OUSMANE THIONGANE

SAPIENZA
UNIVERSITÀ DI ROMA

Reinforcement Learning

Assignment 3 – 1-step Actor Critic & CarRacing-v2

Matricula: 2184503

November 22th, 2024

ASSIGNMENT 3 – 1-STEP ACTOR-CRITIC & CARRACING-V2

I – Theory

1. Let's consider an environment with 2 possible actions and a 2D-state representation ($x(s) \in R^2$). We have the following policy and action-state value function approximators for the 1-step Actor-Critic Algorithm:

$$\pi_\theta(a=1|s) = \sigma(\theta^T x(s)) = \frac{1}{1+e^{-(\theta^T x(s))}}$$

$$Q_w(s, a=0) = w_0^T x(s)$$

$$Q_w(s, a=1) = w_1^T x(s)$$

Given:

$$w_0 = (0.5, 0)^T, \qquad w_1 = (1, 0)^T$$

$$\theta_0 = (0.6, 1)^T$$

$$\alpha_w = \alpha_\theta = \alpha = 0.5$$

$$\gamma = 0.8$$

And the following transition:

$$x(s_0) = (0,1)^T, \qquad a_0 = 1, \qquad r_1 = 1, \qquad x(s_1) = (1,0)^T, \qquad a_1 = 0$$

Let's compute the new values of w_0, w_1 and θ after the transition.

As we have seen in class, the 1-step Actor-Critic Algorithm is defined as follows:

Algorithm 1-step Actor-Critic

```
Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to 0)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \theta)$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S', w) - \hat{v}(S, w)$
        $w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S, w)$
        $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla \ln \pi(A|S, \theta)$          (if $S'$ is terminal, then $\hat{v}(S, w) \doteq 0$)
        $I \leftarrow \gamma \cdot I$
        $S \leftarrow S'$
```

Let's follow the steps of the algorithm. At first, we have:

$$Q_w(s_0, a_0) = Q_w(s_0, 1) = w_1^T x(s_0) = 1 \times 0 + 0 \times 1 = 0$$

$$Q_w(s_1, a_1) = Q_w(s_1, 0) = w_0^T x(s_1) = 0.5 \times 1 + 0 \times 0 = 0.5$$

We then compute the value of δ:

$$\delta = R + \gamma \hat{v}(s_1, w) - \hat{v}(s_0, w)$$

$$\Rightarrow \delta = r_1 + \gamma Q_w(s_1, a_1) - Q_w(s_0, a_0)$$

$$\boxed{\Rightarrow \delta = 1 + 0.8 \times 0.5 - 0 = 1.4}$$

Now we must calculate the updates of the weights:

– For w_0:

$$w_0 = w_0 + \alpha_w \delta \nabla \hat{v}(S, w)$$

$$\Rightarrow w_0 = w_0 + \alpha \delta \nabla Q_w(s_0, a_0)$$

$$\Rightarrow w_0 = w_0 + \alpha \delta x(s_1)$$

$$\boxed{\Rightarrow w_0 = (0.5, 0)^T + 0.5 \times 1.4 \times (1, 0)^T = (1.2, 0)^T}$$

– For w_1:

$$w_1 = w_1 + \alpha_w \delta \nabla \hat{v}(S, w)$$

$$\Rightarrow w_1 = w_1 + \alpha \delta \nabla Q_w(s_1, a_1)$$

$$\Rightarrow w_1 = w_1 + \alpha \delta x(s_0)$$

$$\boxed{\Rightarrow w_1 = (1, 0)^T + 0.5 \times 1.4 \times (0, 1)^T = (1, 0.7)^T}$$

Next, we compute the update for θ:

$$\theta = \theta + \alpha_\theta I \delta \nabla \ln \pi(A|S, \theta)$$

To compute θ, we need to compute the gradient of the log policy:

$$\ln \pi(A|S, \theta) = \ln\left(\frac{1}{1 + e^{-\left(\theta^T x(s_0)\right)}}\right) = -\ln\left(1 + e^{-\left(\theta^T x(s_0)\right)}\right)$$

$$\nabla \ln \pi(A|S, \theta) = -\nabla \ln\left(1 + e^{-\left(\theta^T x(s_0)\right)}\right)$$

