Deep RL Software project

Amine Oueslati

Eötvös Loránd University - AI specialization

Abstract. This report is for explaining the process of implementing reinforcement learning solution for the Acrobot-v1 environment.

1 Introduction

In this task, the "OpenAI Gym" environment used is the "Acrobot-v1", and the agent will be learning through SARSA algorithm with experience replay.

2 Related work

"OpenAI Gym" is a toolkit that allows testing of different reinforcement learning algorithms on various simulated environments with the overall goal of maximizing the reward from interacting with that environment.

There are different available environments, from the simple grid-world type, rendered on the command line, to ones with more complex dynamics, and then the more difficult environments with many compound variables.

3 Environment

Acrobot is a 2-link pendulum with only the second joint actuated. Initially, both links point downwards. The goal is to swing the end-effector at a height at least the length of one link above the base. Both links can swing freely and can pass by each other, meaning that they don't collide when they have the same angle.

4 Methods

At first, the Q-table is randomly initialized, and as the agent interact with the environment the Q values will be learned through small updates.

Thus, one can think of the values derived from the Q function as the targets that the approximated Q values will be pushed towards at some learning rate.

However, the agent is selecting the actions greedily based on the randomly initialized Q-table and not exploring the other actions.

To fix this issue, a new variable epsilon is introduced to balance the probability of exploration and exploitation of the environment.

nevertheless, this variable should be reduced as the agent learns a more optimal Q-table.

This algorithm could be implemented as a neural network, and since the actions are discrete, it makes more sense to represent them as one-hot vectors. thus, the actions will be encoded to nA (number of actions) columns.

To build the network, a dense layer should be applied in the beginning, acting as a hidden layer, with a certain number of nodes. Then, a second dense layer is required to transform the result to match the number of available actions. This will be the vector of Q-values for the input state. To get the Q-value for a specific input action, this vector should be multiplied by a one-hot representation of the action. Subsequently, the result is summed along the columns to get a single Q-value. the mean squared error is used for the loss function where the target is the result of the Q-function using the reward added to the maximum of the next state's Q-values, which is derived from the model.

This diagram can illustrate the topology of the network:

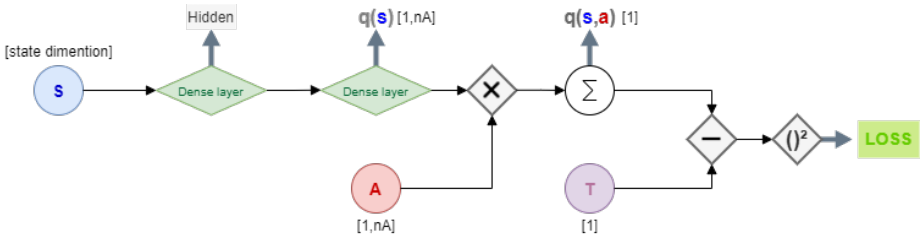

Fig. 1. neural network topology

During this actual process, the training is done with a single tuple of step values which are used to run the optimizer. Thus, each experience is essentially used just once and forgotten when the next step is reached.

This approach creates a bias towards training on common experiences that the agent sees. This might "starve-out" the training on rare experiences which can be more important.

To solve this issue and enhance the training speed of the agent, a technique called experience replay is used. Meaning that each experience will be stored and the agent will sample a number of experiences to train on them at once. This will give the rare experiences more chance to be trained on in case the agent doesn't see them for a while.

To implement this method, at each time step, the experience tuple that the agent just saw is added to a buffer with a fixed length. Then the agent could be trained on a randomly sampled batch from the buffer.

when this buffer is full, the old experiences are deleted to make room to the new ones.

5 Experiments

During these experiments the agent will learning through 500 episodes.

5.1 Learning rate alteration

Due to the complexity of the environment, the total reward will be unstable. However, with a convenient learning rate, the results will converge to a small range.

High learning rates causes drastic updates which leads to divergent behaviours. Thus the agent won't be capable of learning.

These facts can be easily seen in figure2. The highest total-rewards was with a relatively moderate learning rate.

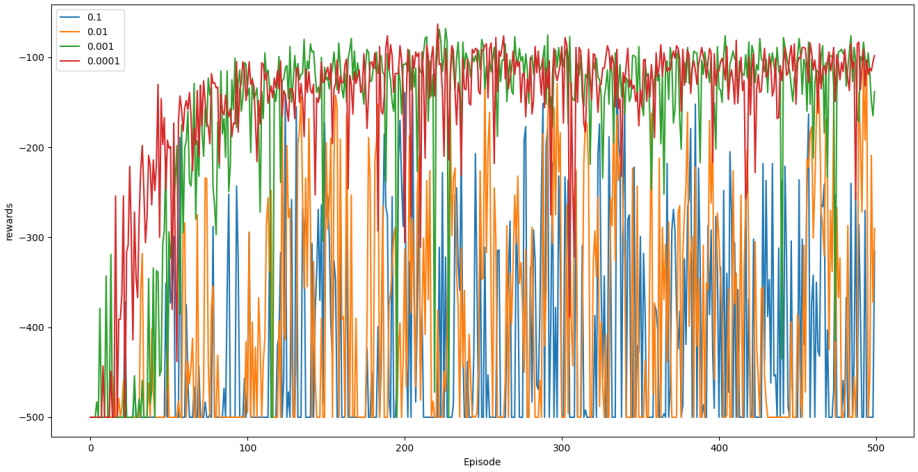

Fig. 2. Total rewards for different learning rates

5.2 Discount factor alteration

The discount factor quantifies how much importance given for future rewards. Gamma varies from 0 to 1. If it is closer to zero, the agent will tend to consider only immediate rewards. If it is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.

