BEN-GURION UNIVERSITY OF THE NEGEV

FACULTY OF ENGINEERING SCIENCES

Deep Reinforcement Learning

Assignment 3: Meta and Transfer Learning

Or Simhon - 315600486

orsi@post.bgu.ac.il

Ofir Ben Moshe - 315923151

ofirbenm@post.bgu.ac.il

Lecturer: Gilad Katz, Ph.D.

January 2022

Contents

1 Training individual networks

In this section, we implemented the one-step actor critic algorithm to 3 different control problems from the openAI gym environments:

- **CartPole**
- **Acrobot**
- **MountainCar-Continues**

To allow the transfer learning we wish to implement in the next sections, we needed to set the same input and output sizes for all 3 actor networks. Therefore, we took the largest sizes from the different problems, i.e. 6 as the input size (state) and 3 as the output size (actions) which is compatible with the Acrobot environment.

1.1 CartPole

In this environment, we defined the learning target to reach an average reward of at least 475 in 100 consecutive episodes.

1.1.1 Networks architectures and hyper parameters

Since this environment is a relatively simple problem to solve, we tried to build a network with only 1 hidden layer, which after trying appears to lead to a satisfying convergence rate and performance. The networks architectures are presented below.

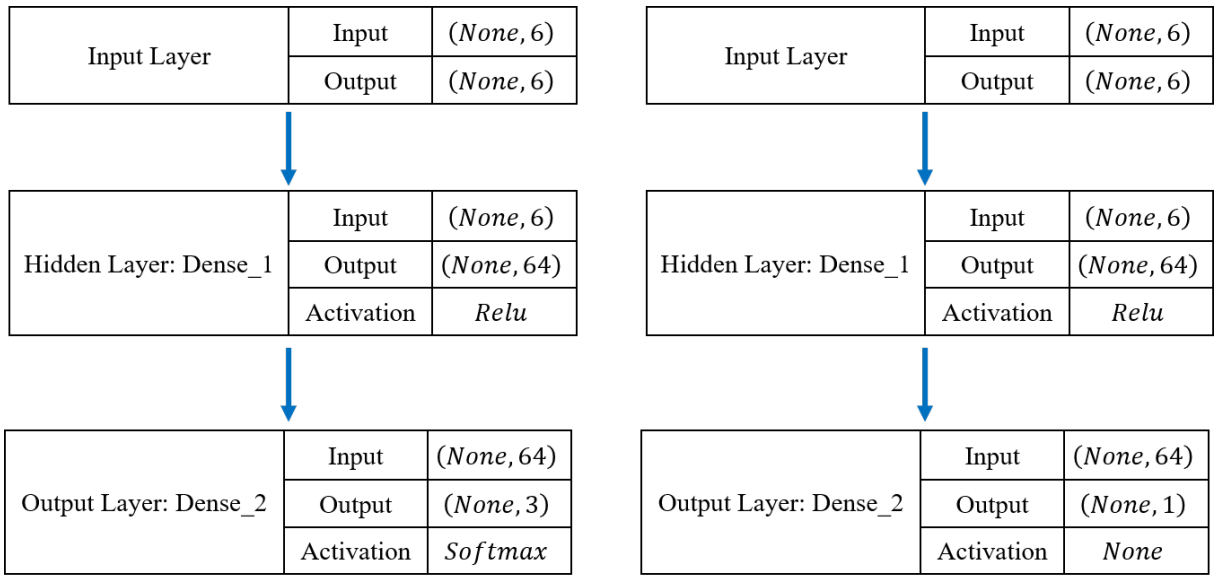


Figure 1: CartPole - Actor network architecture in the left side and Critic network architecture in the right side

The hyper parameters used in this environment presented in the next table.

