ENSAE ParisTech

Master 1st year Python Machine Learning Project

On-Policy Reinforcement Learning for Blackjack

by Mehdi Abbana Bennani

supervised by
Pr. Xavier Dupré

December 19, 2016

Abstract

This report is about the second year Python project I completed at ENSAE ParisTech. the assignment corresponds to the Easy21 assignment provided by the Pr. David Silver in his Reinforcement Learning course at UCL (University College London), I then goes further by extending the algorithms to the case with cards sampling without replacement and the integration of a card history memory to the agent.

Contents

Introduction

The report is organized as follows:in the first part, I define the problem, in the second part I describe the reinforcement learning terminology and the algorithms I will use. In the third part, I will describe some important coding aspects such as code structure and performance improvement tricks.

Motivation

Reinforcement learning is a one of the major Machine Learning classes of algorithms. It achived a better performance than many state of the art algorithms in many domains, such as in games [1] [3], shape recognition [2], and it is used for some very complex problems such as driveless cars.

Through this assignment, I aim to:

- explore and understand Reinforcement Learning Algorithms
- apply these algorithms to concrete problems and explore their limits
- test new approaches and personal ideas in these problems

1 Problem definition

1.1 Simple case

This exercise is similar to the Blackjack example in Sutton and Barto 5.3, however, the rules of the card game are different and non-standard.

- The game is played with an infinite deck of cards (i.e. cards are sampled with replacement). Each draw from the deck results in a value between 1 and 10 (uniformly distributed) with a colour of red (probability 1/3) or black (probability 2/3). There are no aces or picture (face) cards in this game
- At the start of the game both the player and the dealer draw one black card (fully observed)
- Each turn the player may either stick or hit. If the player hits then she draws another card from the deck. If the player sticks she receives no further cards
- The values of the player's cards are added (black cards) or subtracted (red cards). If the player's sum exceeds 21, or becomes less than 1, then she "goes bust" and loses the game (reward -1)
- If the player sticks then the dealer starts taking turns. The dealer always sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome – win (reward +1), lose (reward -1), or draw (reward 0) – is the player with the largest sum

1.2 Card history memory integration and card sampling without replacement

In this part, I will put additional constraints and drop some other hypotheses on the environment and the agent:

- The cards sampling is without replacement, so there is a set of 30 cards, 20 black and 10 red, sampled with the same probability.
- The agent will remember the cards which were played before and will adapt his actions to these information.
- The dealer will not take into account this history and will play as defined in the previous section.

2 Reinforcement Learning

2.1 Definitions

Definition 1 *State Value Function*

The state value function v(s) of a Markov Reward Process is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

Definition 2 *Policy*

A policy π is a distribution over actions given states

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

Definition 3 *Return*

The return G_t is the total discounted reward from time step t.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Definition 4 *State Value Function*

The state value function $v_\pi(s)$ of a MDP is the expected return starting from state s, and then following the policy π

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

Definition 5 *State Action Value Function*

The state action value function $q_\pi(s, a)$ of a MDP is the expected return starting from state s, and taking action a, then following the policy π

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

The state action value function $q_\pi(s, a)$ of a MDP is the expected return starting from state s, and taking action a, then following the policy π

$$q_\pi(s) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

Definition 6 *Optimal Value Function*

The optimal state value function $v_\star(s)$ is the maximum state action value function over all policies

$$v_\star(s) = \underset{\pi}{max} v_\pi(s)$$

The optimal state action value function $q_\star(s, a)$ is the maximum state action value function over all policies

$$q_\star(s, a) = \underset{\pi}{max} q_\pi(s, a)$$

Definition 7 *Optimal Policy*
We define a partial ordering over a policies:
$\pi \geq \pi'$ *if* $v_\pi(s) \geq v_\pi(s)$

2.2 Theorems

Theorem 1 *Optimal Policy For any MDP*

- *There exists an optimal policy* $\pi_\star$ *that is better than or equal to all other policies* $\pi_\star \geq \pi, \forall \pi$
- *All policies achieve the optimal value function* $v_{\pi_\star} = v_\star$
- *All policies achieve the optimal state action value function* $q_{\pi_\star} = q_\star$

Theorem 2 *The Bellman Expectation Equation*
The state-value function can be decomposed into immediate reward plus discounted value of the successor state.

