Reinforcement Learning Agent

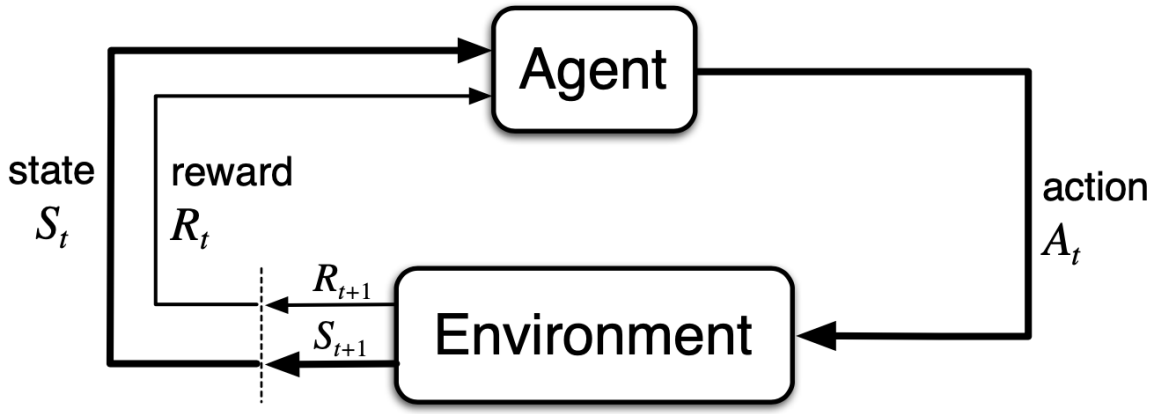
Filip Masár

May 9th 2021

Introduction

Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. [1]

Basic idea is like this. Agent interacts with the environment by taking an action from the state it is in. Then it receives reward for that action and a new state from the environment based on the action it took.



[2] Figure 3.1 of "Reinforcement Learning: An Introduction, Second Edition"

I am using environments provided by Open AI gym. My goal is to solve the CartPole-v1 environment and the LunarLander-v2 environment. For more details see section Environments

Both environments have discrete action space. Common approach in reinforcement learning is to learn the action value function, which, given state and action, estimates return (weighted sum of rewards) from that state if we take that action.

For environments with discrete finite state space, we can learn all these state-action pairs. Those methods are called tabular methods and could be solved with algorithms such as Monte Carlo (estimates returns from whole episode) or Q-learning, Sarsa (temporal difference methods - adjust estimates every time we take an action). Or there is a possibility also to use n-step methods which is somewhere in between Monte Carlo and temporal difference methods - adjust estimates every n steps

Our environments have continuous state space. Therefore I've chosen to implement a Deep Q-learning algorithm which uses a neural network to predict action-value function.

Algorithm

There are many approaches to train the agent based on the environment it is in. I am implementing a deep Q learning algorithm with experience replay buffer. It is:

- temporal difference method - adjusting estimates during episode
- off policy method - different behaviour policy and target policy

The following pseudo code was originally used for training in Atari environments where states are images. Our states are just a list of numbers, so I am not doing any preprocessing of states. Otherwise I am following the steps of this pseudo code

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

[3] Algorithm 1 of the paper "Human-level control through deep reinforcement learning"

Action-value function describes behaviour policy. Target action-value function describes target policy.

I have implemented these functions as 3 layer perceptrons. Number of input neurons is the number of values the state is described by. Number of output neurons is the number of actions we can take. This way I can give this model a state and it will return for every action expected return from that state following that action.

Environments

I am working with environments provided by the open AI gym. Basic documentation of the gym can be found here https://gym.openai.com/docs/. As stated on this page:

> *"Fortunately, the better your learning algorithm, the less you'll have to try to interpret these numbers yourself".*

"these numbers" refers to the representation of actions and space. Therefore they are not really relevant to me. But those representations could be found in the source code of the environment.

CartPole-v1 https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py
LunarLander-v2 https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py

