BEN-GURION UNIVERSITY OF THE NEGEV

FACULTY OF ENGINEERING SCIENCES

Deep Reinforcement Learning

Assignment 2: Policy Gradient Methods

By: Or Simhon, Ofir Ben Moshe

ID: 315600486, 315923151

Mail: orsi@post.bgu.ac.il, ofirbenm@post.bgu.ac.il

Lecturer: Gilad Katz, Ph.D.

Dec 2021

Contents

1. Section 1 – Monte Carlo Policy Gradient (REIFORCE)

1.1 What does the value of the advantage estimate reflect? Why is it better to follow the gradient computed with the advantage estimate instead of just the return itself?

One of the main problems with the *REINFORCE* algorithm stems from the high variance received because of the use of sampling, which might cause the learning process to fluctuate and slow convergence. To decrease the variance, we can use baseline that will reduce the size of the update. To do so, the baseline method propose to use the advantage function $A_t = R_t - b(s_t)$.

In *REINFORCE* algorithm, the update step uses the discounted return value G_t to inform the agent about good actions and bad actions, given some state. Using the advantage estimate rather than just the discounted returns, the agent can be informed also about how much better is it to take some action than expected. Therefore, using the advantage estimate instead of just the discounted return can lead to a faster convergence.

With taking the baseline as a value function $V(s_t)$, the advantage function describe the increase or decrease of the expected return when taking action a in state s with respect to the mean expected return at state s.

1.2 The reduction of the baseline does not introduce bias to the expectation of the gradient since:

$$\boldsymbol{E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a_t|s_t)b(s_t)\,] = 0}$$

What is the prerequisite condition for that equation to be true? Try to prove that equation

First, to better understand, we will expand the expression above all actions.

$$E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a_t|s_t)b(s_t)\,] = \sum_{s\in S}\sum_{a\in A}\mu(s)\cdot\pi_\theta(a|s)\cdot\nabla_\theta log\pi_\theta(a|s)b(s) \qquad (1)$$

From derivative rules, we can say that:

$$\nabla\log(x) = \frac{\nabla x}{x} \quad\rightarrow\quad \nabla x = x\cdot\nabla\log(x) \qquad (2)$$

By implementing this rule to the eq. (1), we will get the following expression.

$$\sum_{s\in S}\sum_{a\in A}\mu(s)\cdot\pi_\theta(a|s)\cdot\nabla_\theta log\pi_\theta(a|s)b(s) = \sum_{s\in S}\sum_{a\in A}\mu(s)\cdot\nabla_\theta\pi_\theta(a|s)b(s) \qquad (3)$$

Using the linearity of gradients $\nabla(f+g) = \nabla f + \nabla g$ and under the assumption that $b(s)$ is independent with any of the actions:

$$\sum_{s\in S}\sum_{a\in A}\mu(s)\cdot\nabla_\theta\pi_\theta(a|s)b(s) = \sum_{s\in S}\mu(s)\cdot b(s)\cdot\nabla_\theta\sum_{a\in A}\pi_\theta(a|s) \qquad (4)$$

Because π_θ is defined to be a probability function, so $\sum_{a\in A}\pi_\theta(a|s) = 1$. Therefore we get:

$$\sum_{s\in S}\mu(s)\cdot b(s)\cdot\nabla_\theta\sum_{a\in A}\pi_\theta(a|s) = \sum_{s\in S}\mu(s)\cdot b(s)\cdot\nabla_\theta(1) = 0 \qquad (5)$$

So finally, as we try to proof, we got that:

$$E_{\pi_\theta}[\nabla log\pi_\theta(a_t|s_t)b(s_t)\,] = 0$$

Which means that the subtraction of a baseline function can reduce variance without changing expectation. To get this result, two prerequisites were assumed:

- The term π_θ is defined to be a probability function, so $\sum_{a\in A}\pi_\theta(a|s) = 1$.
- The term $b(s)$ is independent with any of the possible actions $a \in A$.

