**Using a Neural Network to Optimize Blackjack Profitability**

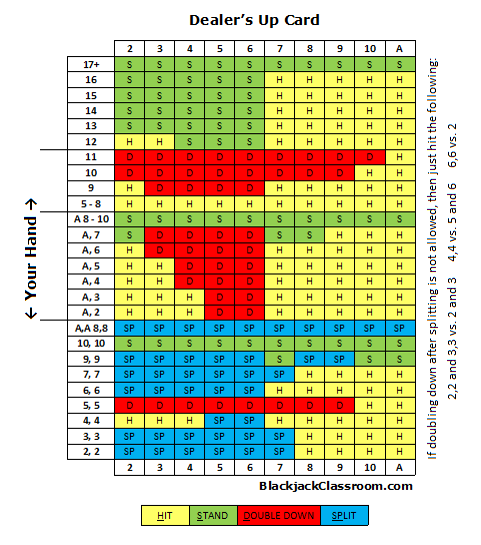
NML 502 Final Project Proposal

Jeremy David, Ken Groszman

3/20/2017

Statement of Problem:

Blackjack, sometimes called twenty-one, is the world’s most popular casino game [1- Scarne Complete Guide to Gambling]. 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” (below) giving the player 49-51 odds of winning in the long-run. In this project, we will teach an artificial neural network how to play blackjack and see what strategy it suggests to maximize profitability in different blackjack situations (varying the number of decks, number of players at the table, etc.). As an extension, we will see if we can simulate teaching our network how to “count cards”, a strategy used by professional gamblers to increase their probability of winning. We hope that through this project, we will either confirm or debunk the common blackjack “optimal strategy” and gauge the effect of a variety of factors on a player’s profitability in a blackjack game.



**Figure 1:** Simple optimal blackjack strategy [2- DroidPoker.com]

**Data** **Description**:

In order to accomplish the goals of this project, we will generate training and testing data simulating the results of real hands of blackjack in different situations, including:

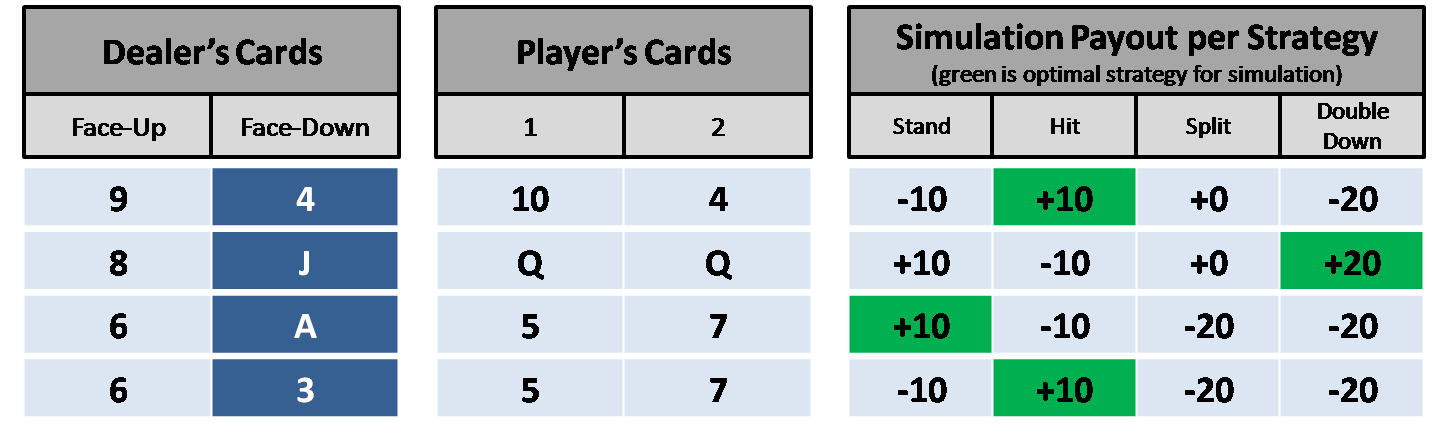
* The number of decks being used to play
* The number of players at the table (and by extension, the number of cards visible to the player)
* Different rules of play (blackjack pays out 3-2 vs. 6-5)
* Multiple hands played from the same set of decks (i.e. the ability to “count cards”)

*Generating Data*:

In summary, we will generate testing and training data by simulating entire hands of blackjack and selecting what the optimal strategy would be for the player in order to maximize profits.