Table 1: CartPole environment hyper parameters

Hyperparameter	Value
Actor lr	0.001
Critic lr	0.01
lr decay factor	0.7
lr decay rate	60 episodes
discount factor	0.99

1.1.2 Training Results

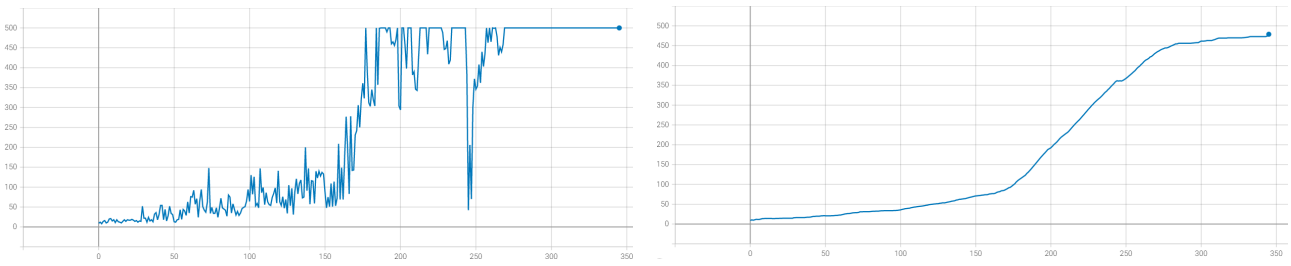
As can be seen in the next figure, it took 346 episodes for the agent to reach an average reward above 475 in consecutive 100 episodes. The time until convergence in this case was 2.6 minutes, and the cumulative reward in the test episode was 500.

```
Episode 344 | Reward: 500.00 | Average over 100 episodes: 473.08
Episode 345 | Reward: 500.00 | Average over 100 episodes: 474.25
Episode 346 | Reward: 500.00 | Average over 100 episodes: 478.83

Great!! You win after: 346 Episodes
Average reward in the last 100 episodes: 478.83
Training complete after 2.6 minutes
Total reward in the test: 500.00
```

Figure 2: CartPole environment summary of training

The next figure shows the resulted reward per episode as well as the average Reward in the consecutive 100 episodes



(a) **(b)**

Figure 3: CartPole - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

The next figure shows the Actor network loss per step.

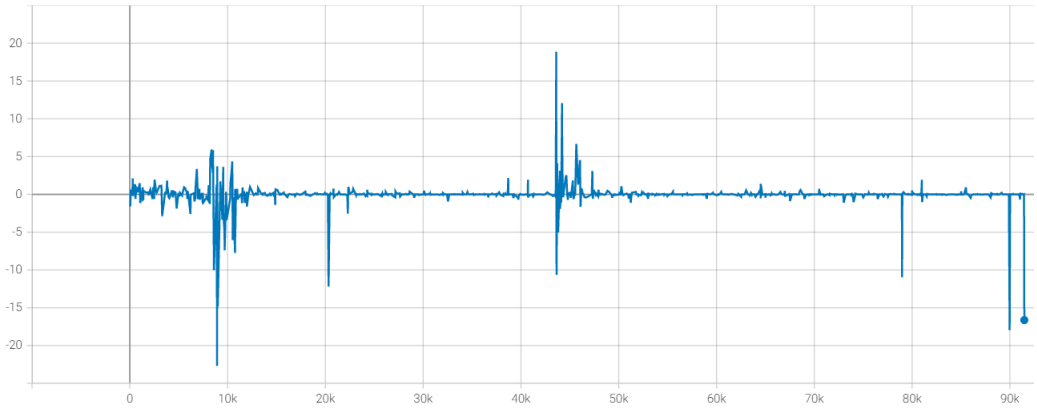


Figure 4: CartPole – Actor network loss per step

From figures 3a, 4 above, we can link between two phenomena. It can be seen that when there is a significant peak in the loss value (among steps 40k to 50k), the policy changed a lot (among episodes 150 to 200) what stems from large updates in the actor network. Also, we can notice from figure 3a that when the agent reached the best score (500 for episode), he apparently tried to find a better solution by exploring the environment, which led him to smaller scores. After some trial to do that, the agent decided to stick to the 500 points per episode solution.

1.2 Acrobot

In this environment we defined as learning target to reach an average reward of at least -100 in 100 consecutive episodes.

1.2.1 Networks architectures and hyper parameters

Since in this environment the learned policy needs to map vector with size 6 (observation dimension) to vector with size 3 (actions dimension), we first thought that it will be preferable to build larger networks relative to those used in the CartPole environment. Although, we first tried the same networks architectures we built in the CartPole environment and were glad to see that the training results were sufficiently good. The networks architectures are presented below.

