Reinforcement learning for the mountain car problem

Manu Srinath Halvagal, Ze ya Yin

June 15, 2018

1 Introduction

In this report, we present a study of a reinforcement learning solution to the mountain car problem with discrete actions. The problem consists of a car learning to apply appropriate forces to swing back and forth in a valley to overcome a steep hill. The problem is interesting because the car does not have enough power to directly climb the hill, i.e, it needs to gain speed (kinetic energy) over time by swinging back and forth.

The problem is formulated as follows:

- **State:** The state consists of the car's position (in the x-direction only) and the velocity along the x-direction. The y-coordinate is irrelevant to the problem, and the z-coordinate is directly determined by the x-coordinate according to $h(x) = \frac{(x-d)^2(x+d)^2}{x^2+C}$. Thus the state-space is a 2-D continuous space in the ranges $[-150m, 30m]$ for the position and $[-15m/s, 15m/s]$ for the velocity. The car starts in the $x < 0$ region with an initial velocity that is insufficient to directly overcome the hill.
- **Actions:** The actions available to the agent at every time step are to do nothing or to apply a force of fixed magnitude in either the forward or backward direction. Thus the action space consists of three discrete actions.
- **Reward:** The goal is to overcome the hill and reach the $x > 0$ region upon which the agent receives a reward of $+1$ and the episode terminates.

2 Method: Sarsa(λ) algorithm

Our solution is based on the Sarsa(λ) algorithm using a biologically plausible neural network to parametrically approximate the continuous state space, and then calculate the state-action values.

2.1 Network Structure

The input layer consists of neurons in a uniform grid spread over the bouded 2-dimensional input space, with each neuron having a localized Gaussian receptive field $r_j(s) = \exp\left(-\frac{(x_j-x)^2}{\sigma_{x_j}^2} - \frac{(\dot{x}_j-\dot{x})^2}{\sigma_{\dot{x}_j}^2}\right)$. The output layer consists of three neurons that output the action values $Q(s, a)$ for each of the three actions in the current state s. The network is fully connected from the Gaussian neurons and the output layer, i.e, $Q(s, a) = \sum_j w_{a,j} r_j(s)$, and it is these weights $w_{a,j}$ that are learned over time.

2.2 Sarsa(λ)

The training (with learning rate η) according to the Sarsa(λ) algorithm proceeds as follows. At every iteration:

1. Get current state s and calculate action values $Q(s, a)$ from the action-value network.

2. Pick action a with policy probabilities given by a softmax over $Q(s, a)$. Apply action a, observe new state s' and reward r.
3. Pick new action a' in state s' in the same manner.
4. Update eligibilities as $e_t(s, a) = \gamma\lambda e_{t-1}(s, a) + \begin{cases} r_j(s) & \text{if a = action taken} \\ 0 & \text{otherwise} \end{cases}$
5. TD error: $\delta = r + \gamma Q(s', a') - Q(s, a)$
6. Update weights $w_{a,j} = w_{a,j} + \eta\delta e_t(s, a)0$