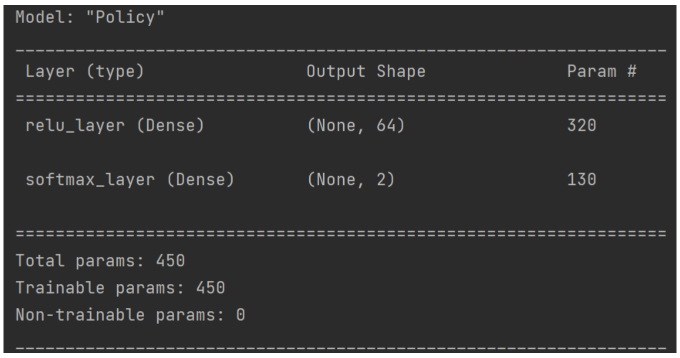
1.3 Python Implementation:

To implement the algorithms ***REINFORCE*** and ***REINFORCE with Baseline*** in python we first implement it with the package of Tensorflow.2 as we preferred, we rewrote the algorithm ***REINFORCE*** from the beginning and then add the baseline improvement. After we got a reply from Gilad that we need to work with the given base code of ***REINFORCE,*** we wrote the scripts again in Tensorflow.1. Therefore, we submit 5 scripts. (3 of the work with TF version 2 and 2 more with TF version 1).

For the ***REINFORCE*** algorithm, we used the following hyper-parameters:

Hyper-Parameter	Final
$lr: \alpha$	0.002
$discount\ factor: \gamma$	0.99

With the following network architecture for the policy.

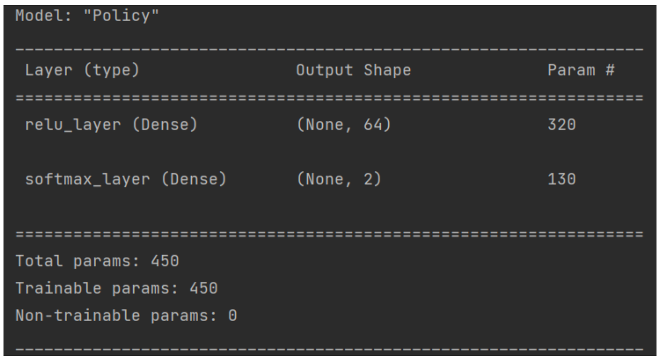


```
Model: "Policy"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 relu_layer (Dense)          (None, 64)                320

 softmax_layer (Dense)       (None, 2)                 130

=================================================================
Total params: 450
Trainable params: 450
Non-trainable params: 0
_________________________________________________________________
```

For the ***REINFORCE with Baseline*** algorithm, we used the following hyper-parameters:

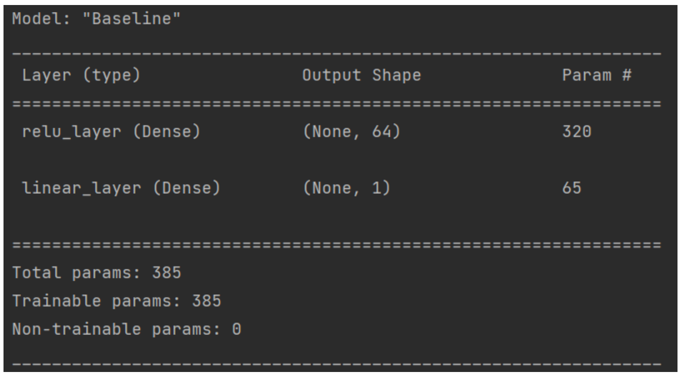
Hyper-Parameter	Final
$policy\ lr: \alpha^{\pi}$	0.002
$baseline\ lr: \alpha^{b}$	0.002
$discount\ factor: \gamma$	0.99

With the following network architecture for the policy.



```
Model: "Policy"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 relu_layer (Dense)          (None, 64)                320

 softmax_layer (Dense)       (None, 2)                 130

=================================================================
Total params: 450
Trainable params: 450
Non-trainable params: 0
_________________________________________________________________
```

And the following network architecture for the baseline.



```
Model: "Baseline"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 relu_layer (Dense)          (None, 64)                320

 linear_layer (Dense)        (None, 1)                 65

=================================================================
Total params: 385
Trainable params: 385
Non-trainable params: 0
_________________________________________________________________
```

1.4 Comparison:

In this section, the convergence results for these 2 algorithms are presented together. The ***REINFORCE*** model is in orange color and the ***REINFORCE with Baseline*** model in blue.