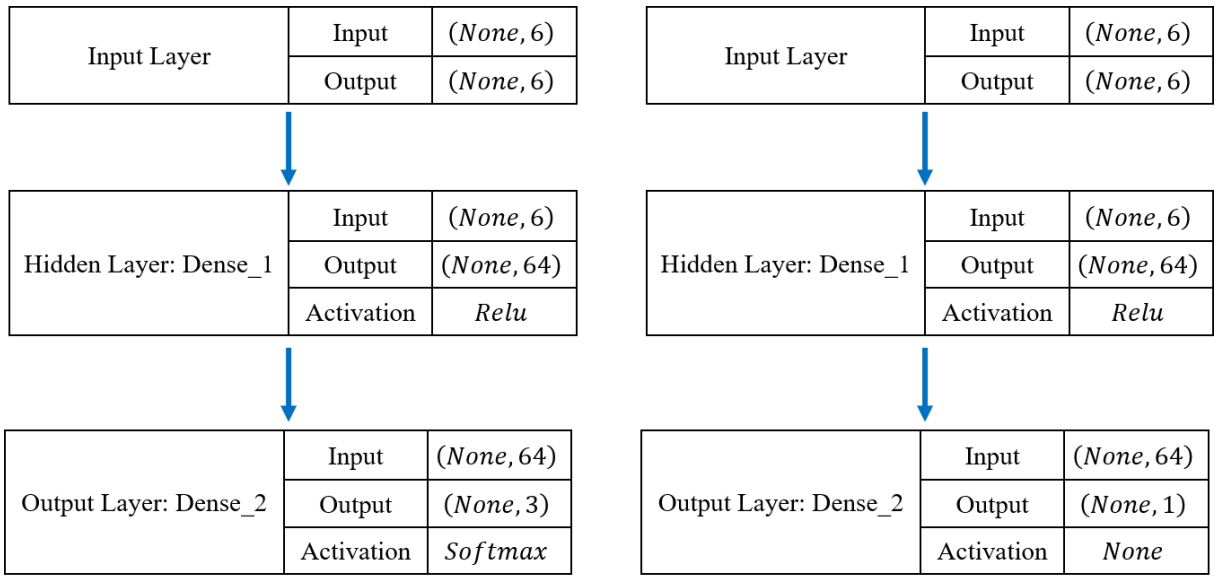


Figure 5: Acrobot - Actor network architecture in the left side and Critic network architecture in the right side

The hyper parameters used in this environment presented in the next table.

Table 2: Acrobot environment hyper parameters

Hyperparameter	Value
Actor lr	0.001
Critic lr	0.01
lr decay factor	0.7
lr decay rate	60 episodes
discount factor	0.99

1.2.2 Training Results

As can be seen in the next figure, it took 111 episodes for the agent to reach an average reward above -100 in consecutive 100 episodes. The time until convergence in this case was 0.38 minutes, and the cumulative reward in the test episode was -107.

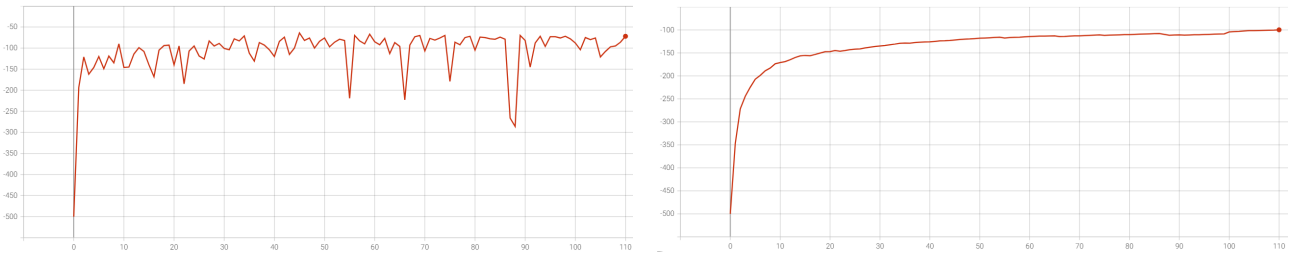
```
Episode 107 | Reward: -108.00 | Average over 100 episodes: -101.19
Episode 108 | Reward: -97.00 | Average over 100 episodes: -100.97
Episode 109 | Reward: -95.00 | Average over 100 episodes: -100.57
Episode 110 | Reward: -86.00 | Average over 100 episodes: -100.53
Episode 111 | Reward: -72.00 | Average over 100 episodes: -99.79

Great!! You win after: 111 Episodes
Average reward in the last 100 episodes: -99.79
Training complete after 0.38 minutes
Total reward in the test: -107.00
```

Figure 6: Acrobot environment summary of training

The next figure shows the resulted reward per episode as well as the average

Reward in the consecutive 100 episodes



(a) **(b)**

Figure 7: Acrobot - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

The next figure shows the Actor network loss per step.