We will use MATLAB to create a vector of cards corresponding to the number of decks which will then be randomly permuted to simulate shuffling. Cards will be “drawn” from this deck as needed in order to simulate a round of the game. The dealer’s strategy will be simulated, according to common casino rules (hit anything less than 16 or a soft 17). We will then simulate all of the player’s different possible strategies (hit, stand, split, double down) to reveal which rational strategy will give the best payout.

An example of this sort of simulation for one player is below. Green boxes indicate the optimal strategy for the simulated. Notice that the simulation is stochastic, so the same input situation will not always give the same optimal payout strategy (as in rows 3 and 4 below). We can train over an arbitrarily large number of simulations, over which some strategy should prevail as optimal for that situation.



**Figure 2:** Four examples of simulated blackjack hands, with the green box indicating what would be the optimal strategy.

To evaluate the network, we will measure the net “payout” after a certain number of simulated hands. This makes the results of our network consequential and transferrable to a real life situation.

*Data Description Technical Details:*

|  |  |
| --- | --- |
| **Origin of Data** | Generated using blackjack simulation script, as explained above |
| **Description of Data Meaning** | ANN INPUT:  A vector representing the cards that a player can see during a blackjack game (his own two cards, the dealer’s face-up card, and the two cards of all the other players at the table)  ANN DESIRED OUTPUT:  A one-in-n output vector representing the player’s optimal strategy for that simulated hand. An optimal strategy is defined as the least risky logical strategy that beats the dealer. |
| **# of Data Points** | We can generate an arbitrary number of training and testing data points. Given the large number of permutations of possible draws from a deck of cards, we hope to start with 1,000,000 training points and increase it as needed. |
| **Input Features and Encoding** | The number of input dimensions will vary based on the situation:  3 FIXED INPUTS:   * Two features for the player’s two cards * One feature for the dealer’s face-up card   VARIABLE INPUTS:   * A variable number of features for the other cards visible to the player, based on the number of players at the table and cards played in past hands (for “counting cards”   Encoding:  Each card {A, 2, ..., 10, J, Q, K} encoded as a corresponding integer {1, …, 13} |
| **Output Classes and Encoding** | Optimal strategies for a particular hand:   * Stand (hit zero times) * Hit once * Hit twice * Hit three times   In more advanced versions, we hope to include:   * Split * Double Down   Encoding:  Each of these strategies will be encoded as a corresponding unit vector |
| **Output Class Variability** | Based on 10,000 hands simulated for one player and four decks:   * Stand: 65.4% * Hit once: 25.8% * Hit twice: 7.2% * Hit three times: 1.5% |

**Objectives:**

Our objective is to create a neural network that maximizes profits playing blackjack. In other words, our goal is to create a trained network that can win more than 50% of blackjack games. What we would like to get out of the network is as follows (in order of importance)

1. Given the dealer’s and player’s cards, make optimal hit/stay decisions
2. When given the dealer’s card, our own cards, and several previous cards, to be able to make even better hit/stay decisions (i.e. “Count Cards”)
3. When given the dealers card, our own cards, and several pervious cards, be able to make optimal hit/stay decisions, as well as split/double down decisions

Ultimately we would like to be able to come close to recreating, or surpassing, the current “optimal strategy”. As a secondary goal, it will be interesting to determine which blackjack playing conditions maximize payout for the player.

**Technical Approach:**

Our approach will be a supervised, multilayer, fully connected, feed-forward perceptron, using back propagation as our learning rule. The “Desired output” will come from simulating a blackjack deck and determining what the optimal hit/stay decision would be. Output layer dimensionality will be determined by the number of possible strategies (stand, hit once, hit twice, etc.) and will be encoded as unit vectors.

In order to add the “counting cards” feature (objective #2), we would take one of two approaches. We would either include the “card count” as an input PE based on which cards have been removed from the deck, or the N previous cards as N more input PEs.

In order to determine success, we will see what percentage of games the network wins, and its net payout. After some number of learning steps, a recall will be done, simulating multiple blackjack games using the decision that is outputted from the net. Profit will then be calculated based on consistent bets of $100.

Learning parameters such as the learning rate, momentum and hidden PEs will be tweaked and optimized.

The simulation of the blackjack game to determine the desired output will comprise of a substantial component of the project. We plan on generating n decks of cards, with cards represented as integers 1-13 in a vector. We will shuffle the deck by permuting the entries in the vector randomly. The first entries in the deck will then be “dealt” to the dealer and the players (number of players will be variable). These inputs will be passed to the neural net, which will decide on a strategy. This will be compared to the desired strategy from the simulation, and weights will update accordingly via back-propagation.

This process will have two separate steps. The first step will be generating the training data by simulating many games of blackjack and the second will be running this data through the neural net.