CS6482 Deep Reinforcement Learning

Assignment 3: Policy Gradient Network for the Cartpole

Elsa Anza Martín (21272808)

Albert Martínez (21251266)

8th May 2022

University of Limerick

Lecturer: J.J Collins

Table of Contents

Overview of Policy Gradient approaches (2 marks)

The policy gradient method is a reinforcement learning technique that relies on optimising a parameterized policy for an expected reward (long-term cumulative reward) via gradient descent. They do not suffer from many of the problems of traditional reinforcement learning methods, such as the lack of guarantees on the value function, intractability due to indefinite state information, and complexity due to continuous states and functions.

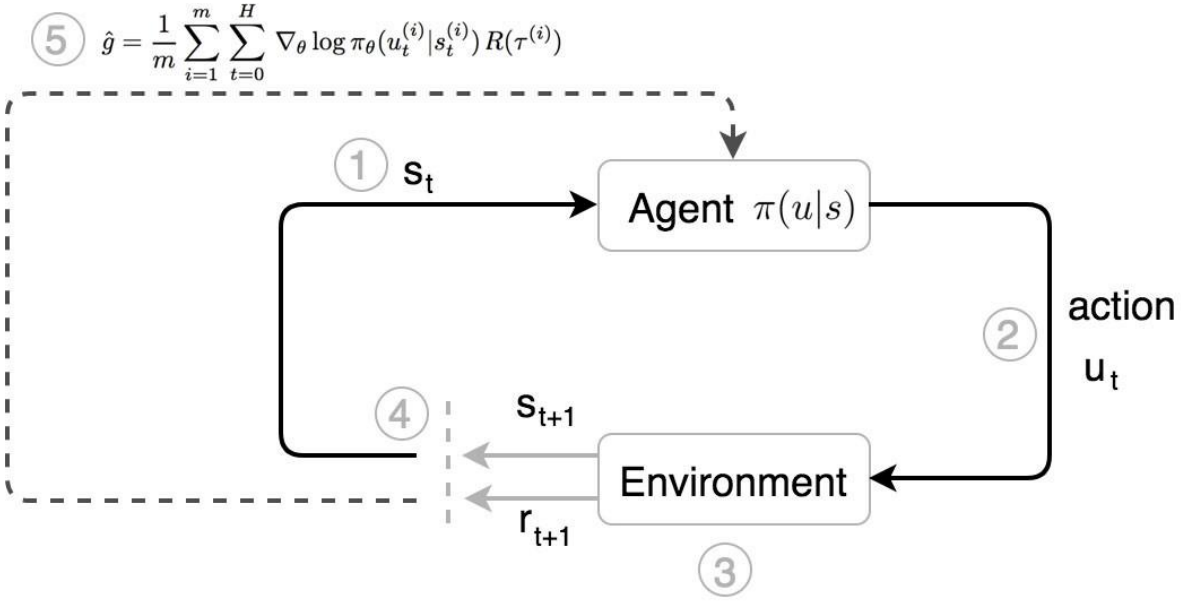


Image showing the relations in the Policy Gradient system, taken from Hui (2020).

REINFORCE algorithm.

This is the approach we're exploring in this project and its original form consists in a Monte Carlo Policy algorithm that uses an alteration of SGD, the gradient ascent. The update of the weights involves just the action taken in each timestep.

REINFORCE with Baseline

The other major approach is REINFORCE with Baseline, a generalisation of the previous one explained, including another action using a baseline b(s) included in the formula

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \Big(q_\pi(s,a) - b(s)\Big) \nabla \pi(a|s,\boldsymbol{\theta}).$$

This can make learning occur faster.

Actor Critic Methods

They are similar to Reinforce algorithms with baseline except they bootstrap. By representing the policy function independently from their value function we can cause policy to decide which action to make while the value function improves the training process for the value function. They do a one-step return. Some variations of these are Ons Step Actor critic and Actor–Critic with Eligibility Traces.

Compare and contrast PG v DQN for the cartpole (1 mark)

The difference between DQN and PG is that the first one is value-based and the second one is policy-based.

The value-based methods optimise the function Q to obtain the preferences of choice of actions by the agent. In DQN neural networks are used for this purpose.

The policy-based methods optimise the policy directly without computing the Q function using a neural network.

So, in both cases we use neural networks, but in one case we need to compute the value function and in the other one we don't.