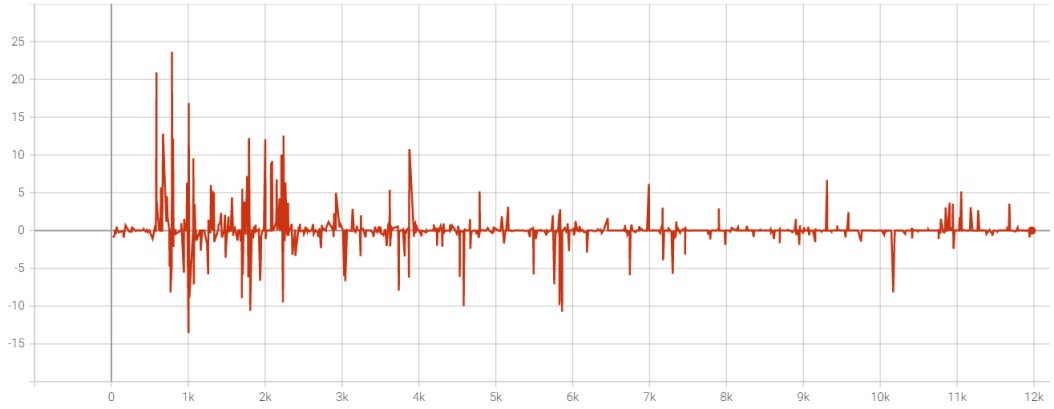


Figure 8: Acrobot – Actor network loss per step

In the same manner as the CartPole environment, we can observe from figures 9, 10 that when there was a significant peak in the loss value (among steps 1k to 2k), the policy changed a lot (among episodes 0 to 5). In this environment, it seems like the networks architectures and the hyperparameters that were chosen fit very well to this specific problem, due to the stability and the fast convergence.

1.3 MountainCar-Continuous

In this environment we define as learning target to reach an average reward of at least 90 in 100 consecutive episodes.
Unlike the previous two environments (CartPole and Acrobot), this environment has a continuous space for the action (state space is ~~discrete),~~ which means that the action to be received from the actor network given some state, is a continuous value that ranges between -1 to 1. Therefore we built our actor network such that given some state, it will output two values that define a normal distribution, mean and standard deviation. From the normal distribution that we get (and update every step) from the actor network, we sampled a continuous action, while clipping the actions values in the continuous range between [-1,1].
The main challenge in this environment stems from the fact that if the game

goes on for too long without reaching the target, the exploration rate declines with time, and therefore the agent is prone to converge to a sub-optimal local minimum that leads to accumulated reward of 0. To deal with it, we added a condition in the main train loop that if the accumulated reward of 5 consecutive episodes is less than 0 and the training runs already more than 15 episodes, so the networks variables are reinitialized (like at the beginning of the train run). This additional condition was implemented in code inside the training loop the following way.

```
if max(average_score[-5:]) < 0 and episode > 15:
    I = 1
    self.actor_lr = initiail_actor_lr
    self.critic_lr = initiail_critic_lr
    self.sess.run(tf.global_variables_initializer())
```

Figure 9: MountainCar - Condition to avoid from sub-optimal solution

1.3.1 Networks architectures and hyper parameters

Regarding the actor and critic networks architecture, we first tried the architectures that worked well for the previous problems, which can be seen in figures 1 and 6, but those resulted in an unstable training process and no convergence. Therefore we thoughts this problem is more complicated relative to the previous (probably because of the continuous action space) and decided to enlarge the actor network while decreasing the learning rates, which together with the condition in the training loop, led to good results. The networks architectures are presented below.