Figure of the Reward per episode

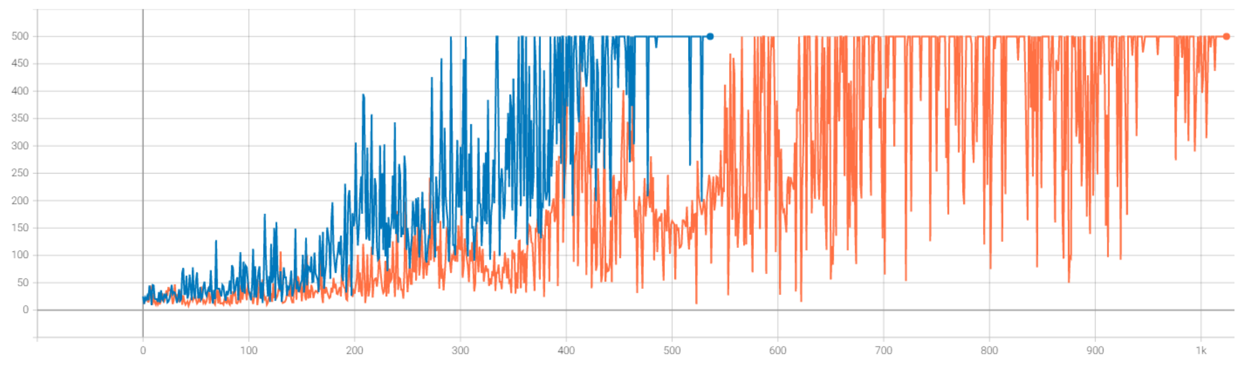


Figure of the average Reward in the last 100 episodes

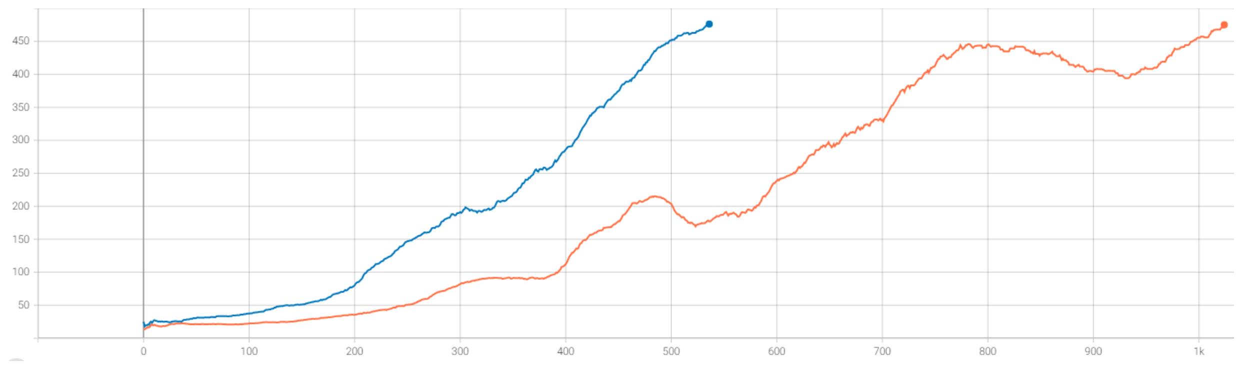
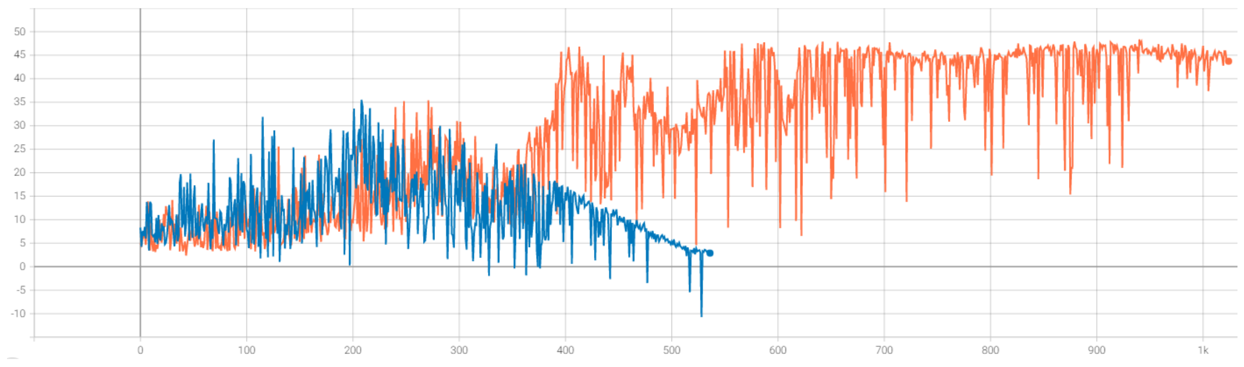


Figure of the Loss per episode



The ***REINFORCE*** model execution (orange) converged after 1024 episodes.