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

The action-value function can similarly be decomposed

$$q_\pi(s) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

2.3 Monte Carlo Algorithms

The intuition behind this algorithm is that, in order to estimate the value of a state action pair, we run a full episode, with a policy such as ϵ greedy policy,then we update our current estimation of the action-value function in the direction of the average of the action value estimation of all visited state-action pairs during the episode.

2.4 Temporal Difference Learning Algorithms

SARSA Algorithm

The intuition behind this algorithm is that, in order to estimate the value of a state action pair, we run a full episode, with a policy such as ϵ greedy policy,then we update our current estimation of the action-value function in the direction of the average of the action value estimation of all visited state-action pairs during the episode.

Algorithm 1 Greedy in the Limit with Infinite Exploration Algorithm (GLIE)

Sample kth episode using $\pi : \{S_1, A_1, R_1, S_2, ...\}$
for $eachstateS_tandActionA_tintheepisode$ **do**
  $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{G_t - Q(S_t, A_t)}{N(S_t, A_t)}$
Improve policy based on new action-value function
$\epsilon = 1/k$
$\pi \Leftarrow \epsilon - greedy(Q)$

Algorithm 2 SARSA Algorithm for On-Policy Control

Initialize $Q(s, a), \forall s \in \mathbb{S}, a \in \mathbb{A}(s)$, arbitrarlily, and Q(terminal-state,-)=0
for each episode **do**
  Initialize S
  Choose A from S using policy derived from Q (e.g., $epsilon - greedy$)
  for each step in the episode **do**
    Take action A, observe R, S'
    Choose A' from S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A)$
    $S \leftarrow S'$
    $A \leftarrow A'$

Algorithm 3 SARSA-λ Algorithm for On-Policy Control

Initialize $Q(s, a), \forall s \in \mathbb{S}, a \in \mathbb{A}(s)$, arbitrarily, and Q(terminal-state,-)=0
for each episode **do**
  E(s,a) = 0 for all $s \in \mathbb{S}, a \in \mathbb{A}(s)$
  Initialize S, A
  for each step in the episode **do**
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g. ϵgreedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    for all $s \in S, a \in \mathbf{A}(s)$ **do**
      $Q(s, a) \Leftarrow Q(s, a) + \alpha\delta E(s, a)$
      $E(s, a) \leftarrow \gamma\lambda E(s, a)$
    $A \leftarrow A', S \leftarrow S'$,

SARSA-λ Algorithm

2.5 Action Value Function Approximation

Where the state action space size is very big, we cannot use table lookup anymore to get the state action value function values for all states. In this case, we approximate the value function by parametrizing it. The state action value function becomes $Q(S, A, w)$, where w is a vector which size is the feature space size.

Monte Carlo with Action Value Function Approximation

Sarsa with Action Value Function Approximation

3 Reinforcement Learning applied to Blackjack

The MonteCarlo Algorithm is the most stable algorithm, the value function approximation 1 is related to the game.........................

We see that it reaches its maximum for the states ....

3.1 Monte Carlo

Figure 1: Optimal Value Function after 10^6 episodes using GLIE

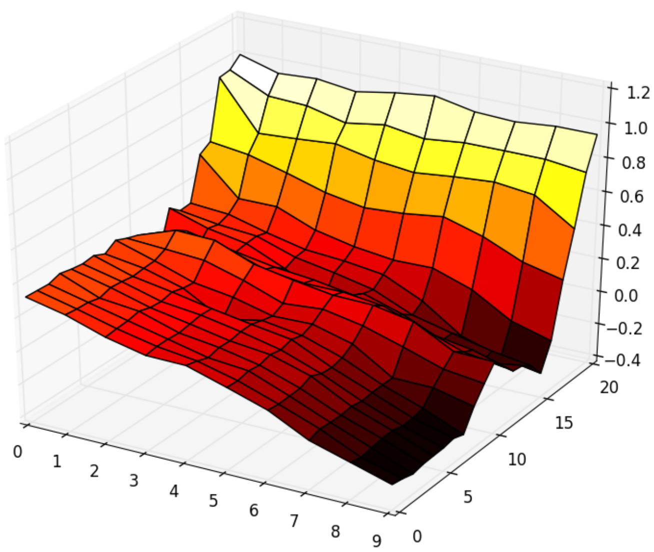
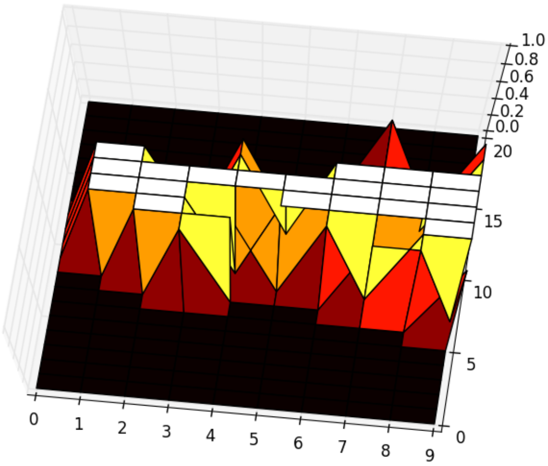


