Examples

1. Blackjack game

1.1. Monte Carlo Prediction

The game is detailed explained on chapter 5.1 – Monte Carlo Policy Evaluation of [1]. Assuming you already know how to play blackjack, the game can be summarized as:

- Face cards (Jack, Queen, King) have point value 10
- Aces can either count as 11 or 1, and it's called 'usable' at 11.
- The game starts with each (player and dealer) having one face up and one face down card.
- The player can request additional cards (hit=1) until they decide to stop (stick=0) or exceed 21 (bust)
- After the player sticks, the dealer reveals their facedown card, and draws until their sum is 17 or greater. If the dealer goes bust the player wins.
- If neither player nor dealer busts, the outcome (win, lose, draw) is decided by whose sum is closer to 21. The reward for winning is +1, drawing is 0, and losing is -1

The observation of the agent at every state is a Python 3-tuple of:

- The player current sum
- The dealer's showing car (1, 2, …, 10)
- Whether or not the player holds a usable ace (0, 1)

Check the detailed code implementation here.

1.2. Monte Carlo Control

In the previous example we simply used a MC approach to perform a policy evaluation. That is, we run episodes of the blackjack game following a certain (given) policy, to determine how good that policy was. Or, in simpler words, how much money will you win or lose if you always act following this policy?

Now, for the control problem, the question becomes more interesting. We actually want to find which is the optimal policy to play this (constrained version of) blackjack game? Or, in simpler words, how to play to win the maximum possible money?

We use this time an Every-visit MC Control algorithm.

The detailed code implementation here.

2. Frozen Lake – Q-Learning

Frozen lake is a really small game in which we are able to visualize tabular methods. It is one of the environments in the OpenAI gym

It consists on a grid world with few holes where the agent 'dies', a starting state and a goal state. The reward is only given once the agent reaches the state. So, it is also very helpful to visualize sparse rewards that will be covered in detailed further.

In Q-Learning, we keep updating the values of the called Q-Table, with maps state-action pairs with rewards. In this demo it can be seen how the agent start running episodes and interact with the map, and update its Q-Values with the experience.

3. Cliff Walking Environment

3.1. Sarsa algorithm: On-Policy Temporal Difference

Hdfh

3.2. Q-Learning: Off-Policy Temporal Difference

Adfaf