Mini-project 2
Reinforcement learning for the mountain-car problem

Rodrigo C. G. Pena
rodrigo.pena@epfl.ch

January 19, 2017

Abstract

This report describes an implementation of the SARSA(λ) (Sutton & Barto (1998)) reinforcement learning algorithm for the solution of the mountain-car problem.

1 Introduction

The mountain-car problem is a standard benchmark in reinforcement learning, popularized by (Sutton & Barto, 1998). In this problem, a car has to learn to climb the valley in which it is initially stuck. This setup is depicted in Figure 1. The car has only two gears, forward and reverse, and a neutral position, giving rise to three possible actions at any given time. The car's engine is not powerful enough to climb the hill in one go, so the car has to learn to swing back-and-forth to reach its goal.

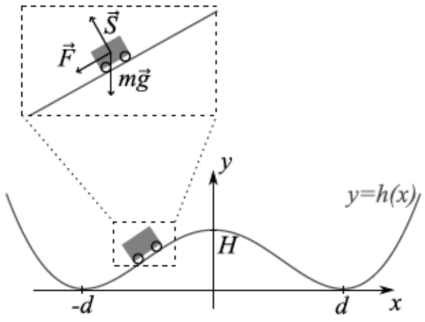


Figure 1: Mountain-car setup

We approach solving this problem by reinforcement: as soon the car has reached the top of the hill, we give it a reward $R = 1$; otherwise, $R = 0$. The algorithm itself is a version of a temporal difference (TD) method called SARSA (State-Action-Reward-State-Action), paired up with eligibility traces to speed-up the training. The eligibility traces are governed by a parameter λ, so the full algorithm is called SARSA(λ).

In SARSA, one discovers how to perform a task by learning so-called Q-values for each action-state pair. Those Q-values govern the probability of taking a given action in a given state. In the mountain-car problem, the actions are "forward force","reverse force", and "no force", while the states are completely determined by car's horizontal position and horizontal velocity, $s(t) = (x(t), \dot{x}(t)), t >= 0$.

The Q-values are modeled with help from a 2-layer neural network. The neurons in the input layer form a grid that spans the state space s for $-150m \leq x \leq 30m$ and $-15m/s \leq \dot{x} \leq 15m/s$. Each neuron j in this layer has activation

$$r_j(s = (x, \dot{x})) = \exp\left(-\frac{(x_j - x)^2}{\sigma_x^2} - \frac{(\dot{x}_j - \dot{x})^2}{\sigma_{\dot{x}}^2}\right), \tag{1}$$

where $s_j = (x_j, \dot{x}_j)$ is the center of neuron j in the grid, and σ_x and $\sigma_{\dot{x}}$ are the strides of neighboring neuron centers in the x and $\dot{x}$ directions, respectively. Finally, each neuron i in the output layer has activation

$$Q(a_i, s) = \sum_j w_{a_i,j} r_j(s), \tag{2}$$

where each $a_i, i = 1, 2, 3$ represents one of the possible actions of the car.

Learning the Q-values reduces then to learning the weight matrix $W = \{w_{a_i,j}\}$ of our neural network. Algorithm 1 details the full SARSA(λ) procedure for doing just so.

2 Experiments

In this section, we take a close look at the learning procedure. In particular, we observe how changes in the input hyper-parameters of the SARSA(λ) algorithm affect the learning curve of the agent. Whenever a trial ends, we record the number of steps the car needed to reach its goal. This number is also known as escape latency. The learning curve is defined as the collection of escape latency values for all trials. The hope is that, as the agent learns its task, the time required to complete it should go down, i.e., the learning curve should tend towards smaller values as the number of trials increases.

Unless stated otherwise, the following default values were used for the hyper-

Algorithm 1 SARSA(λ) for a 2-layer neural network

Input: $\gamma = 0.95$, $0 \leq \lambda \leq 1$, $0 < \eta < 1$, $\tau \geq 0$, $n_{\text{trials}}, n_{\text{steps}} \in \mathbb{N}_+$