The ***REINFORCE with Baseline*** model (blue) converged after 536 episodes.

2. Section 2 – Advantage Actor-Critic

2.1 Why is using the TD-error of the value function for the update of the policy network parameters is practically the same as using the advantage estimate? Prove.

To be able to estimate the return R_t for each step, in order to compute the advantage function A_t, we need to estimate the action value function $Q_w(s_t, a_t) = E[R_t|s_t, a_t]$, so we have the advantage function as:

$$\hat{A}_t = \hat{Q}_w(s_t, a_t) - \hat{V}_v(s_t) \tag{6}$$

But, because of the connection between $Q(s_t, a_t)$ and $V(s_t)$, we can use one less NN. From the Bellman expectation equations, we know that $R_{t+1} + \gamma \cdot V(s_{t+1})$ is an unbiased estimate of the actual return R_t. Based on this fact we can understand the connection between these expressions. For the true value function, the TD error is δ_{π_θ}:

$$\delta_{\pi_\theta} = r + \gamma \cdot V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t) \tag{7}$$

From this point, we can show that the TD-error is an unbiased estimate of the advantage to show that the update rule is practically the same in both cases. By calculating the expected value of the TD-error and using the linearity of the expectation operator, we will get:

$$E_{\pi_\theta}[\delta_t|s_t, a_t] = E_{\pi_\theta}[R_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t)|s_t, a_t] \tag{8}$$

$$E_{\pi_\theta}[\delta_t|s_t, a_t] = \underbrace{E_{\pi_\theta}[R_{t+1} + \gamma \cdot V(s_{t+1})|s_t, a_t]}_{part\ 1} - \underbrace{E_{\pi_\theta}[V(s_t)|s_t, a_t]}_{part\ 2} \tag{9}$$

The expression in eq. 8 divided into two parts. Using the Bellman equations in reverse direction, we know that for part 1:

$$E_{\pi_\theta}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\, |s_t, a_t\right] = Q_{\pi_\theta}(s_t, a_t) \tag{10}$$

And for part 2, with taking in consider that the value is independent with the action.

$$E_{\pi_\theta}[V(s_t)|s_t] = E_{\pi_\theta}\left[E_{\pi_\theta}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\, |s_t\right] |s_t\right] = E_{\pi_\theta}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\, |s_t\right] = V_{\pi_\theta}(s_t) \tag{11}$$

Finally by inserting equations 10 and 11 to eq. 9, we get:

$$E_{\pi_\theta}[\delta_t|s_t, a_t] = Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t) = A_{\pi_\theta}(s_t, a_t) \tag{12}$$

Therefore, we got that the advantage function is equal to the expectation value of the TF-Error, which means that they are practically the same.

2.2 Explain which function is the actor and which is the critic of the model and what is the role of each one?

Both Value-based methods and Policy-based methods have their drawbacks, either large variance together with no accumulation of older information or lacks guaranties of the solution being near optimal. Combining both together we can create a hybrid method that can handle this issues.

The actor is the one that represents the policy. A policy function $\pi_\theta(A_t|s_t)$ that controls how the agent will behave. The actor update the policy distribution in the direction suggested by the critic.

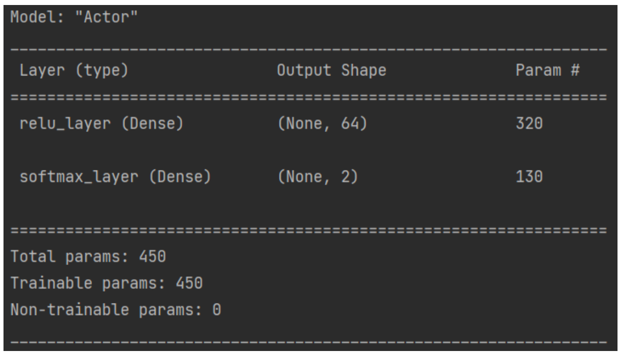
The critic however, is the one that represents the value function. A value function $Q_w(\boldsymbol{s_t}, \boldsymbol{a_t})$ that measures how good was the action to help the actor to improve the policy.