3 Results

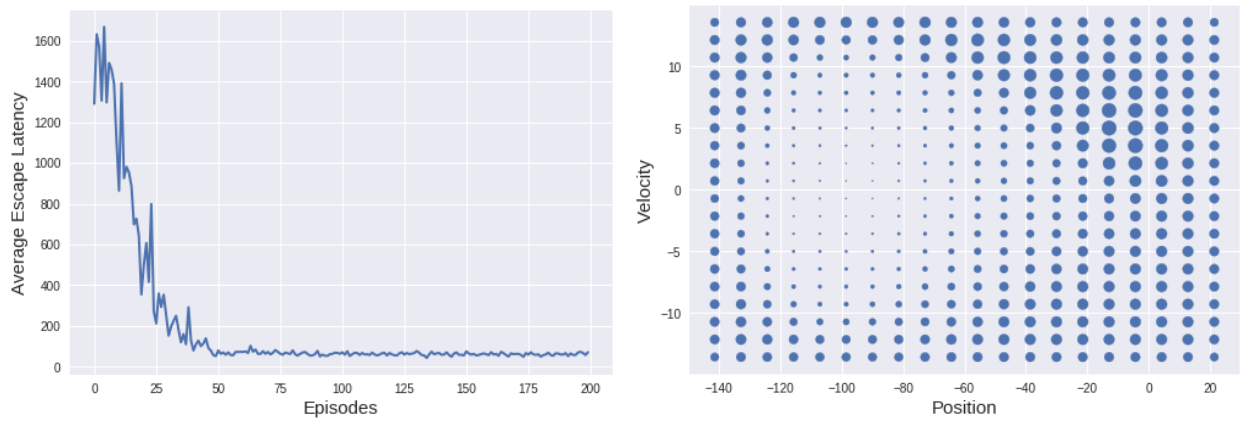
3.1 Learning Curve

Firstly we simulate 10 agents independently for 200 episodes, each with the following parameters:

- temperature $\tau = 0.01$, no decay
- eligibility trace decay rate $\lambda = 0.99$
- initial weights $w_{a,j} = 0.5$ (constant initialization for all weights)
- learning rate $\eta = 0.01$

We use these values as a starting point for further analysis by varying one parameter at a time. Figure 1a shows the average escape latency (averaged over agents) as a function of episodes, and Figure 1b visualizes the learned value function, where the size of the dots indicates the value of the states in different regions. We can immediately note that the value function is closely related to the energy of the car, being high in regions of either high kinetic energy or high potential energy.

We see that the agent learns to quickly solve the task (within 100 steps) after training for just 50 episodes. The agent behaviour (after learning) is visualized for one episode in Figure 2. We see a clear pattern in the actions that the agent takes wherein it steadily gains energy by pushing in one direction consistently before switching to the other direction for a few seconds and so forth until it escapes the valley. The state trajectory shows a clear outward spiral structure indicating this back-and-forth motion of increasingly larger amplitudes, and the energy plot clearly shows a steady increase in the car's energy until it escapes the valley after about 60 time steps.