for $k = 1, \ldots, n_{\text{trials}}$ **do**
  Reset eligibility traces: $e(a_i, j) = 0, \forall$ action i, neuron j.
  Pick random initial state s^*.
  Pick first action a^* with probability $\mathbb{P}(a^* = a_i) = \frac{\exp(Q(a_i, s^*)/\tau)}{\sum_i \exp(Q(a_i, s^*)/\tau)}$.
  for $l = 1, \ldots, n_{\text{steps}}$ **do**
    Observe next state s' after action a^*.
    Observe reward R.
    Pick next action a' with probability $\mathbb{P}(a' = a_i) = \frac{\exp(Q(a_i, s')/\tau)}{\sum_i \exp(Q(a_i, s')/\tau)}$.
    $\delta \leftarrow R + \gamma Q(a', s') - Q(a^*, s^*)$
    $e(a_i, j) \leftarrow \gamma\lambda e(a_i, j), \forall i, j$
    $e(a^*, j) \leftarrow e(a^*, j) + r_j(s^*)$
    $w_{a_i,j} \leftarrow w_{a_i,j} + \eta\delta e(a_i, j), \forall i, j$
    if $R > 0$ **then**
      break
    $a^* \leftarrow a'$
    $s^* \leftarrow s'$

Output: $W = \{w_{a_i,j}\}$

parameters in Algorithm 1:

$$\begin{gathered} \gamma = 0.95 \\ \lambda = 0.95 \\ \eta = 0.01 \\ \tau = 0.1 \\ n_{\text{trials}} = 100 \\ n_{\text{steps}} = 10000 \\ w^{(0)}_{a_i,j} \sim \mathcal{U}(0,1), \forall i \in \{1,2,3\}, \forall j \in \{1, \ldots, 400\} \end{gathered} \quad . \tag{3}$$

All the figures in this section can be reproduced with help from the accompanying python scripts and jupyter notebook in github.

To show that the default setup allows for learning, Figure 2 shows the average and standard deviation of the learning curves over 16 agents, using the default hyper-parameter values in Algorithm 1.

2.1 Most likely action in each state

First we take a glimpse at how the policy of the agent changes throughout the training. Figure 3 shows a vector field of the directions (actions) with highest Q-values on the car's state space. In view of the action policy in Algorithm 1, those vectors also represent the most likely action to be taken by the car at each given point of its state space.

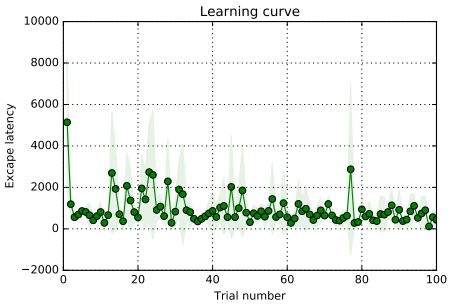


Figure 2: Average and standard deviation of the learning curves over 16 agents, using the default hyper-parameter values in Algorithm 1.

2.2 Changes in the exploration temperature τ

Figure 4 shows the effect of changes in the exploration temperature τ on the learning curve of the agent.

2.3 Changes in the eligibility trace decay rate λ

Figure 5 shows the effect of changes in the eligibility trace decay rate λ on the learning curve of the agent.

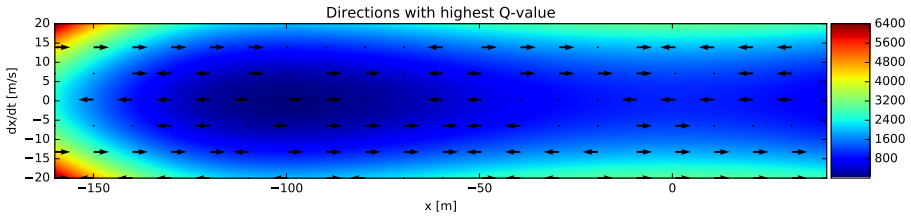
2.4 Changes in the initialization of the weights $\{w_{a_i,j}\}$

Figure 6 shows the effect of changes in the initialization of the weights $\{w_{a_i,j}\}$ on the learning curve of the agent.

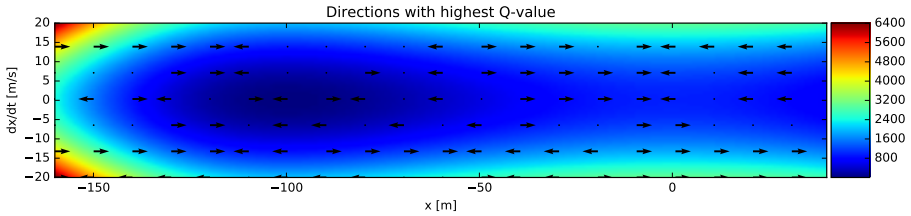
3 Discussion

Figure 3 is reassuring, because it shows that the agent is roughly learning that it has to accelerate in the direction of its horizontal speed in order to gain momentum. Figure 2 further confirms that the agents are learning to perform their task because both the average and variance of the escape latency decrease as the number of trials increases.