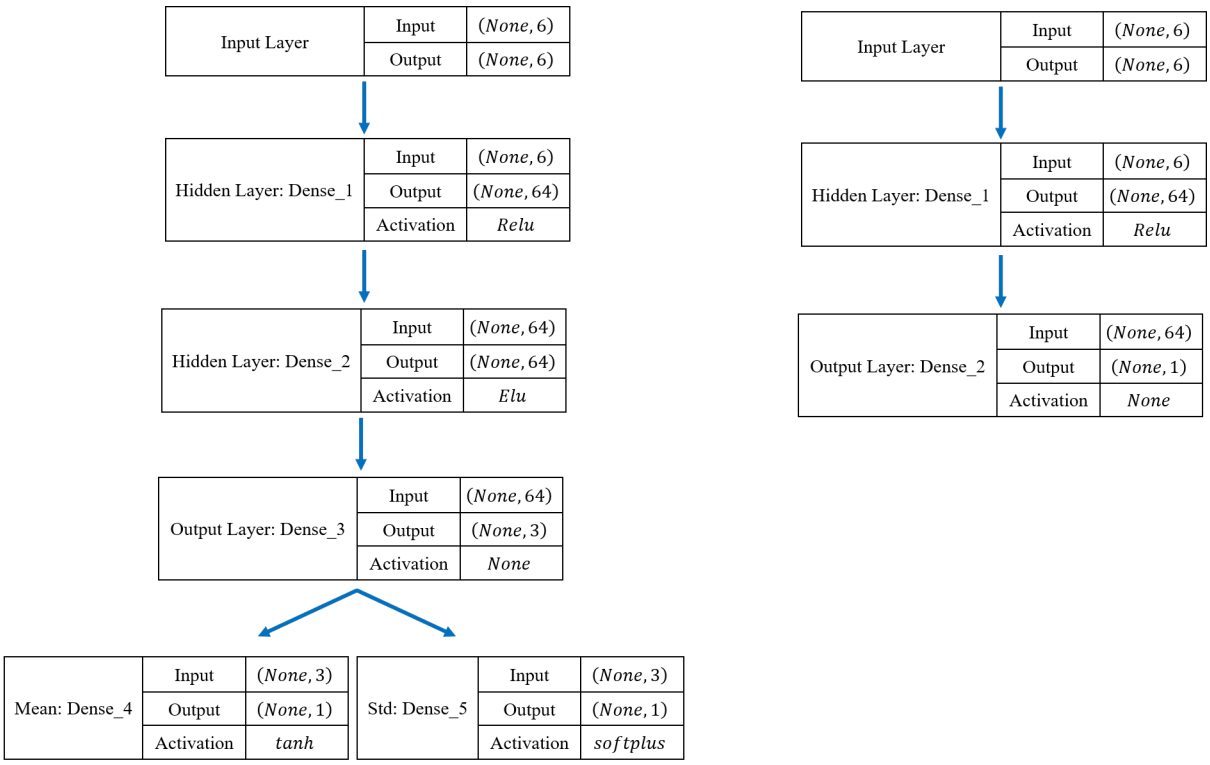


Figure 10: MountainCar - Actor network architecture in the left side and Critic network architecture in the right side

The hyper parameters used in this environment presented in the next table.

Table 3: MountainCar environment hyper parameters

Hyperparameter	Value
Actor lr	0.00002
Critic lr	0.001
lr decay factor	0.7
lr decay rate	60 episodes
discount factor	0.99

1.3.2 Training Results

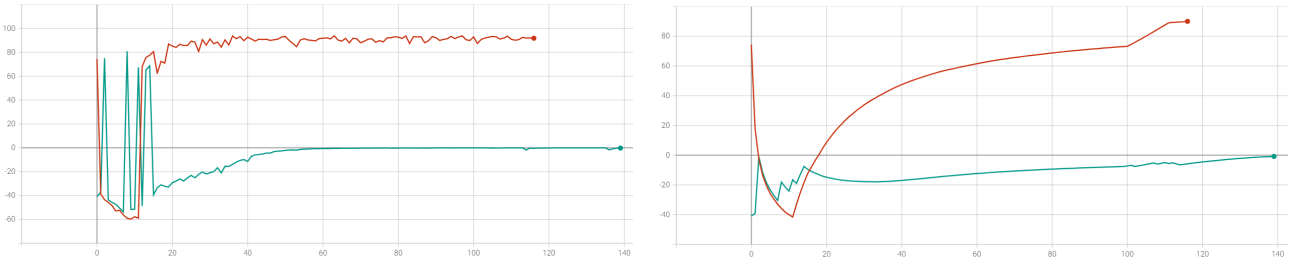
As can be seen in the next figure, it took 117 episodes for the agent to reach an average reward above -100 in consecutive 100 episodes. The time until convergence in this case was 1.2 minutes, and the cumulative reward in the test episode was 93.32.

```
Episode 113 | Reward: 90.61 | Average over 100 episodes: 89.30
Episode 114 | Reward: 92.50 | Average over 100 episodes: 89.47
Episode 115 | Reward: 91.96 | Average over 100 episodes: 89.61
Episode 116 | Reward: 92.01 | Average over 100 episodes: 89.72
Episode 117 | Reward: 91.79 | Average over 100 episodes: 90.02

Great!! You win after: 117 Episodes
Average reward in the last 100 episodes: 90.02
Training complete after 1.2 minutes
Total reward in the test: 93.32
```

