means the players retains their original bet and plays the hand as usual, with an unlimited number of possible hits.
2. Following the betting decision, the player must make a decision whether to hit or stay. Should the player decide to hit, he will draw another card and make this decision again until he eventually either busts (exceeds 21) or decides to stay.
    - *Hitting* means the player draws another card and updates their sum
    - *Standing* means the player is satisfied with their hand

The game's structure lends itself for a two network approach, which we describe in figure 1 below.

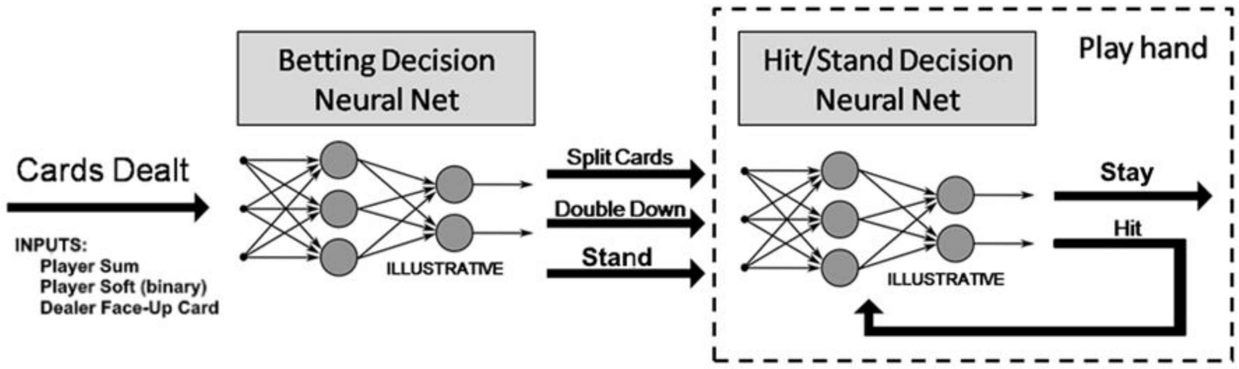

Figure 1: The two-decision structure of blackjack lends itself to a modeling scheme with two separate neural networks.

Simulation of Blackjack for Data Generation:

Each of the two decision-making neural networks requires a data set for training and testing. Here, we briefly describe the simulation process for each entire hand of blackjack, and the logic used to decide on the optimal strategy for each neural network.

We use MATLAB to create a vector of cards corresponding to the 6 decks in a standard casino blackjack game. We then randomly permute the vector entries to simulate shuffling. Cards are "drawn" from this deck as needed in order to simulate a round of the game. The dealer's strategy is simulated according to common casino rules (hit anything less than 16 or a soft 17).

Then, the player's strategies are simulated. For the split/double-down/stand (SDS) network, all three strategies are simulated as hypotheticals, and the winner is chosen as the strategy with the highest payout. In the case of a tie in payout, the strategy with the lowest risk is chosen. For the hit/stay (HS) decision network, the strategy is chosen that beats the dealer while taking the lowest amount of risk. If no strategy beats the dealer, we choose the strategy that maximizes score without busting.

Training and testing datasets were generated by following this 20000 times. The table below outlines technical information about data generation.

	Split/Double-Down/Stand Network	**Hit/Stay Network**
Origin of Data	Generated using blackjack simulation script, as explained above	
Date Meaning	INPUTS: 1. Player's hand sum 2. Sum status (soft/hard) 3. Dealer's up card 4. Splittable, describes if player is allowed to split hand OUTPUTS: 3x1 unit vector encoding optimal strategy (split, double-down, or stand)	INPUTS: 1. Player's hand sum 2. Sum status (soft/hard), binary 3. Dealer's up card OUTPUTS: 2x1 unit vector encoding optimal strategy (hit, stay)
# of Data Points	20,000	20,000
Input Feature Encoding and Scaling	1. Player hand sum encoded as an integer 2. Sum status encoded as a 0-1 binary variable 3. Dealer's up-card encoded as an integer 1-10, representing the card's value 4. Splitability encoded as a 0-1 binary variable Variables 1, 2, and 4 were scaled linearly to [-0.9, 0.9] Variable 3 was scaled according to the following mapping, so that the Ace would be treated differently from the other cards: [1:10] → [-0.9, (-0.5 : 0.175 : 0.9)]	1. Player hand sum encoded as an integer 2. Sum status encoded as a 0-1 binary variable 3. Dealer's up-card encoded as an integer 1-10, representing the card's value Variables 1 and 2 were scaled linearly to [-0.9, 0.9] Variable 3 was scaled according to the following mapping, so that the Ace would be treated differently from the other cards: [1:10] → [-0.9, (-0.5 : 0.175 : 0.9)]
Output Class Encoding	Each network's corresponding output strategies are encoded as classes in a one-in-n representation	