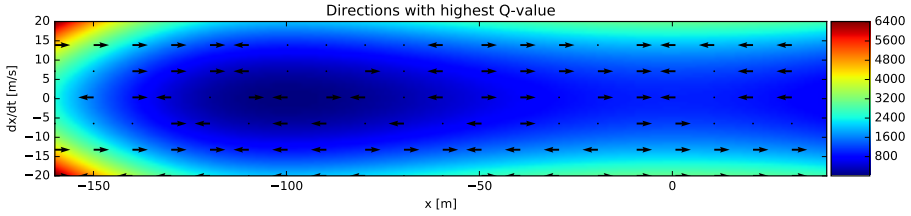
In Figure 4a, $\tau = 0$ means the agent is always going to choose the action with the highest Q-value in any given state (greedy policy). Initially, the agent's model of the Q-landscape (represented by the weights W) is bad the escape latency will have a lot of variance: the escape latency is only going to be small if the state the car begins in is favorable with respect to the actions the agent is always choosing. Even though we observe a decrease in mean and variance around trial 70, this behavior doesn't hold up, indicating that this greedy policy is not efficient here, given the way we initialized the network's weights (uniformly at random).



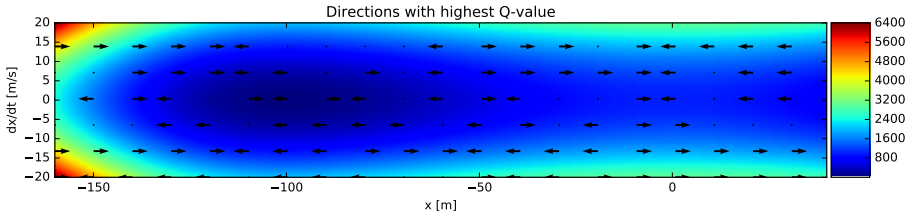
(a) Before training



(b) After 25 trials

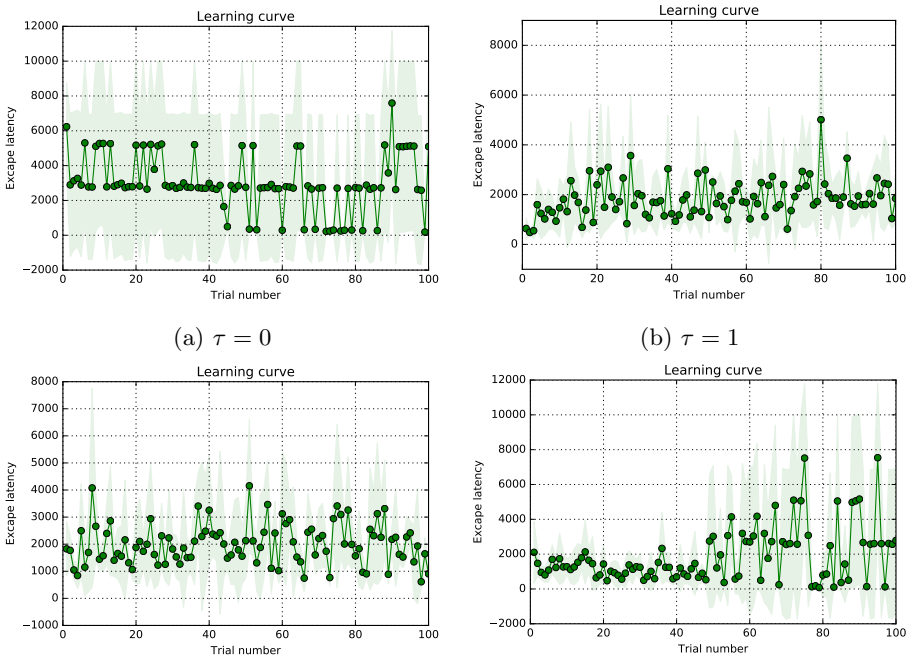


(c) After 50 trials



(d) After 75 trials

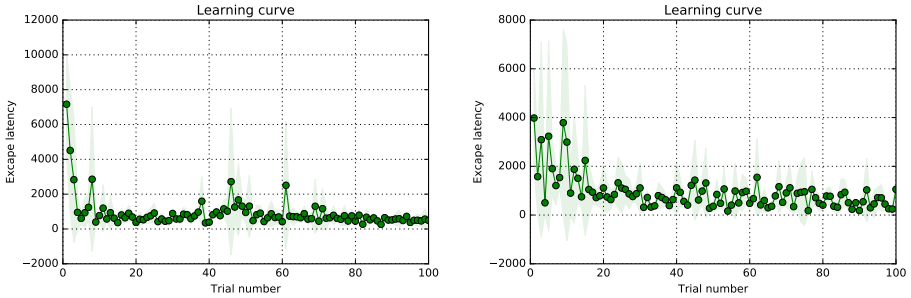
Figure 3: Vector fields of most likely actions on the state space. Snapshots were taken every 25 trials.



(a) $\tau = 0$ (b) $\tau = 1$

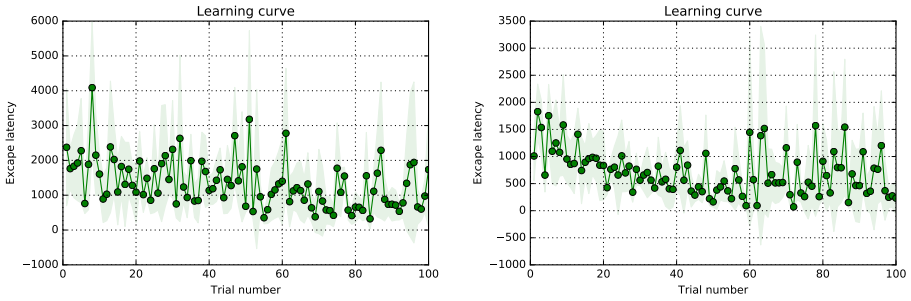
(c) $\tau = \infty$ (d) $\tau^{(l)} = \epsilon^{k/n_{\text{trials}}}, 0 < \epsilon \ll 1$

Figure 4: Effect of τ on the learning curves. Curves show the mean and standard deviation over 16 different agents.



(a) $\lambda = 0$ (b) $\lambda = 0.95$

Figure 5: Effect of λ on the learning curves. Curves show the mean and standard deviation over 16 different agents



(a) $w^{(0)}_{a_i,j} = 0, \forall i, j$ (b) $w^{(0)}_{a_i,j} = 1, \forall i, j$

Figure 6: Effect of weight initialization on the learning curves. Curves show the mean and standard deviation over 16 different agents

In the $\tau = \infty$ case (Figure 4c), the policy ceases to depend on the Q-values, and the actions are chosen uniformly at random at each step. Even if the agent is learning proper Q-values by reaching its goal by chance every once in a while, this learning is not reflected in the escape latency values, as those will depend merely on chance. Hence the large variance observed at each trial.