PG offers some advantages versus DQN in Cartpole. For example, we no longer have to worry about coming up with an environment scanning strategy, like the epsilon-greedy policy. Now, the neural network of a policy-based algorithm will return the probabilities of each possible action, and the exploration is done automatically when choosing an action following the distribution of probabilities (it does not need to be explicitly included as a step in the algorithm). Furthermore, by obtaining the action in this way, there is no need to add "trick"" to the algorithm (such as experience replay or target network) to improve the stability of the neural network training.

The Environment Data Set (1 marks)

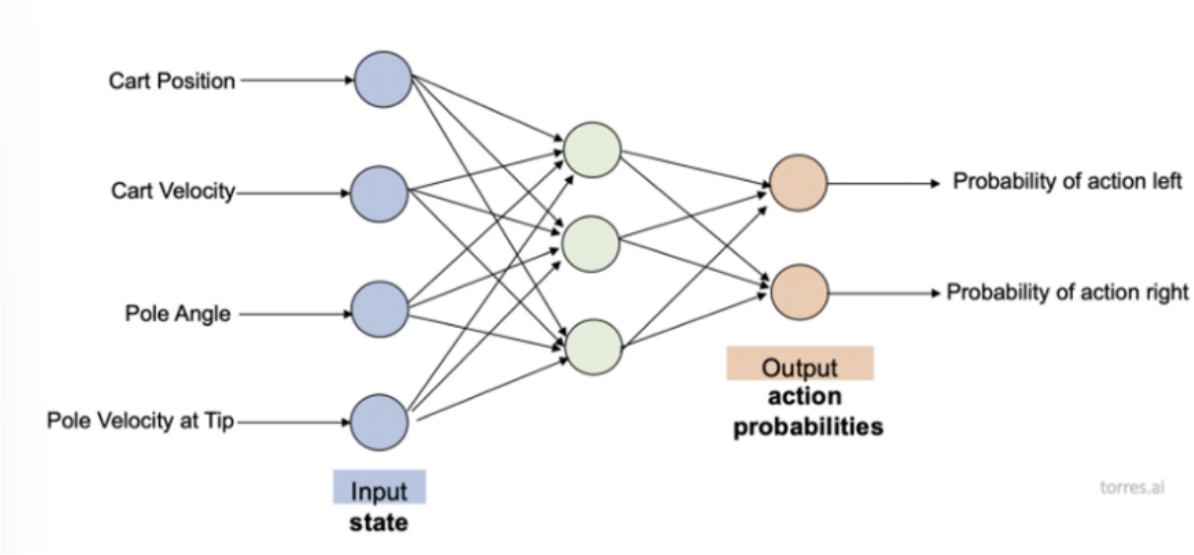
Cart-Pole, which can be found in the Gym library, consists of simulating a cart that moves horizontally along a frictionless track and that has a post connected by a joint at the top. The post starts upright and the goal is to prevent it from falling to one side or the other. The way to keep the pole in balance is to balance it by moving the cart to the left or right.

The environment provides a +1 reward for each additional time step the pole remains upright. The episode ends when the pole leans too far (when it reaches a certain number of degrees from vertical on either side), or when the cart has moved too far (beyond a point on both sides of the centre where it has moved). The system is controlled with two possible actions: apply a force that pushes the cart to the left or apply a force that pushes the cart to the right.

The state that this environment returns in each time step is a tuple of 4 values that contain its position (position), its velocity (velocity), the angle of the post (angle) and the angular velocity of the post at the tip (angular velocity).

The network structure and other hyperparameters (1 marks)

The network structure is based on the RELU structure. It has a state size of 4 (cart Position, Cart Velocity, Pole Angle and Pole Velocity at Tip), an action size of 2 (Probability of action left and Probability of action right) and a hidden size of 32.



The Loss Function (2 marks)

The loss function is based in the formula $\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t$. We introduce the negative sign of the loss function to set up the problem so that we minimise an objective function instead of maximising it, since it works well with PyTorch's built-in optimizers (using stochastic gradient descent).

Specifically, the loss function that has been programmed in this code requires the probabilities of each action performed, $\pi_\theta(a_t|s_t)$. On the other hand, it requires the expected returns, G_t.