Figure 11: MountainCar environment summary of training

In the following figures, we will introduce two runs of the MountainCar environment training, one for an optimal solution (in red color) that uses the additional condition for reinitializing the networks variables and one for a sub-optimal solution (in green color) which not used this method.



(a) **(b)**

Figure 12: MountainCar - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

The next figure shows the Actor network loss per step.

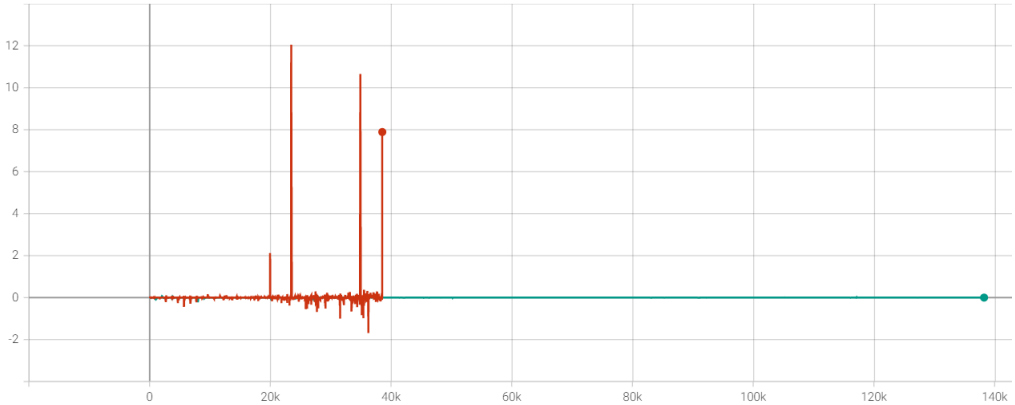


Figure 13: MountainCar – Actor network loss per step

From the figures above, it seems that the green line run (without the reinitialization condition) began better (until episode 15) than the red one. Which is probably related to the weight's initial values, but finally the red run was converged to the optimal solution for this environment, and the green one doesn't. This is because after episode 15 the condition became valid and the weights initialization was executed, this time the initial weights of the red run were apparently

sufficient good for the agent to be able to converge to the optimal solution. Note that the convergence to sub-optimal solution issue could be dealt with in different ways, for example adding intermediate positive rewards the higher the agent climbs the hill.

2 Fine-tune an existing model

In this section and the next one we needed to use the weights and biases from source network and passes them to another network. To do that before initializing the main tensorflow session we load each one of the needed variables as constants and passed them as variables to the constructors of the Actor and the Critic classes.

2.1 Acrobot → CartPole

In this training, we first tried to load the actor network final variables (weights and biases) of the Acrobot single hidden layer and freeze them, but the model did not converge that way (also tried to vary the hyper parameters). Therefore, the next thing to try was to initialize the variables of CartPole using the Acrobat variables, but this time without freezing them. With this method, the model was converged. The hyper parameters and networks architectures used in this section were the same as those were used in the original CartPole training, which can be seen in table 1 and figure 1 respectively. The next figure shows how the fine-tuning actor network was built.

```
self.W1 = tf.get_variable('W1', initializer=Acrobot_model_W1)  # Without Freezing
self.b1 = tf.get_variable("b1", initializer=Acrobot_model_b1)  # Without Freezing
# self.W1 = Acrobot_model_W1 # If we were freeze
# self.b1 = Acrobot_model_b1 # If we were freeze
self.W2 = tf.get_variable("W2", [64, self.action_size],
                          initializer=tf.keras.initializers.glorot_normal(seed=0))
self.b2 = tf.get_variable("b2", [self.action_size], initializer=tf.zeros_initializer())

self.Z1 = tf.add(tf.matmul(self.state, self.W1), self.b1)
self.A1 = tf.nn.relu(self.Z1)
self.output = tf.nn.softmax(tf.add(tf.matmul(self.A1, self.W2), self.b2))
```