(a) Learning curve *(b) Learned state-action values*

Figure 1: Results with running Sarsa(λ) over 200 episodes

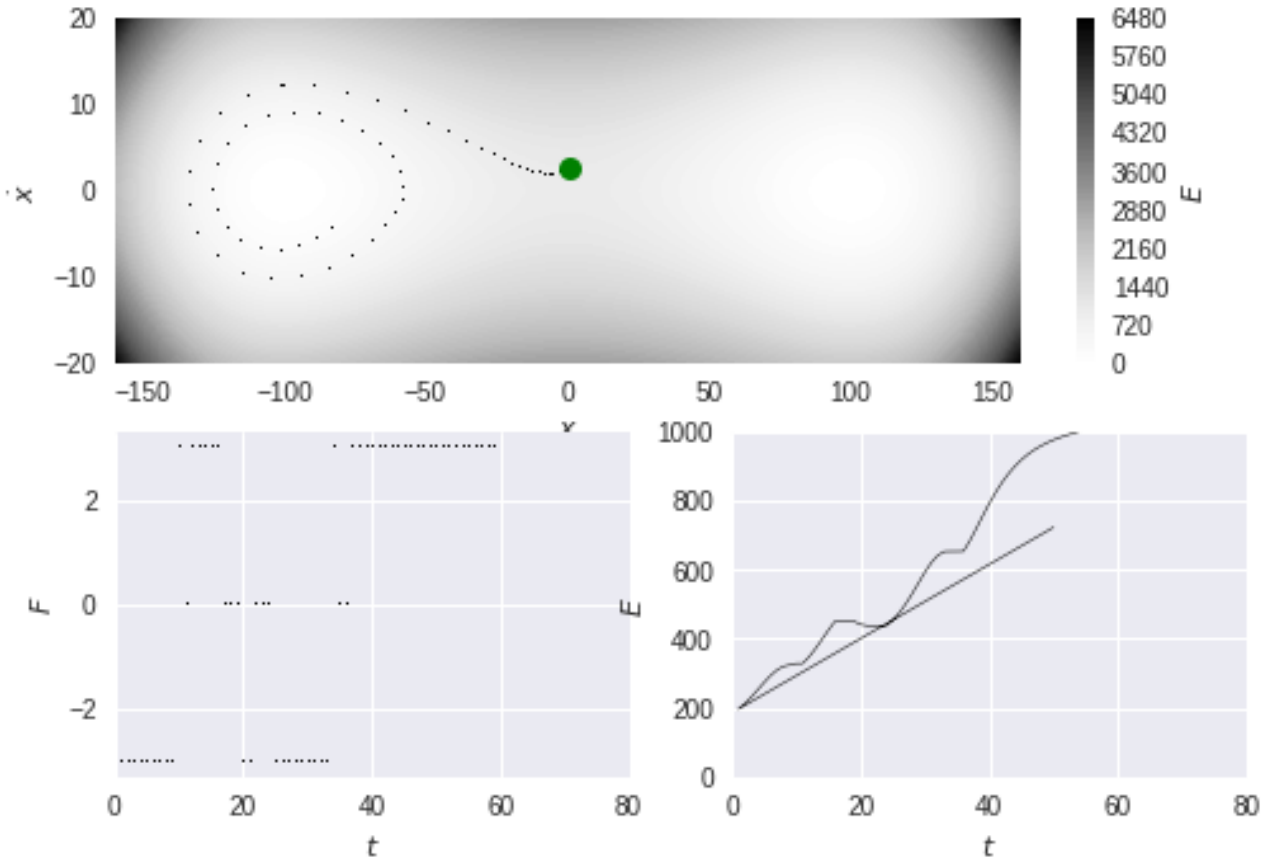


Figure 2: Demonstration of learned actions over one episode

3.2 Learned Policy

(a) Initial policy *(b) Policy after 25 episodes* *(c) Policy after 50 episodes*

(d) Policy after 75 episodes *(e) Policy after 100 episodes* *(f) Policy after 125 episodes*

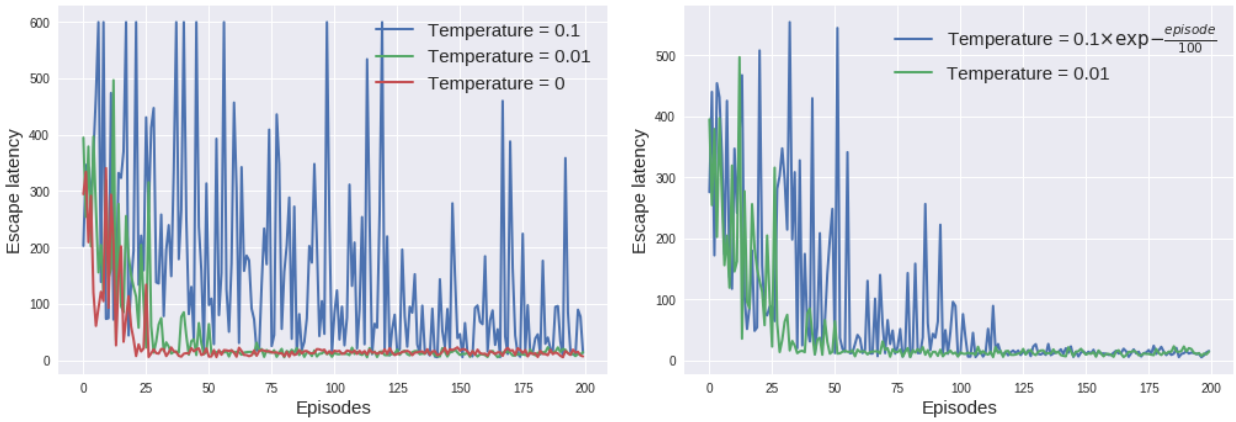
Figure 3: Evolution of the learned policy over 125 episodes

Directly visualizing the learned policy after a few episodes of learning is an illustrative way to study the learning progress. We have presented such a visualization in Figure 3. Here, the arrows indicate the direction of the highest valued action at any given state ($s = (x, \dot{x})$). If there is no arrow present at a state, it indicates that the highest valued action is to do nothing/stay in neutral gear. We see that the policy more or less converges after 50 episodes.

