CS182 Final Project Status Update

Amy Kang, George Zhang

November 22, 2016

1 Problem

Blackjack is a popular casino game with a very simple premise: given two cards out of a standard card deck and knowledge about one of the dealer's cards, act optimally to beat the dealer without busting. We would like to investigate whether a reinforcement learning algorithm, such as Q-learning, can learn to play the game and win on a consistent basis. Here, we will define "win" to mean monetary value, and not number of hands won.

2 Progress

Over the last 2 weeks since submitting the proposal, our goal has been to get the game itself running. We wanted to head into Thanksgiving break with a working version of blackjack that can be easily trained on, so that during and coming out of the break, we can focus on learning the game. To that end, we were relatively successful, having coded a basic version of the game.

We are currently able to simulate several rounds, with the caveat that the user is only able to hit and stand at this point. We found a "basic strategy" table online, which recommends actions based on player hand value and dealer upcard, but the actions include doubling and splitting. We would like to compare our initial learned version to this strategy as a benchmark, and so we expect to have to implement doubling and splitting as well before we start training.

The game itself is run on keyboard input from the user. We expect this format to easily translate to an automated learner, as the actions returned to the game are integers, which we can easily return using a learned function.

3 Challenges

We have not run into any significant challenges at this point. There have been some design decisions that were not easily made, and we hope that the solutions we settled upon will not complicate the learning process in the future.

4 Next Steps

As stated previously, we may need to implement doubling and splitting before learning anything. If we decide this is the way to go, this will be our next step. Alternatively, we can implement a simple Q-learning algorithm with only these two options to ensure compatibility with our current framework. Both of these are valid next steps and will need to be completed, but we have yet to decide which will come first.

At some point, we would also like to implement a random agent in order to provide both a framework for our later learning agents as well as a standard of comparison for our more intelligent agents. The random agent will serve as a baseline for other strategies.

We also have the basic strategy table we found. We may also decide to implement this strategy to serve as another benchmark and see if we can beat this strategy. We expect Q-learning to come up with a strategy that is similar, but additional improvements (such as counting cards and strategized betting) should end up performing better.