Figure 14: Acrobot to CartPole fine-tuning actor network

2.1.1 Training Results

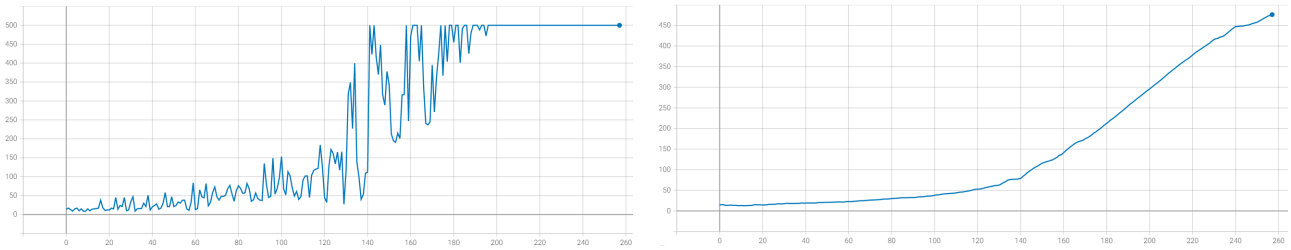
As can be seen in the next figure, it took 258 episodes for the agent to reach an average reward above 475 in consecutive 100 episodes. The time until convergence in this case was 1.8 minutes, and the cumulative reward in the test episode was 500.

```
Episode 254 | Reward: 500.00 | Average over 100 episodes: 466.69
Episode 255 | Reward: 500.00 | Average over 100 episodes: 469.54
Episode 256 | Reward: 500.00 | Average over 100 episodes: 472.53
Episode 257 | Reward: 500.00 | Average over 100 episodes: 474.37
Episode 258 | Reward: 500.00 | Average over 100 episodes: 476.20

Great!! You win after: 258 Episodes
Average reward in the last 100 episodes: 476.2
Training complete after 1.8 minutes
Total reward in the test: 500.00
```

Figure 15: Acrobot To CartPole summary of training

The next figure shows the resulted reward per episode as well as the average Reward in the consecutive 100 episodes



(a) **(b)**

Figure 16: Acrobot to CartPole - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

The next figure shows the Actor network loss per step.