In Figure 4b, we observe a similar behavior as in Figure 4c, because $\tau = 1$ is a relatively high value in our problem. Figure 4d interpolates, in a way, between the behaviors observed in Figure 4b and Figure 4a. The mean and variance of the escape latency is particularly low when $\tau \approx 0.1$, which is why this value was chosen as default for the exploration temperature.

It is worth noting that τ controls the trade-off between exploration (large τ) and exploitation (small τ), and it is particularly important in the first couple of trials, when the agent is not confident about its Q-values, that we allow for random actions to be taken.

Figure 5 is more puzzling, because the presence of eligibility traces ($\lambda > 0$) is supposed to speed up the learning process, by propagating the Q-value updates across the state space. However, we observe the opposite: having $\lambda = 0$ results in agents that learn faster.

Finally, we see from Algorithm 1 that the weights coming from the initialization $w^{(0)}_{a_i,j} = 0, \forall i, j$ (Figure 6b) will only start changing when the car reaches the goal. Until then agent's policy is simply to choose the next action uniformly at random. We would expect that this delayed update should drag down the learning speed, and this suspicion is certainly confirmed by Figure 6. Note, nonetheless, that the initialization $w^{(0)}_{a_i,j} = 1, \forall i, j$ also has its problems, as the agent will consistently choose the same action throughout most of the car's state space. This contributes to the variance observed, particularly after trial 60 in Figure 6a. It was to avoid those two extreme behaviors that the initialization $w^{(0)}_{a_i,j} \sim \mathcal{U}(0, 1), \forall i, j$ was chosen by default.

References

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, 1998. ISBN 0262193981. URL https://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html.