Reinforcement Learning: Q-Learning

JingYang Zeng, David Pomerenke
Advanced Concepts of Machine Learning (Kurt Driessens) 2020
Department of Data Science and Knowledge Engineering, Maastricht University

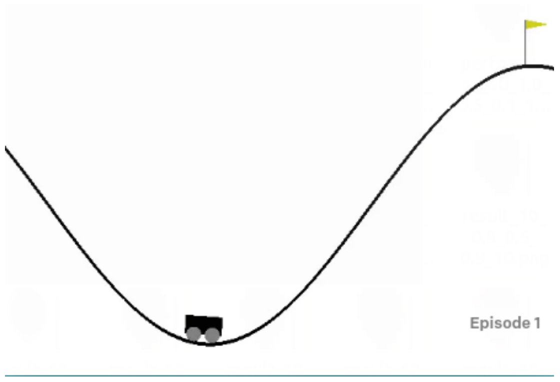

Fig. 1: Mountain car problem.

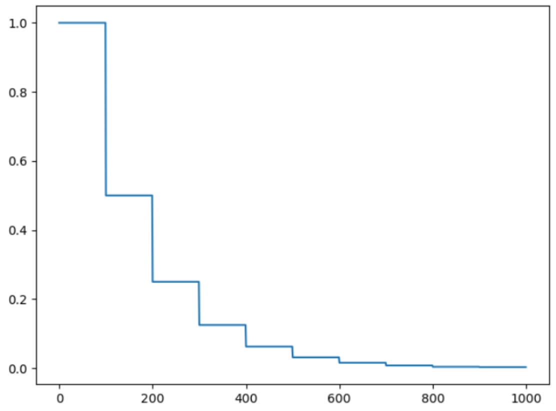

Fig. 2: Dynamic learning rate.

***Abstract*—We implement a Q-learning algorithm to solve the mountain car reinforcement learning problem. We observe good and robust results with a relatively simple implementation.**

I. Mountain car problem

The mountain car problem (see Figure 1[1]) from the *OpenAI Gym*[1] standard benchmark for reinforcement learning is based on an intermediate example from Andrew Moore[2]: A car is trapped between two mountains and does not have enought power to evade them directly. Thus, it needs to go back and forth between the mountains in order to accumulate enough speed.

The state space is continuous and consists of position and speed. The action is the pressure on the pedal.

II. Approach

We use *Q-learning*, that is the learning of Q-values that represent the combined probability and reward for every state-action pair, because it appears to us to be both the most effective and the simplest algorithm for reinforcement learning.

Q-learning is for discrete state spaces, so we discretize the state space into (a) 10^2 and (b) 100^2 regions.

Specifically, we use the *SARSA (state-action-reward-state-action)* algorithm, an online Q-learning algorithm, with *ϵ-greedy* as a policy for the selection of the next action to be investigated. ϵ-greedy means that with a probability of ϵ, a random action will be selected, and with a probability of $1-\epsilon$, an action will be selected in accordance with the learned Q-values $P(a_0|(u,v)) = \frac{e^{Q((u,v),a_0)}}{\sum_{a\in A} e^{Q((u,v),a)}}$. Even though ϵ-greedy does not have a sound mathematical foundation, it is known to be an effective policy for choosing an action.

[1]https://gym.openai.com/envs/MountainCar-v0/

A. Dynamic learning rate

Although the reinforcement learning problem can be solved by some brief and efficient algorithms, we still meet problems when we implement those algorithms. One of the problems is using a static learning rate or dynamic learning rate. We assume that the learning rate should decay when each episode starts so that the learning can converge in a certain episode.

Therefore, we set a dynamic learning rate of

$$max(\text{min learning rate}, \text{initial learning rate} \cdot 0.5^{\frac{i_{\text{episode}}}{n_{\text{episodes}}}})$$

rather than a constant value. However, after the experiments, we found there is not an obvious difference between the static and dynamic learning rate (see Figure 2) when the reinforcement learning problem can be properly solved. In other words, learning with a decay learning rate has a more stable performance, but it does not have a significant impact in this simple experiment.

III. Implementation

We implement the algorithm in Python using a template by Kurt Driessens. Our dependencies are *gym*, *numpy*, *matplotlib* and *seaborn*, and the code can be run by `python Qlearning.py`, which will eventually reproduce all figures for this report.

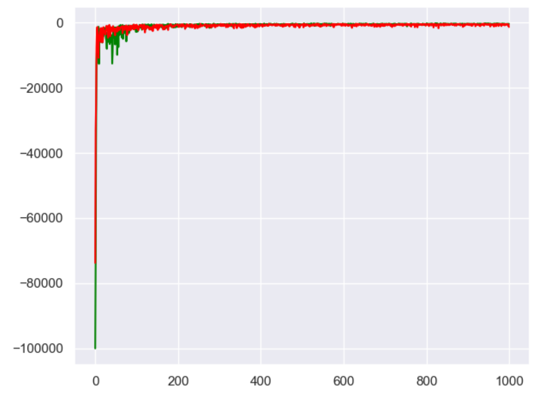

Fig. 3: Decay learning: Performance with (red) and without (green) decay learning rate. USing the decay learning rate yields more stable results.

IV. Experiments

We perform multiple trainings with a small number of training episodes of 10 and 100. Here, we vary the degree of discretization between 10^2 and 100^2, the discount factor of the problem between 0.8, 0.9, and 1.0, and ϵ between 0.1, 0.5 and 0.9. A discount factor of 1.0 seems to make sense to us, since the problem a clear end, and since the reward is only obtained at the end in the case that the car surpasses the mountain. A smaller discount factor, however, might set an incentive to find the quickest solution. Some interesting results are reported in Figure 5. We arrive at similar learned Q-values as for the following experiment, but there is no clear convergence.

