# Project II - SF2957 Statistical Machine Learning

Due, 15 December, 13:00

## 1 Reinforcement learning for Blackjack

The purpose of this project is to use reinforcement learning to train an agent to play blackjack. Two different representations of the state space will be considered, one which is based on representing the state as the set of cards and the dealer's card sum, this will be referred to as the "hand" state-space and one that represents the state as the agent's and dealer's card-sum. The card-sum representation is implemented in some standard approaches to Blackjack, but is somewhat flawed (as will be discussed later). Both "On-policy Monte-Carlo learning" and "Q-learning" will be considered.

The software environment is already rather complete, so you will only have to implement the core of the learning algorithms and evaluate the results.

## 2 Blackjack

In this project the game of Blackjack is setup in the following manner.

- One agent plays against the dealer, with the agent staking one unit on each hand.
- In each non-terminal state, two actions possible: ask for another card, or stay,
- The cards 2–10 counts as their numerical value, suites counts as 10, and ace counts as either 1 or 11 depending on whichever is best. If an ace can be counted as 11 without the agent going bust (card sum exceeds 21) it is known as a *usable* ace, the same goes for the dealer.

The goal of the agent is to beat the dealer in one of the following ways.

- Obtain 21 points on the first two cards, knows as a blackjack, without a dealer blackjack. Reward 1.5.
- Reach a final score higher than the dealer without exceeding 21. Reward: 1.

- Dealer gets points exceeding 21 and player does not. Reward: 1.

The dealer always plays according to the same strategy: draw cards until it has a card sum greater than or equal to 17. The game starts with the dealer giving the player two cards (visible) and himself two cards (one visible, one hidden). The agent is then allowed to ask for more cards until she decides to stay , after which the dealer follows his strategy until done. If player's points exceeds 21 his stake is lost regardless of dealers outcome (this is where the house edge comes from); if player's points equals dealer's points this is a push and stake is returned to player; otherwise payout is made according to above rules. Standard practice is that the cards are drawn from 6–10 decks.

### 2.1 The state space of Blackjack

In Blackjack the color and suite of the cards does not matter, only their numerical values. The numerical values will be used as the cards identifier, with aces equal to 1. The state space may be represented by the agent's hand, a vector $(s_1^a, \ldots, s_{10}^a)$ where $s_i^a$ is the number of cards of value $i$ with card sum $\sum_{i=1}^{10} i s_i^a \leq 32$, and the dealer's hand $(s_1^d, \ldots, s_{10}^d)$, where $\sum_{i=1}^{10} i s_i^d \leq 26$. With this state representation it is always possible to determine if a state is terminal, it is sufficient to check if the agent or dealer is bust, or if dealer's card sum exceeds 17. Moreover, the hand $(1, 0, 0, 0, 0, 0, 0, 0, 0, 1)$ is a Blackjack.

By the rules of the game, the agent's policy need only to take into account the dealer's first card and the state space can be reduced to

$$\mathcal{S}_1 = \{(s_1, \ldots, s_{10}, S_d) \, : \, \sum_{i=1}^{10} i s_i \leq 31, S_i \geq 0 \text{ for } i = 1, \ldots, 10, \text{ and } S_d \leq 26\},$$

where $S_d$ is the dealer's card sum of the dealer. The rewards $R_1, R_2, \ldots$ follows the above payout rules, with $R_t = 0$ if $S_t$ is not terminal.

An alternative state space, sometimes encountered for Blackjack environments is to further reduce to

$$\mathcal{S}_2 = \{(S_a, u, S_d) : 1 \leq S \leq 31, u \in \{0, 1\}, 1 \leq S_d \leq 26\}$$

where $S_a, S_d$ is the card sum of the agent and the dealer, and $u$ indicates if the player is holding a usable ace (1) or not (0). This state space has some flaws. For instance, the state $(21, 1, S_d)$ may or may not be a Blackjack (depending on how many cards have been drawn). Moreover, in the finite deck setting, this representation does not keep track of the remaining number of cards in the deck.

In the numerical experiments both $\mathcal{S}_1$ and $\mathcal{S}_2$ will be considered.

## 3 Numerical experiments for Blackjack

A Python code for reinforcement learning of Blackjack is provided in the `code` directory. Before you begin you need to do the following:

1. Install `aigym` in your version of Python 3, see `https://gym.openai.com/docs/#installation`.

2. Download the files in the `code` directory, available on the Canvas page under `Files`.

3. In the `code` directory, create subdirectories, named `data` and `figures`. The output from the learning algorithms will be saved to these directories.

The main program is `run_RL.py` which is setup to train the agent for $10^7$ episodes on a specified number of decks, $1, 2, 6, 8$, and infinitely many decks. The larger state space, $\mathcal{S}_1$, called the *extended state space*, is encoded in the environment `env`, whereas the smaller state space $\mathcal{S}_2$ is called the *sum-state space* is encoded in the envrionment `sum_env`.

The learning algorithms are coded in the file `RL.py` in the functions `learn_MC` for *On-policy Monte-Carlo learning* and `learn_Q` for *Q-learning*. In both of these programs, you need to add the suitable code for the learning of the action-values for each state-action pair.

## Assignments

(a) *On-policy Monte-Carlo learning.* Add the code, at the indicated place in the function `learn_MC`, for updating the reward in the variable `avg_reward` and update the action value in the dictionary named `Q`.

(b) *Q-learning.* Add the code, at the indicated place in the function `learn_Q`, for computing the learning rate and updating the action value in the dictionary named `Q`.

(c) For the following three cases

   (1) on-policy Monte-Carlo learning on the state space $\mathcal{S}_1$,

   (2) Q-learning on the state space $\mathcal{S}_1$,

   (3) Q-learning on the state space $\mathcal{S}_2$,

   compare the performance of the learning algorithms. Which one achieves better learning? For (2) and (3), experiment on the learning rate, to find a suitable decay rate of the form $\alpha_n = cn^{-\omega}$, $0 \leq \omega < 1$. Also explore the impact of having decaying exploration rate $\epsilon$. The exploration rate is the probability of not choosing the optimal policy.

(d) For the best learning algorithm, provide a close to optimal strategy by specifying a table of actions (”hit” or ”stay”), depending on the agent’s card sum, the dealer’s card sum, and whether there is a usable ace, or not.

## Report

Your group must hand in a report, by uploading it on Canvas, containing the following items:

- A mathematical description of the on-policy Monte-Carlo learning algorithm on which the solution in (a) is based.
- A mathematical description of the Q-learning algorithm on which the solution in (b) is based.
- Plots comparing the performance of the algorithms as mentioned in (c) as well as relevant plots that support your evaluation of the learning rate.
- A description of a close to optimal strategy by specifying a table of actions as mentioned in (d).