HACKJACK

Taylor Zheng (TaylorZheng2015@u.northwestern.edu)
Joonyong Rhee (JoonyongRhee2016@u.northwestern.edu)
Alexander Wenzel (AlexanderWenzel2017@u.northwestern.edu)

Northwestern University
EECS 349 Machine Learning - Spring Quarter 2015

Machine Learning solutions can be applied to a wide variety of problems, including decision-making tasks. One key area of decision-making in machine learning is the ability of a machine to learn rules and strategies in an environment in which outcomes are based on probability, such as card games, dice or other chance-based games, or any environment in which the result cannot be anticipated based on the action taken. We demonstrate that current machine learning techniques can reach human-level performance on probability-based problems with Hackjack, a Python implementation of a Blackjack-learning machine.

Hackjack uses Q-learning techniques to teach itself to play Blackjack, maintaining a number of 'Q values' corresponding to different game states and updating them based upon a reward system. Through trial-and-error and repetitive learning, Hackjack teaches itself first the rules of the game, and eventually, higher-level strategy. In fact, we demonstrate that Hackjack's performance, through constant feedback from gameplay, converges to the performance achieved by the "Dealer's Strategy," one of the most commonly-employed strategies in the game (and explained later in this report).

To implement Hackjack, we wrote three Python elements. First, we made a Blackjack game engine, which manages the deck, interacts with the player, and gives feedback on the state of the game. Second, we made a Player, which keeps track of its own hand and card values and makes decisions. Lastly, we implemented our own version of the Q-learning algorithm tailored for this task, which was responsible for evaluating each decision made by the player and for providing the player with positive or negative feedback based on information from both the player and the game. Testing was conducted by running a set number of games and by recording the player's winning percentage after each game. For comparison, we also built two separate players, one for playing a "random strategy" and one for playing the "dealer's strategy."

After building and testing this implementation, we see that Hackjack's winning percentage improves greatly over time, and that it eventually converges to the same win percentage as the popular "Dealer's Strategy." For instance, after playing 100,000 games, Q-learning achieved a winning percentage of 39.5%, only 0.3 percentage points away from the Dealer Strategy's 39.8%, and performed significantly better than chance, as it easily beat the random strategy's 27.9%. With Hackjack, we demonstrate the viability of machine learning techniques, such as Q-Learning, for providing usable solutions to problems involving a significant degree of uncertainty.