We also perform one experiment with a high number of training episodes of 1000 in order to observe the influence on the convergence. The convergence is indeed good for this setup, see Figure 4.

V. Discussion

In comparison to other machine learning techniques, the Q-learning algorithm is easy to implement, and also straight-forward to extend with SARSA and ϵ-greedy.

It is also not too hard to get it to learn something useful. The learned Q-values are similar for a small and a high number of training episodes, and for different choices of the parameter ϵ. The discretization coarsity does not have an influence on the results. Even though we can achieve good results with a small number of training episodes around 10, we only obtain good convergence properties for a much larger number of training episodes around 1000.

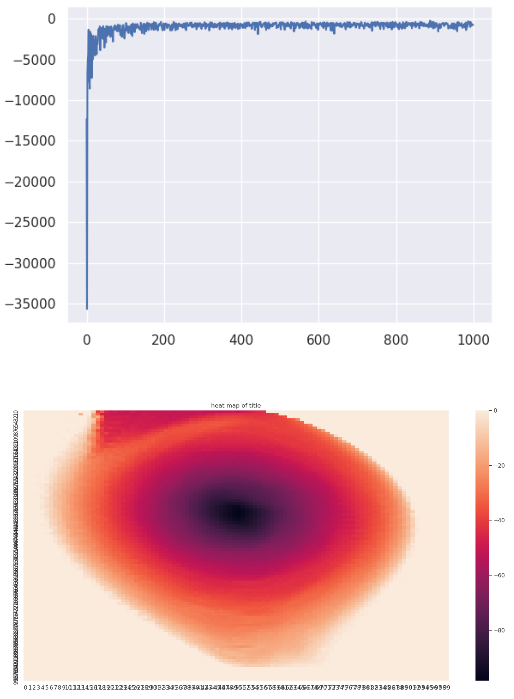

Fig. 4: Training performance and learned Q-values for a setting of 1000 episodes, discretization 100^2, discount factor 1.0 and $\epsilon = 0.5$.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[2] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-209, Nov. 1990. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-209.pdf.

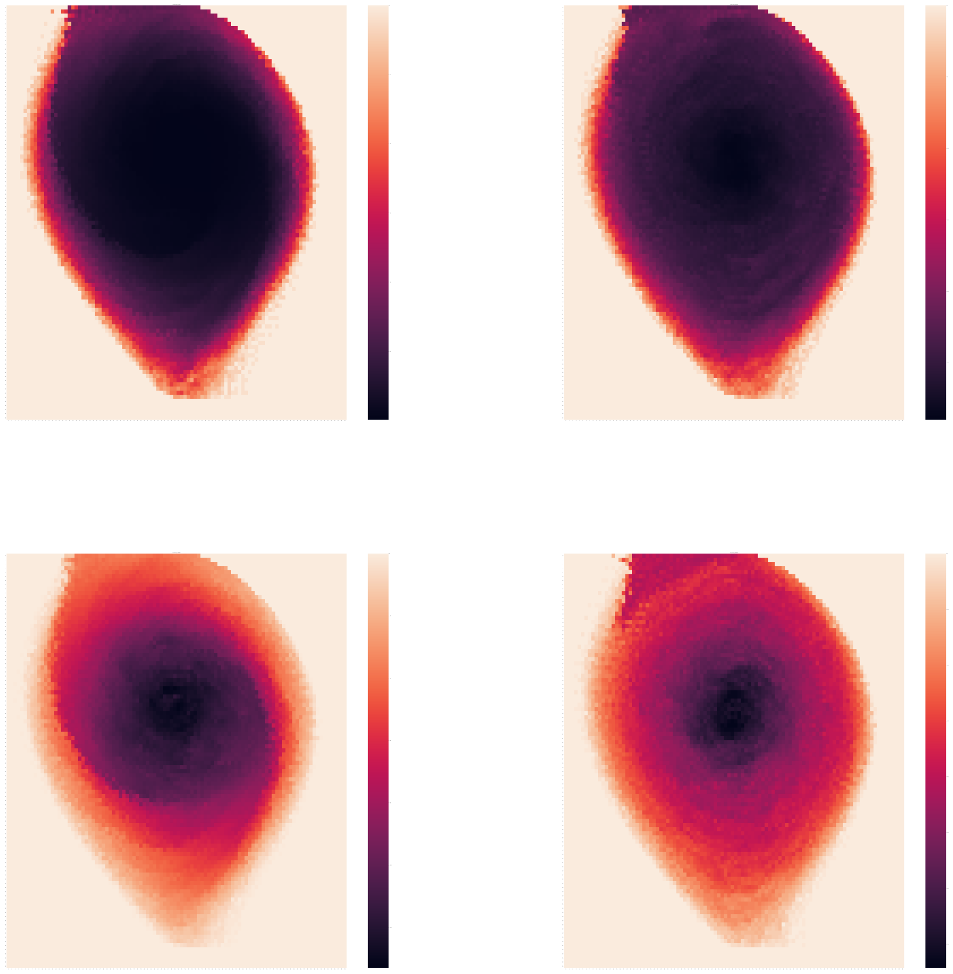

Fig. 5: Different intensity patterns emerge for different parameters, but the shape of the learned data remains the same. At the top, a discount factor of 0.9 has been chosen, at the bottom one of 1.0. $\epsilon = 0.1$ at the left, and $\epsilon = 0.5$ at the right. The number of training episodes is 100 and the level of discretization is 100^2 for all four settings.