<u>Objectives</u>:

Our objective is to create a neural network that maximizes profits playing blackjack. What we would like to get out of the network is as follows (in order of importance):

1. Train two neural networks corresponding to the two blackjack decision points in order to profitably play blackjack
    a. Train a neural network to make hit/stay decisions
    b. Train a neural network to make split/double down/stand decisions
2. Test the behavior of a BP neural network on stochastic, contradicting training dataset

<u>Technical Approach:</u>

We utilized MATLAB's *Neural Networks Toolbox* in order to generate, train, and test both neural networks. In both cases, we mostly chose MATLAB's default parameters for network training, which are chosen according to the Levenberg-Marquardt optimization scheme. A brief overview of these parameters is in the table below. While we were unable to change the default MSE error function, we calculate classification accuracy at the end of training to evaluate network performance.

Levenberg-Marquardt Optimization Scheme Parameters:

Transfer Function	*tanh(.)* with slope = 1
Initial Weights	Initialized by MATLAB according to Nguyen-Widrow initialization method
Learning parameter (μ)	According to MATLAB 'trainlm' function: Initial μ = 0.001 μ decrease factor = 0.1 μ increase factor = 10 Maximum μ = 1e10
Epoch Size (K)	1 (on-line training)
Maximum Learn Steps	1000 cycles through all training samples
Error Measurement Method	For training, MATLAB default is used: Mean Squared Error (MSE) For performance evaluation, classification accuracy is used
Stopping Criteria	MATLAB default: 6 consecutive validation failures
Monitoring frequency of error	Every 20,000 learn steps

Split/Double-Down/Stand Network:

The split/double-down/stand network was a two-layer backpropagation network with the following topology:

4 (+1) - 20 (+1) - 3

The network was trained using MATLAB's 'trainlm' scheme, yielding the learning history in **Figure 2**. Note the x-axis of the graph, whose numbering indicated the number of cycles through all training data.

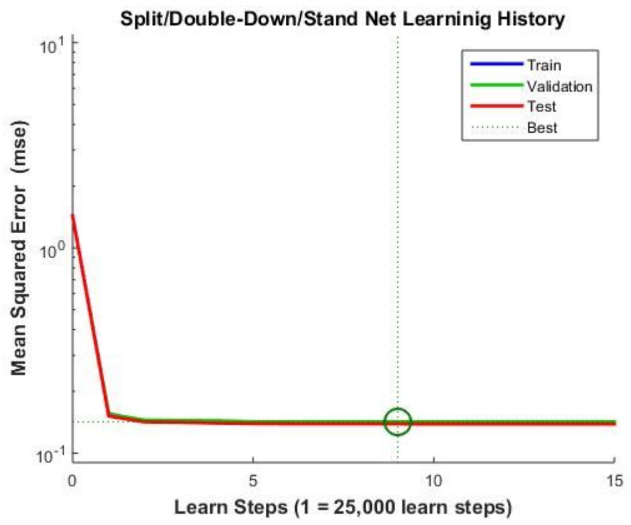

Figure 2: Split/Double Down/Stand Network Learning History

Hit/Stay Network:

The hit/stay network was a four-layer backpropagation network. We increased the number of layers due to subpar results with fewer layers. The network had the following topology:

3 (+1) - 20 (+1) - 10 (+1) - 5(+1) - 2

The network was trained using MATLAB's 'trainlm' scheme, yielding the learning history in **Figure 3**. Note the x-axis of the graph, whose numbering indicated the number of cycles through all training data.