By applying the chain rule, we got:

$$\nabla \ln \pi(A|S, \theta) = \frac{1}{1 + e^{-\left(\theta^T x(s_0)\right)}} \times \nabla\left(1 + e^{-\left(\theta^T x(s_0)\right)}\right)$$

$$\Rightarrow \nabla \ln \pi(A|S, \theta) = -\frac{e^{\theta^T x(s_0)} x(s_0)}{1 + e^{\theta^T x(s_0)}} = -\frac{e^1}{1 + e^1}(0, 1)^T = (0, -0.731)^T$$

That finally gives us:

$$\boxed{\Rightarrow \theta = (0.6, 1)^T + 0.5 \times 1 \times 1.4 \times (0, -0.731)^T = (0.6, 0.489)^T}$$

In conclusion, the updated values for w_0, w_1 and θ are:

$$\boxed{w_0 = (1.2, 0)^T \quad w_1 = (1, 0.7)^T \quad \theta = (0.6, 0.489)^T}$$

II – Practice

Let's solve the CarRacing-v2 environment with a continuous action space (e.g. 3 actions: steering, gas and braking):

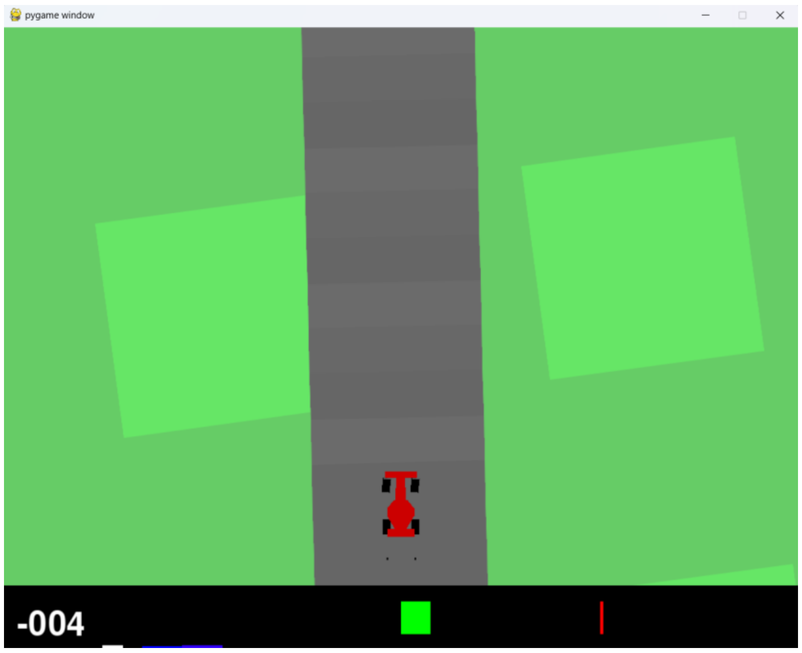

Figure 1 – CarRacing-v2 environment display

Among the proposed algorithms, I chose to implement the **Proximal Policy Optimization (PPO) algorithm.**

As stated in the paper, the PPO algorithm is a policy gradient method, which alternate between sampling data by interacting with the environment and optimizing a surrogate objective function by using stochastic gradient ascent. The PPO is more effective for some simpler problems than Q-learning or vanilla policy gradient methods and is less complex than its predecessor the Trust Region Policy Optimization (TRPO). Unlike traditional gradient policy methods which perform a single update per sample, PPO uses multiple mini-batches to optimize the gradient on the same data.

The observation space provided by the CarRacing-v2 environment are 96x96 RGB images. To use them with the PPO, we pre-process them by normalizing them and converting them to grayscale. Since the bottom bar doesn't give us any important information about the car's behavior, I decided to crop the images to reduce the irrelevant data supplied to the model.

The PPO agent processes images with CNN and fully connected layers. I chose to add a Dropout layer to prevent possible overfitting.

The actions are modeled by a Gaussian distribution, which the agent use to choose the next move.

The Adam optimizer is used to reduce the loss function, which is defined as in the paper for an iteration t:

$$\boxed{L_t^{CLIP+VF+S}(\theta) = \widehat{\mathbb{E}_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]}$$

With:

– $c_1 = 0.5 \equiv$ The value loss coefficient
– $c_2 = 0.01 \equiv$ The entropy coefficient
– $S \equiv$ The entropy bonus
– $L_t^{VF} \equiv$ The squared error loss: $\left(V_\theta(s_t) - V_t^{targ}\right)^2$
– $L_t^{CLIP} \equiv$ The policy loss, defined as follows:

$$\boxed{L_t^{CLIP}(\theta) = \widehat{\mathbb{E}_t}\left[\min\left(r_t(\theta)\hat{A}_t, \operatorname{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]}$$

Here, epsilon $\epsilon = 0.2$ is a hyperparameter. The first term corresponds to the surrogate objective of the TRPO algorithm:

$$\boxed{L^{CPI} = \widehat{\mathbb{E}_t}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right]}$$

Where:

– $\widehat{A_t}$ is the advantage at time t
– $\pi_\theta(a_t|s_t)$ and $\pi_{\theta_{old}}(a_t|s_t)$ are respectively the new and old action policies. I calculated the ratio by using the log difference of policies.

The second term of the min function is a clipping term used to prevent large policy updates which could destabilize the process and make the new policy too different from the old one.

The rollout function was used to collect experiences, rewards states and losses to backpropagate the network.

Observation and results:

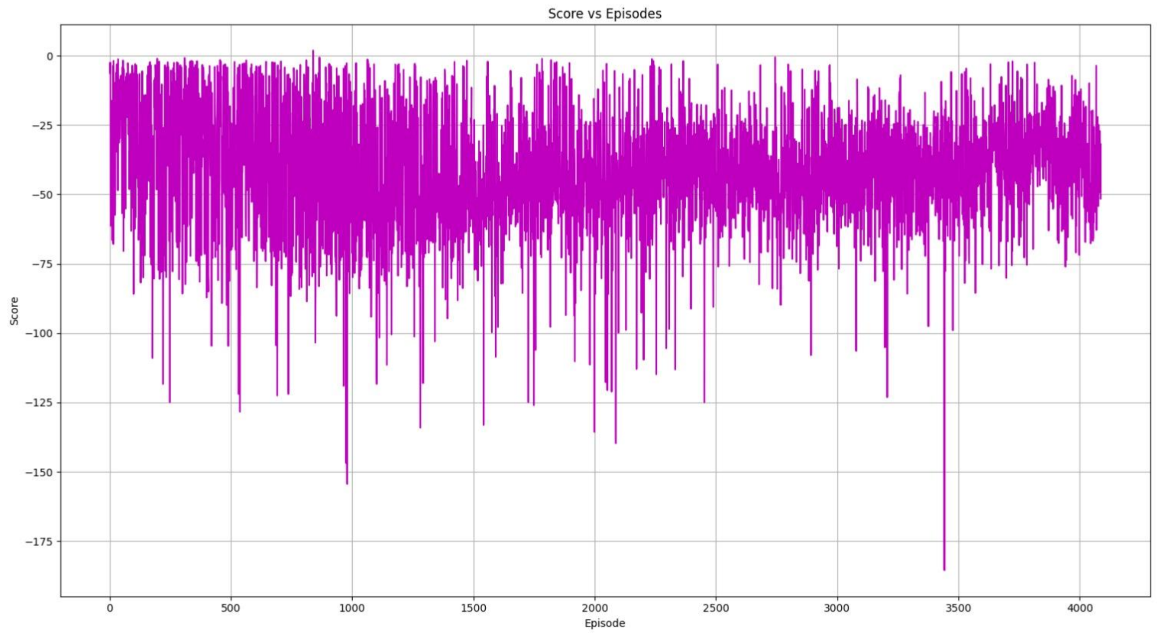

Figure 2 – Scores vs Episodes after 4080 epochs

The results of the training indicate that the agent performs poorly overall. The car managed to move forward a little, but slowly, so the average reward is negative (Between -30 and -50). This underperformance could be because of the complexity of the problem which would have required a longer training session than the one I did. I would also have liked to test other configurations for the hyperparameters (modify the size of the CNN network, slightly increase the learning rate or the entropy factor), but the computation time required to run many episodes only allowed me to make one attempt. In the future, exploring these adjustments and extending the training time would likely yield better results.

III – Resources

1. My notes and slides from the lecture "REINFORCE, Baselines and Actor-Critic".

2. Proximal Policy Optimization (PPO) Algorithms paper:
https://arxiv.org/pdf/1707.06347

3. CarRacing-v2 – Gymnasium environment:
https://gymnasium.farama.org/environments/box2d/car_racing/