2.3 Python Implementation

For the ***One-step Actor-Critic*** algorithm, we used the following hyper-parameters:

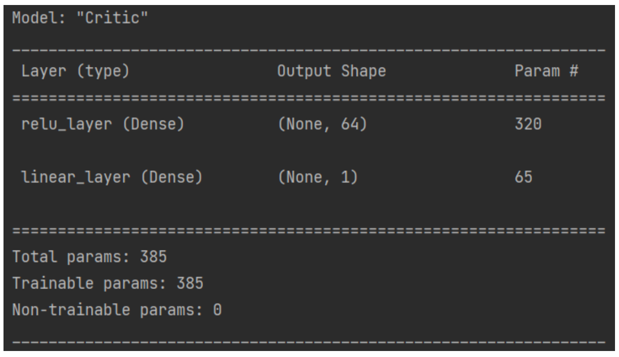
Hyper-Parameter	Final
$Actor\ lr: \alpha^{Actor}$	0.002
$Critic\ lr: \alpha^{Critic}$	0.001
$LR\ decay\ factor$	0.7
$LR\ decay\ rate$	40 episodes
$discount\ factor: \gamma$	0.99

With the following network architecture for the Actor.



```
Model: "Actor"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 relu_layer (Dense)          (None, 64)                320

 softmax_layer (Dense)       (None, 2)                 130

=================================================================
Total params: 450
Trainable params: 450
Non-trainable params: 0
_________________________________________________________________
```

And the following network architecture for the Critic.



```
Model: "Critic"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 relu_layer (Dense)          (None, 64)                320

 linear_layer (Dense)        (None, 1)                 65

=================================================================
Total params: 385
Trainable params: 385
Non-trainable params: 0
_________________________________________________________________
```

2.4 Comparison

In this section, the convergence results for all the 3 implemented algorithms are presented together. The ***REINFORCE*** model is in orange color, the ***REINFORCE with Baseline*** model in blue and the ***One-step Actor-Critic*** in red.