Figure 2: Optimal Actions after 10^6 episodes using GLIE



3.2 TD Learning

SARSA

Figure 3: Optimal Value Function after 10^6 episodes using SARSA

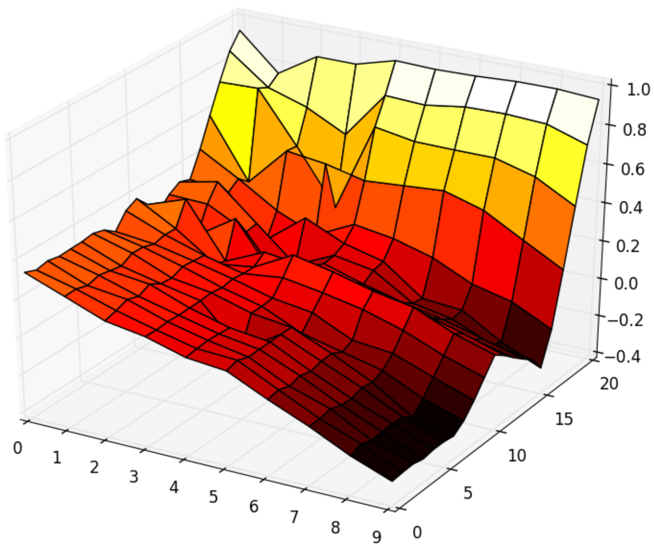
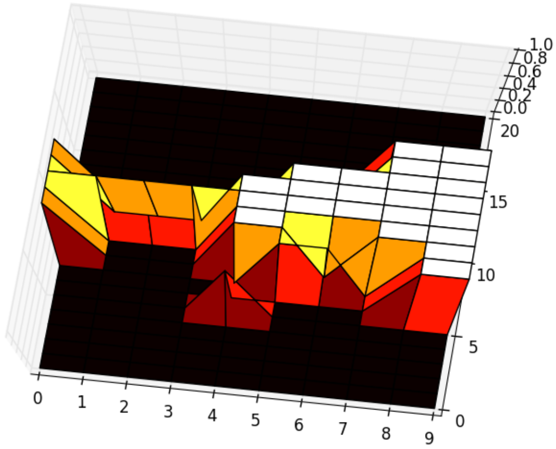


Figure 4: Optimal Actions after 10^6 episodes using SARSA



SARSA-λ

Figure 5: Optimal Value Function after 10^6 episodes using SARSA-λ for $\lambda = 0.8$