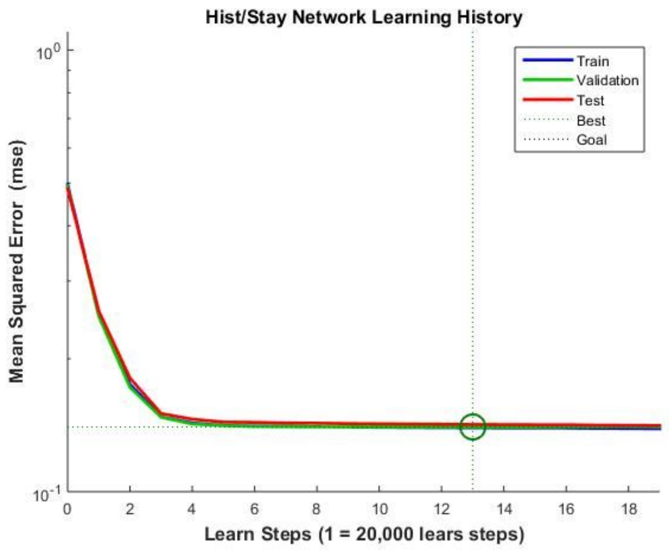

Figure 3: Hit/Stay Learning History

<u>Results:</u>

Performance metrics:

Two metrics were used in order to measure network performance. First, the percentage of situations that the network plays in accordance with the analytically calculated optimal blackjack strategy. This metric is only in reference to the hit/stay decisions, so only applies to the hit/stay network. This was calculated by forming an input set that contained one of every possible blackjack situation (468 total). The output was then extracted from the network, and compared to the accepted optimal blackjack strategy. The split/double-down/stand network was not analyzed further, as it performed adequately when compared to the optimal strategy. The second performance metric was profitability when both networks were used to play blackjack. This is the ultimate performance metric, as it tests how well both networks work in conjunction function to accomplish our ultimate goal. Profitability was taken by simulating 10,000 rounds of blackjack using the networks to make decisions, and then compared with the profitability of the optimal strategy and of a network with untrained weights.

Initial results using "confusing data":

The network was initially able to make decisions in accordance with the optimal blackjack strategy in 80% of situations. This was not quite good enough to be profitable.

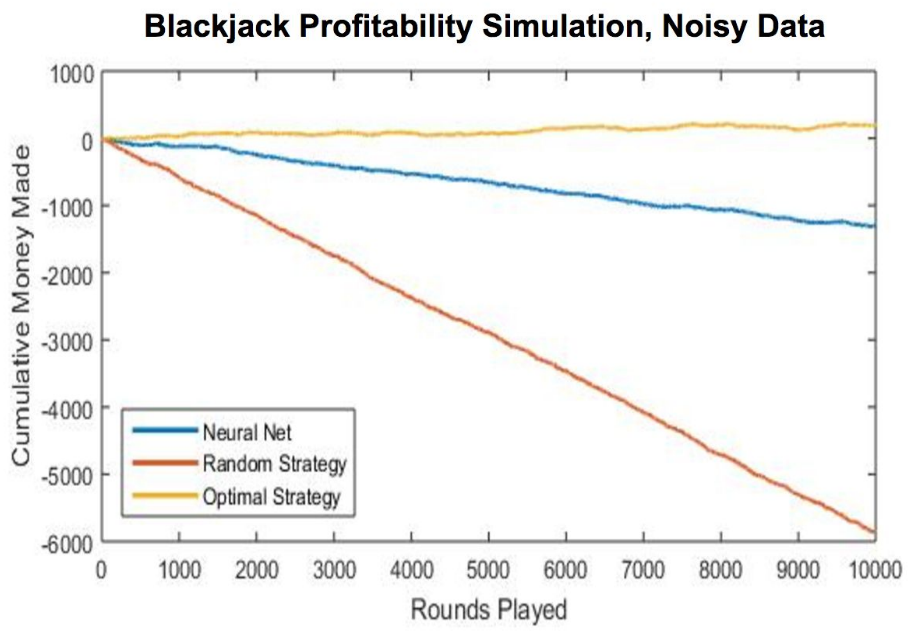

Figure 4: Initial Profitability Results

As seen in **Figure 4**, the network was able to distinctly outperform a random initialization of weights; however, was not able to make money and underperformed when compared to the optimal strategy.

The primary reason for this was that the data set used for training (20,000 simulated rounds), had many of the same inputs that called for different classes as output. In other words, the network was sometimes trained to hit with one input, and at other times trained to stay with the same input. The reason for this is the way the data was generated. Simulating a round by drawing a card and testing whether it was a good or bad decision to draw that card depends on the card that was drawn. So, there are times where the simulation will get “lucky” and draw a good card in a situation that most often should not have drawn a card. The histogram below depicts the fraction of the time the same three input values (2 of the inputs being 6 and -0.9, unscaled, and the third input being 15, 16 or 17, unscaled) had hit as their desired output.