Figure of the Reward per episode

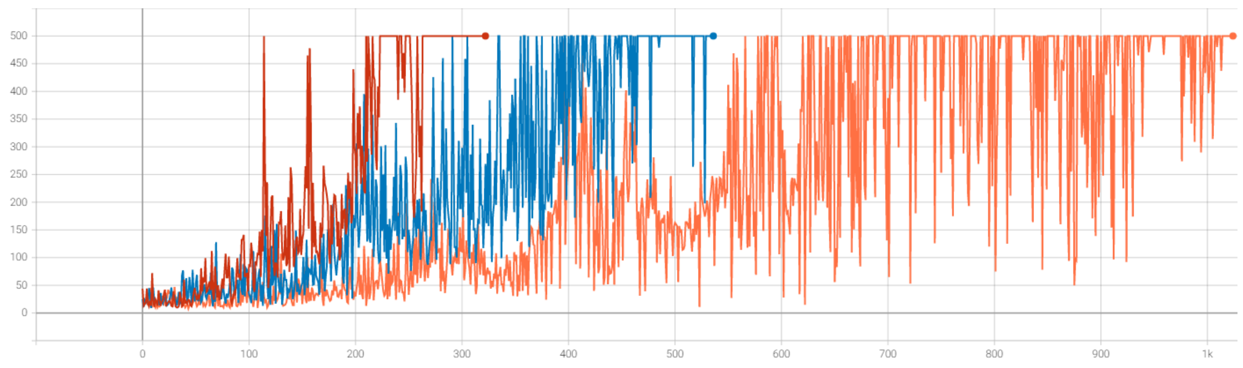


Figure of the average Reward in the last 100 episodes

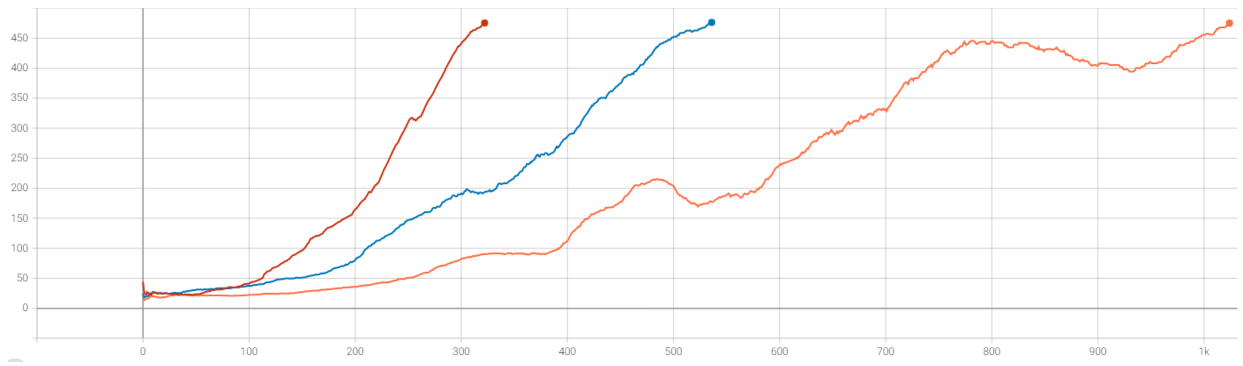
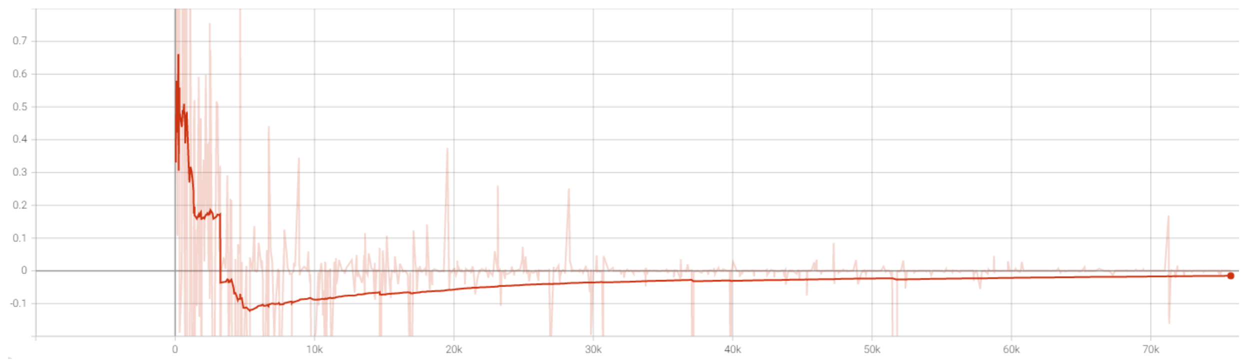


Figure of the Loss per step on the ***One-step Actor-Critic*** algorithm



The ***REINFORCE*** model execution (orange) converged after 1024 episodes.

The ***REINFORCE with Baseline*** model (blue) converged after 536 episodes.

The ***One-step Actor-Critic*** model (red) converged after 322 episodes.

3. Code Execution Explanation

As we said in section 1, we first wrote the scripts for this work in Tensorflow version 2, but afterward, we got a reply from Gilad that we need to work with the given base code of REINFORCE, so we wrote the scripts again in Tensorflow.1. Therefore, we submit 5 scripts. (3 of the work with TF version 2 and 2 more with TF version 1).

In order to run the .py files, there is a need to install all the relevant packages using the requirements.txt file. When located in the directory of this file in the Anaconda Prompt, you can use the following commands in your virtual environment (or to create new one) to do it. This project created in python 3.8.

If you want to create new virtual environment (optional):

- **conda create --name TensorFlowEnv python=3.8**

Activate the virtual environment (optional):

- **conda activate TensorFlowEnv**

Install all the required packages to the new virtual environment:

- **pip install -r requirements.txt**

Now after the environment is set up you can run the assignment files with the following commands from the anaconda prompt.

Scripts wrote with Tensorflow version 2:

For section 1 you should run: python Section1_Or_Ofir_REINFORCE_TF2.py

For section 1 you should run: python Section2_Or_Ofir_REINFORCE_WithBaseline_TF2.py

For section 3 you should run: python Section3_Or_Ofir_ActorCritic_TF2.py

Scripts wrote with Tensorflow version 1:

For section 1 you should run: python Section2_Or_Ofir_REINFORCE_WithBaseline_TF1.py

For section 2 you should run: python Section3_Or_Ofir_ActorCritic_TF1.py