Trained models

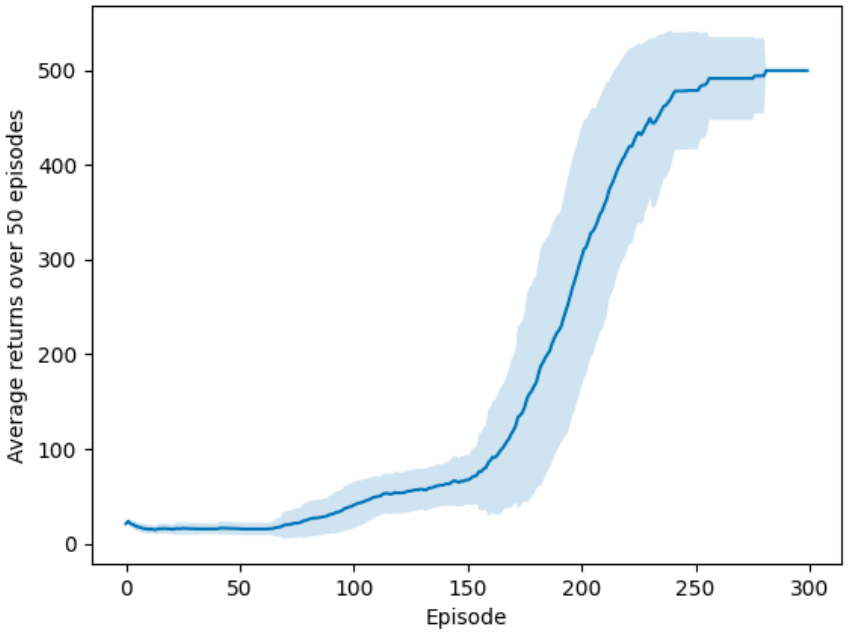
Below are the hyperparameters I've used for each environment. Plots show the average returns over 50 episodes during training (dark blue) and standard deviation of returns (light blue).

LunarLander-v2 is considered to be solved if agents can obtain 200 points during the episode. I have tested my agent multiple times and it is receiving on average around 100 points. I didn't have the hardware capacity to train it further. I believe that if it was trained on 10000 episodes or even more (not 3000) that it will learn this environment perfectly. It is just a matter of more training 💪

Both models are provided in models/ directory and can be loaded to see their performance (see README.md)

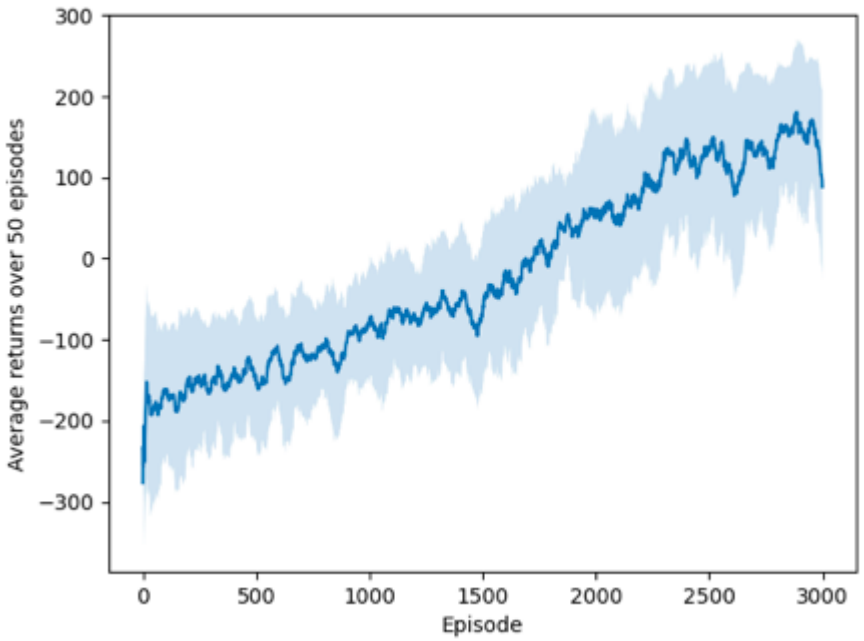
CartPole-v1

episodes=300,
batch_size=32,
gamma=1.0,
epsilon_start=0.5,
epsilon_final=0.1,
epsilon_final_at=300,
target_update_freq=0,
learning_rate=0.001,
hidden_layer_size=50



LunarLander-v2

episodes=3000,
batch_size=32,
gamma=0.99,
epsilon_start=0.8,
epsilon_final=0.1,
epsilon_final_at=2500,
target_update_freq=10,
learning_rate=0.0001,
hidden_layer_size=100



Conclusion

We can now train an agent in environments with a discrete set of actions. State has to be described as a 1D list of numbers. This can be extended for example to states being images. The network, that is approximating the action value function, would just need to change its architecture (e.g. some convolutional layers in the beginning). This way we can train an agent that is even capable of playing Atari games.

References

To understand the key concepts in reinforcement algorithm, please check out https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

[1] Hu, J.; Niu, H.; Carrasco, J.; Lennox, B.; Arvin, F. (2020). "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning". *IEEE Transactions on Vehicular Technology*. **69** (12): 14413-14423.

[2] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2014. *https://web.stanford.edu/*, https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[3] Mnih, Volodymyr. “Human-level control through deep reinforcement learning.” *https://web.stanford.edu*, 2015, https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf