Sapienza Università di Roma

Department of Computer, Control and Management Engineering Antonio Ruberti

Assignment 3

Reinforcement Learning

Master program in Artificial Intelligence and Robotics

Professor:

Roberto Capobianco

TAs:

Andrea Fanti

Michela Proietti

Student:

Lavalle Leonardo 1838492

Academic Year 2023/2024

Contents

1 Theory

Given the transition depicted in Figure 1, we need to compute one update step of the *1-step Actor-Critic Algorithm* in an environment with two possible actions and a 2-d state representation ($x(s) = (x_1(s) \quad x_2(s)) \in \mathbb{R}^2$).

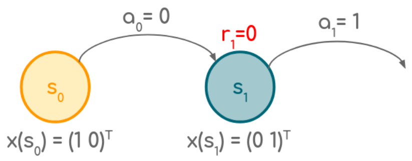


Figure 1

The policy approximator is in this form:

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

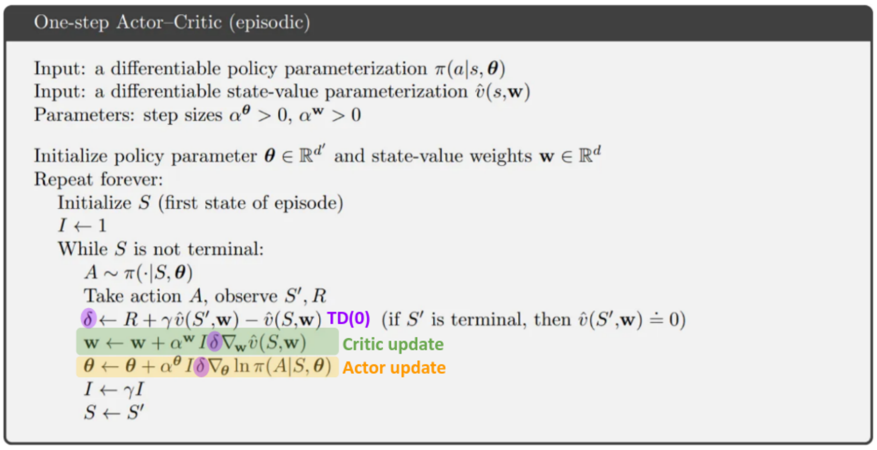
and indeed, being a *sigmoid* function:

$$\pi_\theta(a = 0|s) = 1 - \frac{1}{1 + e^{-(\theta^T \cdot x(s))}} \tag{2}$$

The action-state value function approximator is instead linear:

$$Q_w(s, a = 0) = \begin{pmatrix} w_{01} \\ w_{02} \end{pmatrix} \cdot (x_1(s) \quad x_2(s)) \tag{3}$$

$$Q_w(s, a = 1) = \begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix} \cdot (x_1(s) \quad x_2(s)) \tag{4}$$



```
One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$
Repeat forever:
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    While $S$ is not terminal:
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ TD(0) (if $S'$ is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$ Critic update
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S, \boldsymbol{\theta})$ Actor update
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

Figure 2

Following the pseudo-code of the algorithm in Figure 2, we know compute $(w_0 \quad w_1)$ and θ after the transition.

First of all, we compute δ that is used by both updates.

$$\begin{aligned}
\delta &= r_1 + \gamma Q_w(s_1, a_1) - Q_w(s_0, a_0) \\
&= r_1 + \gamma \cdot \begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix} \cdot (x_1(s_1) \quad x_2(s_1) - \begin{pmatrix} w_{01} \\ w_{0_2} \end{pmatrix} \cdot (x_1(s_0) \quad x_2(s_0)) \\
&= r_1 + \gamma \cdot \begin{pmatrix} 0.4 \\ 0 \end{pmatrix} \cdot (0 \quad 1) - \begin{pmatrix} 0.8 \\ 1 \end{pmatrix} \cdot (1 \quad 0) \\
&= 0 + 0.9 \cdot \begin{pmatrix} 0.4 \\ 0 \end{pmatrix} \cdot (0 \quad 1) - \begin{pmatrix} 0.8 \\ 1 \end{pmatrix} \cdot (1 \quad 0) \\
&= 0 + 0.9 \cdot 0 - 0.8 = -0.8
\end{aligned}$$

Let's start with the *critic* update:

$$\begin{aligned}
(w_0 \quad w_1) = \begin{pmatrix} w_{01} & w_{11} \\ w_{02} & w_{12} \end{pmatrix} &= \begin{pmatrix} w_{01} & w_{11} \\ w_{02} & w_{12} \end{pmatrix} + \alpha_w \cdot \delta \cdot \nabla_w Q_w(s_0, a_0) \\
&= \begin{pmatrix} w_{01} & w_{11} \\ w_{02} & w_{12} \end{pmatrix} + \alpha_w \cdot \delta \cdot \nabla_w \left(\begin{pmatrix} w_{01} \\ w_{0_2} \end{pmatrix} \cdot (x_1(s_0) \quad x_2(s_0)) \right) \\
&= \begin{pmatrix} w_{01} & w_{11} \\ w_{02} & w_{12} \end{pmatrix} + \alpha_w \cdot \delta \cdot \nabla_w \left(\begin{pmatrix} w_{01} \\ w_{0_2} \end{pmatrix} \cdot (1 \quad 0) \right) \\
&= \begin{pmatrix} w_{01} & w_{11} \\ w_{02} & w_{12} \end{pmatrix} + \alpha_w \cdot \delta \cdot \nabla_w \left(w_{01} \right) \\
&= \begin{pmatrix} 0.8 & 0.4 \\ 1 & 0 \end{pmatrix} + 0.1 \cdot (-0.8) \cdot \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0.8 & 0.4 \\ 1 & 0 \end{pmatrix} - \begin{pmatrix} 0.08 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0.72 & 0.4 \\ 1 & 0 \end{pmatrix}
\end{aligned}$$

Then continue with the *actor* update:

$$
\begin{aligned}
\theta = (\theta_1 \quad \theta_2) &= (\theta_1 \quad \theta_2) + \alpha_\theta \cdot \delta \cdot \nabla_\theta \ln \pi_\theta(a_0|s_0) \\
&= (\theta_1 \quad \theta_2) + \alpha_\theta \cdot \delta \cdot \nabla_\theta \ln \left(1 - \frac{1}{1+e^{-(\theta^T \cdot x(s_0))}}\right) \\
&= (\theta_1 \quad \theta_2) + \alpha_\theta \cdot \delta \cdot \nabla_\theta \ln \left(1 - \frac{1}{1+e^{-\begin{pmatrix}\theta_1 \\ \theta_2\end{pmatrix} \cdot (1 \quad 0)}}\right) \\
&= (\theta_1 \quad \theta_2) + \alpha_\theta \cdot \delta \cdot \underbrace{\nabla_\theta \ln \left(1 - \tfrac{1}{1+e^{-\theta_1}}\right)}_{\text{see Appendix 1.1 for details}} \\
&= (\theta_1 \quad \theta_2) + \alpha_\theta \cdot \delta \cdot \left(-\left(\frac{1}{e^{\theta_1}+1}\right) \cdot e^{\theta_1} \quad 0\right) \\
&= (1 \quad 0.5) + 0.1 \cdot (-0.8) \cdot \left(-\frac{e^1}{e^1+1} \quad 0\right) \\
&= (1 \quad 0.5) - (0.08) \cdot (-0.73 \quad 0) \\
&= (1 \quad 0.5) + (0.08 \cdot 0.73 \quad 0) \\
&= (1 \quad 0.5) + (0.0584 \quad 0) = (1.0584 \quad 0.5)
\end{aligned}
$$

1.1 Appendix

We separately compute $\nabla_\theta \ln \left(1 - \frac{1}{1+e^{-\theta_1}}\right)$. We initially manipulate the function:

$$
\begin{aligned}
\ln\left(1 - \frac{1}{1+e^{-\theta_1}}\right) &= \ln\left(1 - \frac{1}{1+\frac{1}{e^{\theta_1}}}\right) = \ln\left(1 - \frac{e^{\theta_1}}{e^{\theta_1}+1}\right) \\
&= \ln\left(\frac{e^{\theta_1}+1-e^{\theta_1}}{e^{\theta_1}+1}\right) = \ln\left(e^{\theta_1}+1\right)^{-1} = -\ln\left(e^{\theta_1}+1\right)
\end{aligned}
$$

Finally:

$$
\begin{aligned}
\nabla_\theta\left(-\ln\left(e^{\theta_1}+1\right)\right) &= \left(\frac{\partial\left(-\ln\left(e^{\theta_1}+1\right)\right)}{\partial\theta_1} \quad \frac{\partial\left(-\ln\left(e^{\theta_1}+1\right)\right)}{\partial\theta_2}\right) \\
&= \left(\frac{\partial\left(-\ln\left(e^{\theta_1}+1\right)\right)}{\partial\theta_1} \quad 0\right) \\
&= \left(-\left(\frac{1}{e^{\theta_1}+1}\right) \cdot e^{\theta_1} \quad 0\right)
\end{aligned}
$$

2 Code

2.1 Car Racing

I choose the Proximal Policy Optimization (PPO) algorithm for solving the *Car Racing* environment. It's a policy gradient method which have some of the benefits of trust region policy optimiza- tion (TRPO), but it's much simpler to implement and more general to be applied on.
The algorithm was conceived with the idea of being trained in parallel, meaning that more *actor* networks are involved in the training at the same time. My implementation only exploits one agent! It must be said that the original paper doesn't give much implementation details and it doesn't exist an original code repository. For this reason, my sources from which I took inspiration were multiple. Starting from the "REINFORCE.ipynb" notebook from the practicals, arriving at these two articles: The 37 Implementation Details of Proximal Policy Optimization and A Graphic Guide to Implementing PPO for Atari Games.
PPO algorithm is conceptually simple. Basically it can be divided in two phases: (i) the *rollout* phase where the agent gather new data by acting in the environment; (ii) the *learning* phase where the agent, through the rollout data, learns the weights both of the *actor* and of the *critic* models (convolutional neural networks). In practice, the implementation is not so trivial and I encountered not few problems along the way. As already mentioned, since there is no proper official and well explained code, one is quite free to implement its own PPO version. This is what I did.

2.2 Implementation

Beginning with the architecture of the two networks, I decided to employ a shared weights architecture to be more computationally efficient (I also initialized them via the so called *orthogonal initialization*). As we already saw in the practicals, the way to go in these kind of tasks is to stack subsequent frames of the game, manipulating them in a certain way and fed them into the convolutional networks. The *pre-processing* steps followed exactly the implementation of the REINFORCE.ipynb notebook. The optimization process involved Adam optimizer with a learning rate of 0.0001 and a batch size of 256. The learning rate is decayed over the training episodes thanks to a learning rate linear annealing approach. Gradient clipping was tried but with poor results.
An hyperparameter that was pretty hard to set was the number of *rollout steps* per episode, i.e. the maximum number of steps needed by the agent to store training data (set to 5000 for not being stingy). Immediately after this step and immediately before the actual learning process, the *advantage* values are estimated. Also in this case,

the paper wasn't clear on how to compute them. Anyway, I computed the *advantage* through the Generalized Advantage Estimation (GAE) form. Before using them to compute the loss, is essential to normalize them.
How the *actor-critic* networks learn? Through the minimization of a loss. The final loss in this algorithm is composed by three component: the "*clipped surrogate objective*" of the policy network ($\mathcal{L}^{CLIP}$), the mean squared error of the value function ($\mathcal{L}^{VF}$) and the *entropy* bonus ($\mathcal{S}$) of actions distribution.

2.3 Results

The algorithm behaved decently in the environment from the outset, but to achieve better results I had to tweak a lot all the hyperparameters involved. After reaching satisfiable *total rewards*, I was interested in observing how the number of frames to stack would have influenced the behaviour of the agent. Indeed, I tested it with 4, 8 and 16 number of stacked frames for 10 and 50 evaluation episodes.

	Mean Total Rewards	
	10 episodes	50 episodes
4 frames	577.817	556.429
8 frames	**712.219**	**650.755**
16 frames	638.395	469.29

As we can see in the above table, the best configuration is in the middle with 8 stacked frames (is the one I delivered for the assignment). The superiority of the 8 frames setting emerged also from a *qualitatively* point of view: when the environment is rendered, it can be seen how the car trajectories are smoother and also more correct (from a human point of view) than the ones performed by the other two cases. In

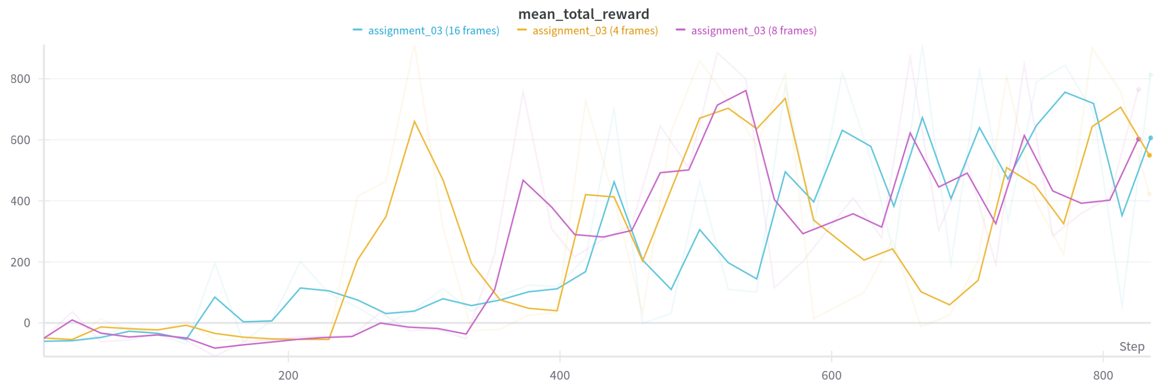


Figure 3

Figure 3 the plot illustrates the mean (computed over 10 evaluation episodes) of the total rewards throughout the training phase. The measure is taken each 5 training

episodes. There are not significant differences between 4 and 8 number of frames, while is quite evident how inferior the "16 version" is in terms of performance.
For the sake of curiosity, I also want to show the plots (Figure 4) of the three components of the PPO loss (from top to bottom): clipped policy gradient loss, value function loss and entropy measure.

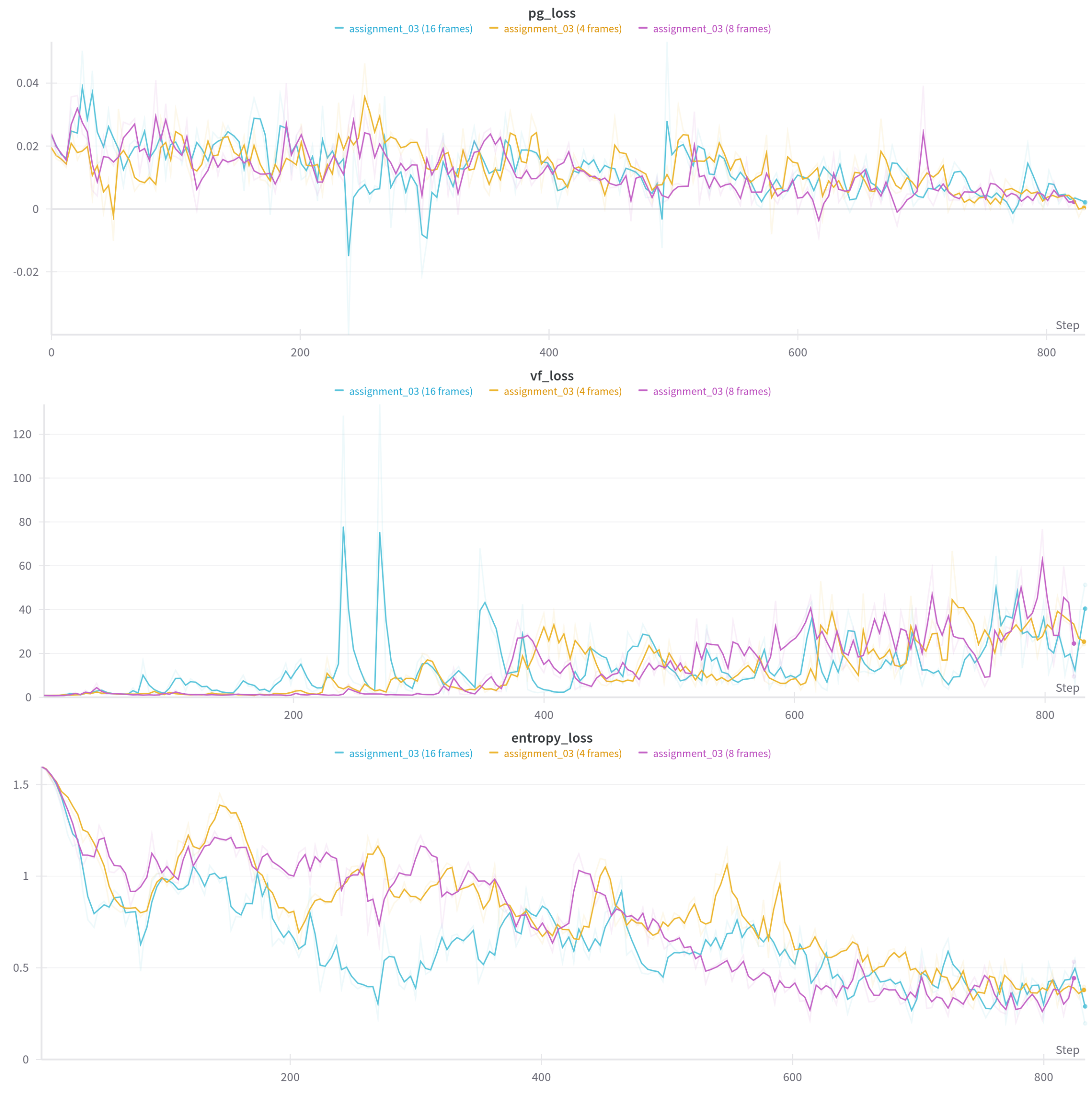


Figure 4