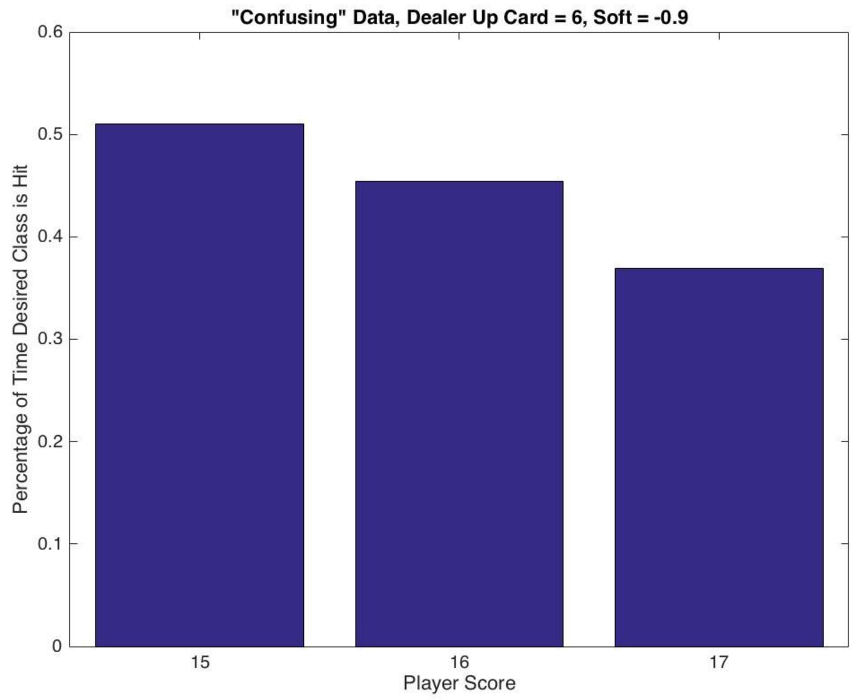

Figure 5: Noisy Data Histogram

As can be seen in **Figure 5**, there are several inputs for which the desired output is different almost half the time. The BP network is thus not able to converge to the optimal strategy for many of these "borderline" inputs.

Our solution to this was to thin out the training inputs so that they contained only one of each possible blackjack situation, with the desired output being what is optimal most of the time. For example, in the histogram above, the input 6, -0.9, 15 would have hit as the desired output, as that is the best strategy above 50% of the time. This network was trained with the same parameters as the previous hit/stay network.

Final results using thinned out data:

After reducing the training data set, we were able to achieve the optimal strategy in 96% of situations. This was considerably better than the 80% achieved with the contradicting data set. As can be seen below, the networks can now play blackjack on-par with the optimal strategy, and is profitable over 10,000 hands.

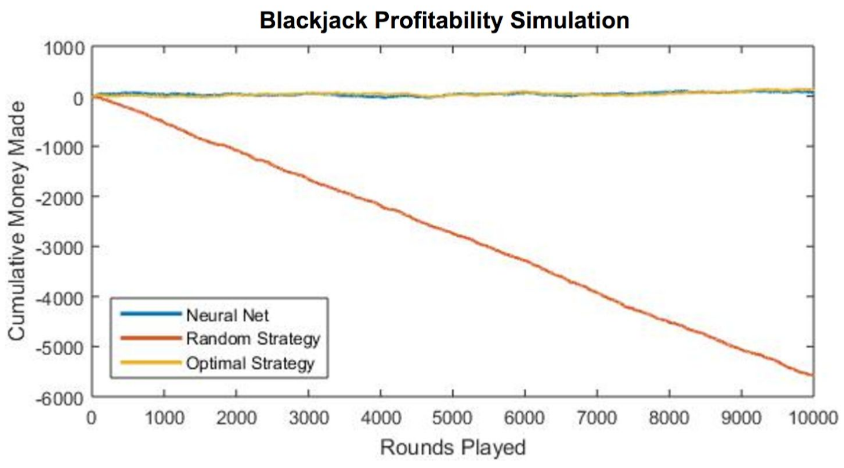

Figure 6: Final profitability Results

Conclusion:

We have found that BP Neural Networks are not a good choice in situations where the data is stochastic or contradicting. However, after noise removal, the BP paradigm is able to learn how to play blackjack optimally. Other network paradigms, such as probabilistic networks or reinforcement networks, may be better fit for this application. In conclusion, by training two neural networks corresponding to the two major blackjack decision points, a BP ANN can be taught how to profitably play blackjack.