In this environment, having a higher discount rate give rise to a higher total-rewards.

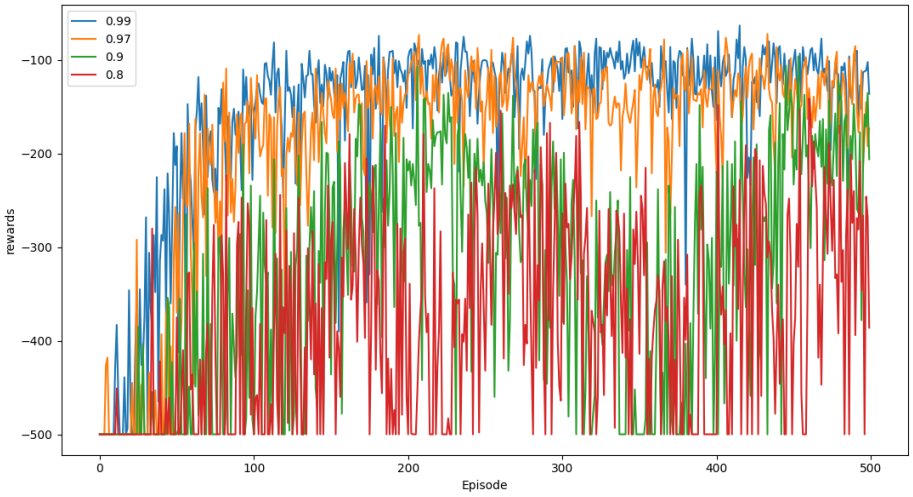

Fig. 3. Total rewards for different discount factors

5.3 Epsilon alteration

This parameter is introduced to balance the exploration of the full set of actions, and selecting actions from the Q-learning policy using the Q-table. Thus, Epsilon will represent the probability of prioritizing an exploratory action over a policy action.

This variable is decayed through time. Hence, as the agent learns a more optimal q-table at each episode, epsilon will be multiplied by a decay factor.

The effect of the epsilon decay factor is highlighted in figure4.

If the decay factor is low, epsilon will be near zero after few episodes, thus the agent won't be able to explore all the action space, and will keep choosing the greedy action from the randomly initialized Q-table.

However, if it is too high, the probability of selecting the greedy action will stay minimal and the random choices will be more frequent. That's why an optimal decay factor should be set to ensure the maximum rewards.

In addition, this variable cannot be less than a minimum threshold to prevent it from vanishing.

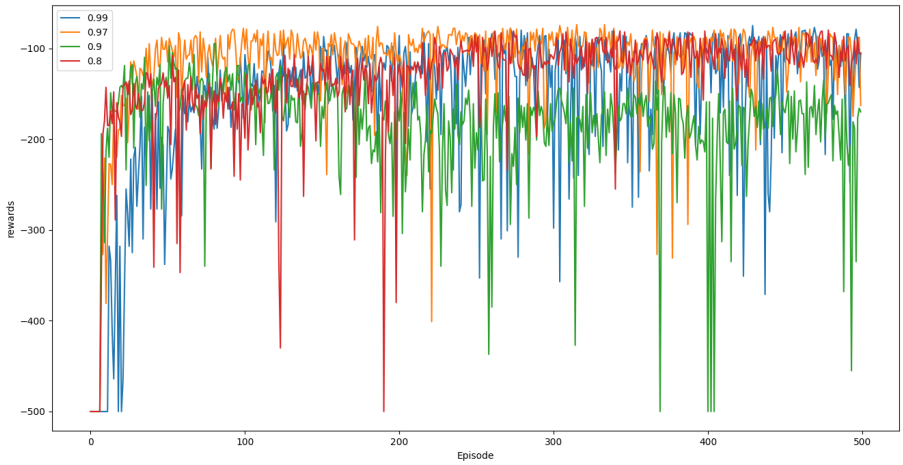

Fig. 4. Total rewards for different Epsilon decays

6 Conclusion

In this report, multiple experiments were conducted to study the behaviour of SARSA algorithm agent with experience replay in the Acrobot environment.

The complexity of the state and action space made the task harder for the agent than the one from the frozen lake. However, with the experience replay technique and the right hyper-parameters it was able to get remarkable rewards.

This algorithms can be improved by a technique called prioritized experience replay meaning that experience replay will not be randomly sampled. Instead, each old experience will have a probability to be sampled in the training batch.

CODE : https://github.com/amine-oueslati/DRL/blob/main/acrobotDeepQNetwork.py