The learned policy agrees with our intuition. At high positive velocities (upper half of the state space), the best thing to do is to gain even more speed in the positive x-direction so that the mountain can be crossed. The same holds for the right half of the state space, where the car is already quite close to the peak, and a little more push in the right direction is likely to be sufficient for the car to escape. This holds true even if the car has a small negative velocity. However, for larger negative velocities, a positive force is not optimal if the car is a little distance away from the peak. More interesting is the lower left region of the state space where the car is near the bottom of the valley or even further left and is moving away from the mountain we wish to cross. In this case, the best action is to accelerate in the direction of motion in order to gain energy, which is exactly what the agent learns.

We also notice large gaps (the location of these gaps are not the same in every run) in the plot which indicate that the agent believes doing nothing is optimal in those regions. Accelerating in an appropriate direction would probably get it to the goal faster than doing nothing. However, after 125 episodes, the agent has still not learned this fact. This is probably due to a combination of two factors. First of all, we use a value of nearly 1 for λ, which means that a neutral action could be perceived as optimal as long as later actions are optimal. This is because the eligibility traces do not decay very quickly and therefore, it does not matter very much how soon the goal is reached. In other words, the escape latency has a negligible effect on the learning in such cases. This effect could also be exacerbated by a low state visitation frequency for some states. These regions are always outside the region where the car is allowed to start and after the agent has learned to solve the task quickly, some regions might become unlikely to be visited, which would lead to a slowdown of learning for such states. We see that these gaps get progressively filled in, meaning that it is only a matter of the agent getting more experience. However, this does not seem to affect the latency judging from the learning curve which flattens after 50 episodes.

3.3 Exploration temperature parameter



(a) Two different values for the temperature *(b) Decaying the temperature parameter*

Figure 4: Effect of the exploration temperature parameter

We now study the effect of varying the exploration temperature parameter τ. This parameter is used as a scaling factor in the soft-max over Q values for picking the next action. A high value for τ means

that differences in Q values have less effect on the policy probabilities. In the extreme case, for $\tau = \infty$, every action is equally likely irrespective of the learned value function and is thus no different from a purely random policy. On the other hand, for $\tau = 0$, the agent is almost purely exploitative, and therefore does not explore the state space effectively. However, some amount of exploration occurs because of the optimistic weight initialization ($w_{a,j} = 0.5$) and this was found to be sufficient for learning an optimal policy as we found in our experiments.

In Figure 4a, we compare the learning curves for a temperature of 0, 0.1 and 0.01. A higher value (1.0) led to a nearly random policy, and a very noisy learning curve. As could be expected, a higher temperature leads to more exploration at the price of a worse steady-state performance. The learning curve is much noisier for a higher temperature because the agent keeps picking a sub-optimal action frequently even though it probably has a better estimate for the value function, having explored a larger part of the state-action space. We can have the best of both worlds by starting with a high temperature and decaying it over time. This strategy is demonstrated in Figure 4b. Note that in this problem, exploration does not seem to be very crucial since the agent finds an optimal policy even with $\tau = 0$. However, this would be a serious issue in case of more complex problems with larger state spaces where rewards are harder to find and extensive exploration is needed.