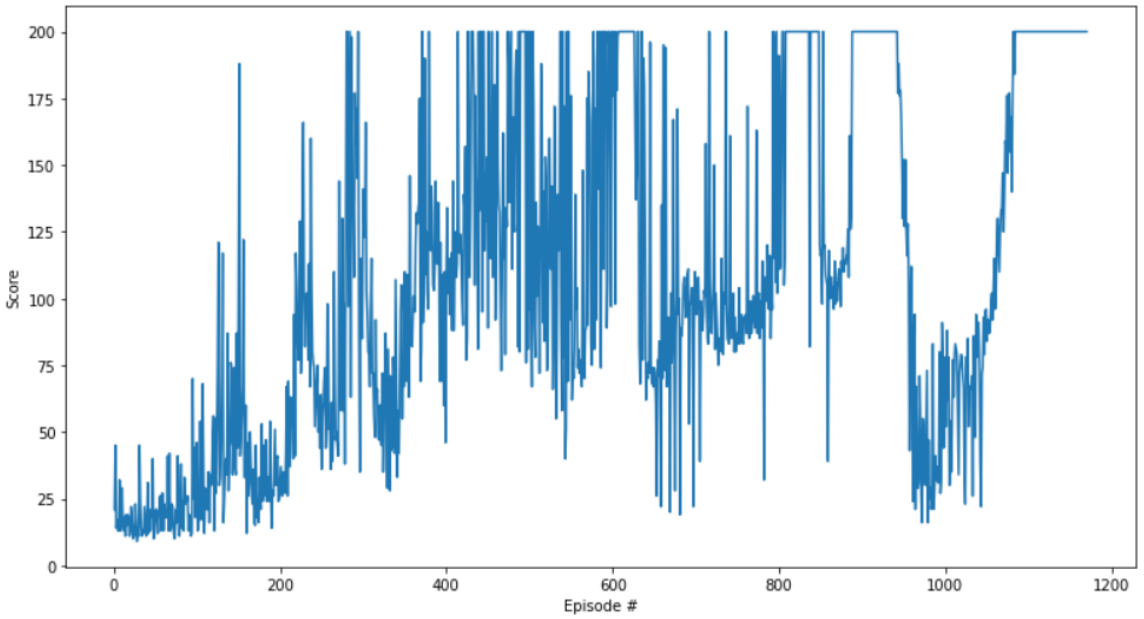
The optimiser (1 marks)

The gradient ascent has been implemented as an optimization problem using a loss function.We have used an Adam optimiser which is a stochastic gradient descent method to

do backpropagation in order to update network weights. Adam optimiser is effective in only needing to use a small amount of memory. It is a great way to solve significant problems with a lot of data and parameters (Harry, 2020). Adam optimisation algorithm combines the benefits of both Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), which are also a variant of stochastic gradient descent (Brownlee, 2021).

Results with plots (1 marks)

After 1070 episodes, we were able to solve the environment. Here we show the plots and results after the training.




```
Episode 100     Average Score: 20.22
Episode 200     Average Score: 41.21
Episode 300     Average Score: 85.37
Episode 400     Average Score: 91.89
Episode 500     Average Score: 143.20
Episode 600     Average Score: 125.37
Episode 700     Average Score: 122.81
Episode 800     Average Score: 103.71
Episode 900     Average Score: 160.88
Episode 1000    Average Score: 126.33
Episode 1100    Average Score: 111.43
Environment solved in 1070 episodes!    Average Score: 195.44
```

Evaluation of the results (1 marks)

As shown in the results above, we can see how the graph shows the learning trajectory. As the policy gradient is constantly searching for the optimal direction on the ascent gradient, it improves the policy to achieve a better score.

It took only 1070 episodes with an average score of 195,44 and 2 minutes to learn how to achieve it.

With the REINFORCE algorithm, the method updates the policy parameters by taking random samples. This introduces high inherent variability in probabilities and expected reward values, because during training each trajectory can deviate from each other to a large extent. Consequently, this will generate noisy gradients that will cause unstable learning with slow convergence and may bias the learning in a non-optimal direction.

In the exposed method we have seen that we have to wait for full trajectories to be completed before we can start training. This fact is not a problem for short episodes, such as those in the Cart-Pole setting. But in environments like Pong, where each episode can be thousands of frames long, both the episodes and the paths must be very long to be useful for training.

References

Kang, C. (2021). REINFORCE on CartPole-v0. [online] Chan's Jupyter. Available at: https://goodboychan.github.io/python/reinforcement_learning/pytorch/udacity/2021/05/12/REINFORCE-CartPole.html [Accessed 7 May 2022]

Sutton, R.S. and Barto, A. (2018). Reinforcement Learning: An introduction. Cambridge, Ma ; London: The Mit Press.

Hui, J. (2020). RL — Policy Gradient Explained. [online] Medium. Available at: https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146 [Accessed 8 May 2022].