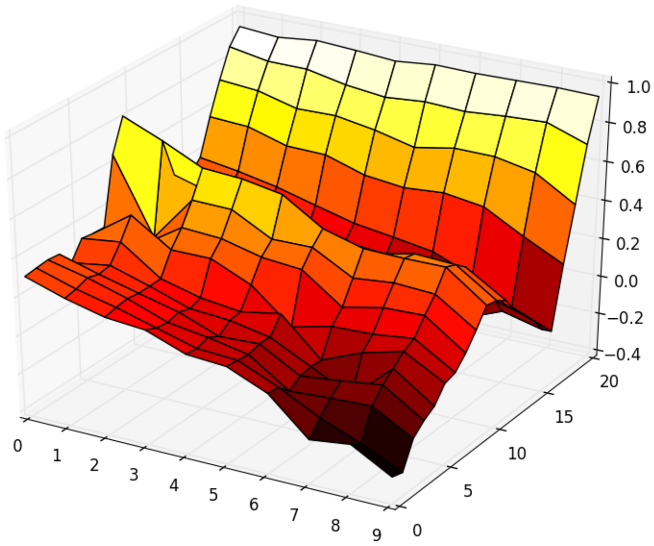
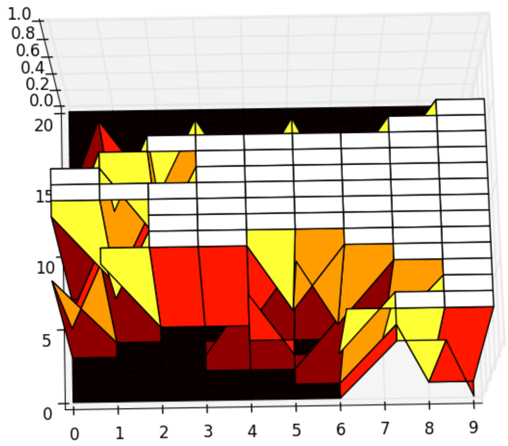


Figure 6: Optimal Actions after 10^6 episodes using SARSA-λ for $\lambda = 0.8$



3.3 Function Approximation

Linear Function Approximation

Figure 7: Optimal Value Function after 10^6 episodes using Linear Function Approximation and SARSA-λ for $\lambda = 0.8$

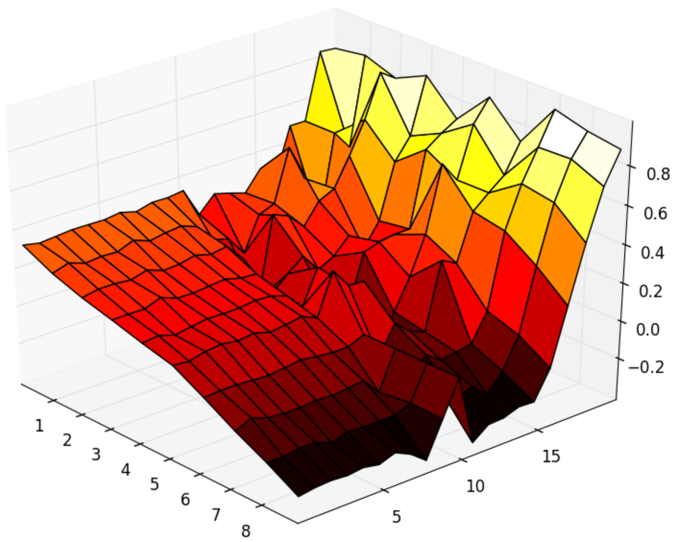
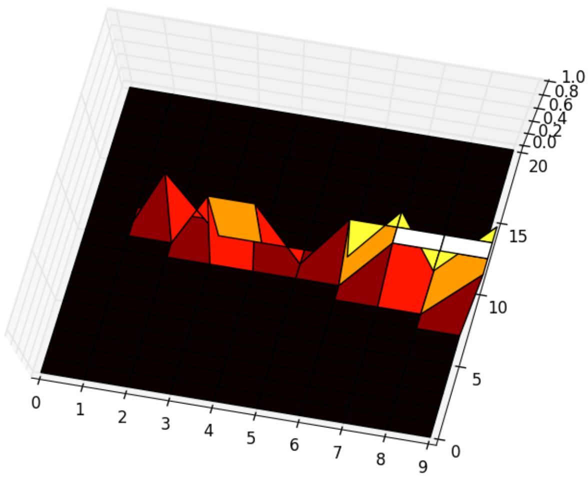


Figure 8: Optimal Value Function after 10^6 episodes using Linear Function Approximation and SARSA-λ for $\lambda = 0.8$



3.4 Conclusion

4 Coding aspects

4.1 Profiling

4.2 Algorithms Optimization

5 Conclusion and further possible developpements

Deep Q Networks and Experience Replay [1]

6 Appendix

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[2] Danilo Jimenez Rezende, S. M. Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. *CoRR*, abs/1607.00662, 2016.

[3] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[4] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.