3.4 Eligibility trace decay rate

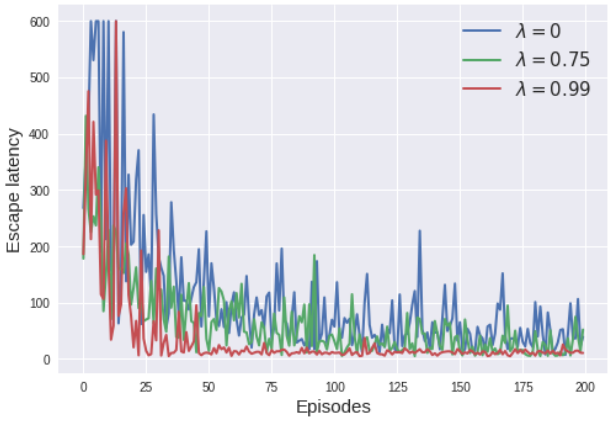
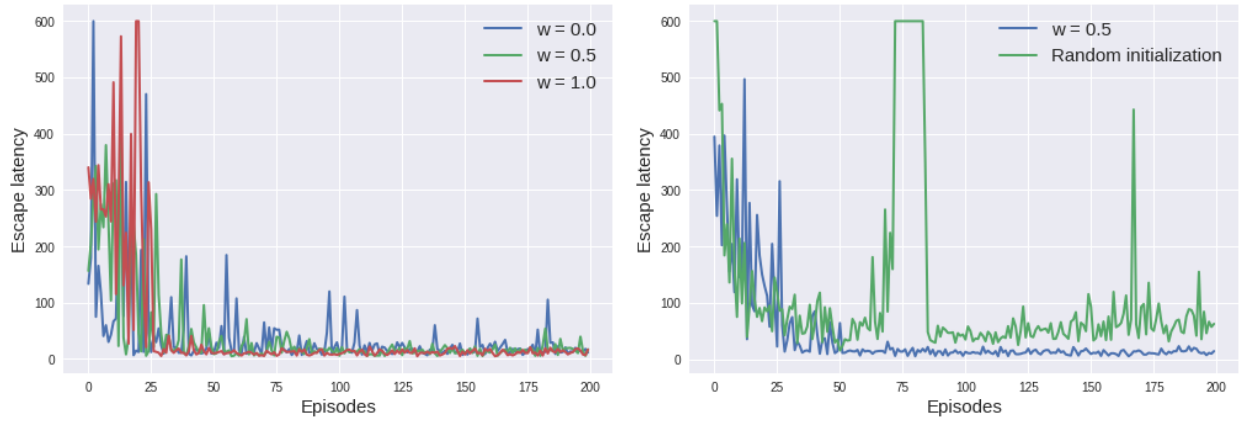


Figure 5: Effect of the eligibility trace decay rate

Figure 5 demonstrates the effect of the eligibility trace decay rate λ. The function of eligibility traces is to assign credit to actions taken even in the distant past. As such, λ represents a sort of memory in the system. Larger the value, further in the past credit for obtained rewards propagates. We see that for our problem, higher values for λ lead to faster convergence to a good policy. However, as explained before, this might lead to a sub-optimal policy because the escape latency loses influence on the learned policies. Thus, the agent might learn to reach the goal every time, but could end up taking unnecessarily long routes to the goal. Worse, it might learn to pick the wrong actions in some states as long as later actions are optimal. Thus, the choice of the decay rate needs to be done carefully and should depend on the task at hand.

3.5 Weight initialization



(a) Effect of the initial weights *(b) Random initialization of weights*

Figure 6: Effect of weight initialization

The initial weights have an interesting effect on the learning as we see in Figure 6a. Large initial weights lead to high Q-values for every action, an instance of exploration by optimistic initialization. This leads to more exploration as indicated by the initially noisy curves in the case of an initial weight of 1.0. On the other hand, for an initial weight of 0, the agent converges very quickly to a good policy but sometimes falters at later stages when it encounters some states for the first time. This kind of exploration strategy is better in some ways compared to using a temperature parameter. There is no need to find a decay schedule because here we are ensured to perform adequate exploration in the beginning and as learning progresses, the exploration dies off naturally.

Additionally, we investigated a different kind of weight initialization. Instead of having identical initial values for every weight, we initialized every weight with a uniform random distribution between 0 and 1. This kind of initialization is another way to ensure adequate exploration in the beginning, which dies out naturally over time. For our mountain car problem, this added stochasticity was too much, and the learning procedure was not very stable as can be seen in Figure 6b. We noticed that these disturbances died out with training on more and more episodes.

4 Conclusions

Here, we have presented a reinforcement learning solution using Sarsa(λ) for the mountain-car problem. Furthermore, we have implemented this RL algorithm using a biologically plausible neural network. We have also presented an analysis of the effects of the various hyperparameters of the learning algorithm along with an intuitive interpretation of the policies learned.