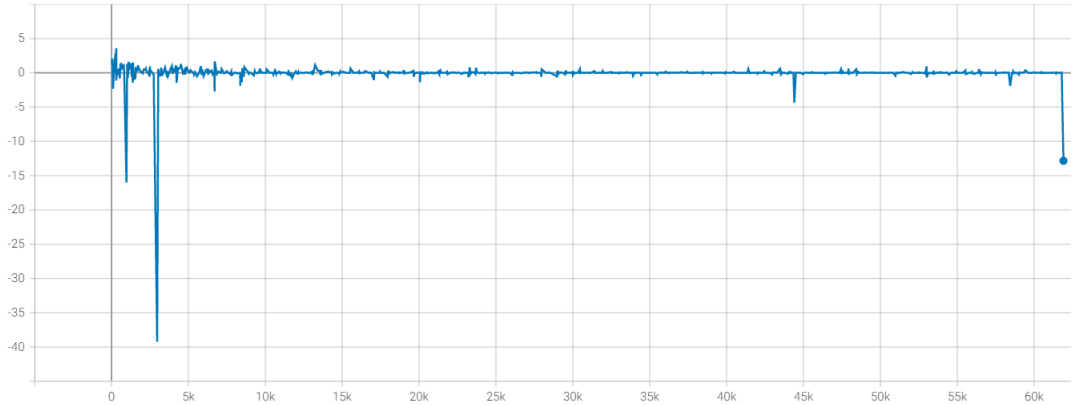


Figure 17: Acrobot to CartPole – Actor network loss per step

2.2 CartPole → MountainCar

In this training, we load the actor network final variables (weights and biases) of the CartPole single hidden layer to the MountainCar actor policy and freeze them. In this section the model converged without any issues. The hyper parameters and networks architectures used in this section were the same as those were used in the original MountainCar training, which can be seen in table 3 and figure 12 respectively. The next figure shows how the fine-tuning actor network was built.

```
self.W1 = CartPole_model_W1
self.b1 = CartPole_model_b1
self.W2 = tf.get_variable("W2", [64, 64],
                          initializer=tf.keras.initializers.glorot_normal(seed=0))
self.b2 = tf.get_variable("b2", [64], initializer=tf.zeros_initializer())
self.W3 = tf.get_variable("W3", [64, self.action_size],
                          initializer=tf.keras.initializers.glorot_normal(seed=0))
self.b3 = tf.get_variable("b3", [self.action_size], initializer=tf.zeros_initializer())

self.W_mean = tf.get_variable("W_mean", [self.action_size, 1],
                              initializer=tf.keras.initializers.glorot_normal(seed=0))
self.b_mean = tf.get_variable("b_mean", [1], initializer=tf.zeros_initializer())
self.W_std = tf.get_variable("W_std", [self.action_size, 1],
                             initializer=tf.keras.initializers.glorot_normal(seed=0))
self.b_std = tf.get_variable("b_std", [1], initializer=tf.zeros_initializer())

self.Z1 = tf.add(tf.matmul(self.state, self.W1), self.b1)
self.A1 = tf.nn.relu(self.Z1)
self.Z2 = tf.add(tf.matmul(self.A1, self.W2), self.b2)
self.A2 = tf.nn.elu(self.Z2)
self.output = tf.add(tf.matmul(self.A2, self.W3), self.b3)

self.mean = tf.nn.tanh(tf.add(tf.matmul(self.output, self.W_mean), self.b_mean))
self.std = tf.keras.activations.softplus(tf.add(tf.matmul(self.output, self.W_std), self.b_std)) + 1e-5
```

Figure 18: CartPole To MountainCar fine-tuning actor network

2.2.1 Training Results

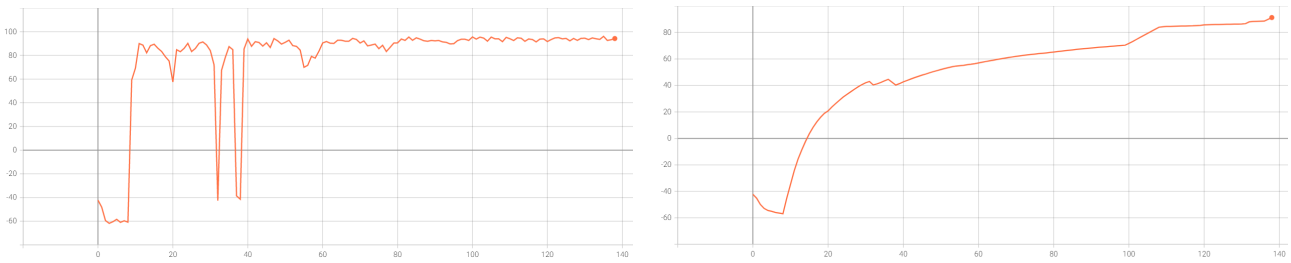
As can be seen in the next figure, it took 139 episodes for the agent to reach an average reward above 90.0 in consecutive 100 episodes. The time until convergence in this case was 1.6 minutes, and the cumulative reward in the test episode was 94.05.

```
Episode 201 | Reward: 98.00 | Average over 100 episodes: 89.77
Episode 202 | Reward: 96.85 | Average over 100 episodes: 89.83
Episode 203 | Reward: 96.51 | Average over 100 episodes: 89.97
Episode 204 | Reward: 96.45 | Average over 100 episodes: 90.04

Great!! You win after: 204 Episodes
Average reward in the last 100 episodes: 90.04
Training complete after 4.6 minutes
Total reward in the test: 97.83
```

Figure 19: CartPole to MountainCar summary of training

The next figure shows the resulted reward per episode as well as the average Reward in the consecutive 100 episodes



(a) (b)

Figure 20: CartPole To MountainCar - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

The next figure shows the Actor network loss per step.

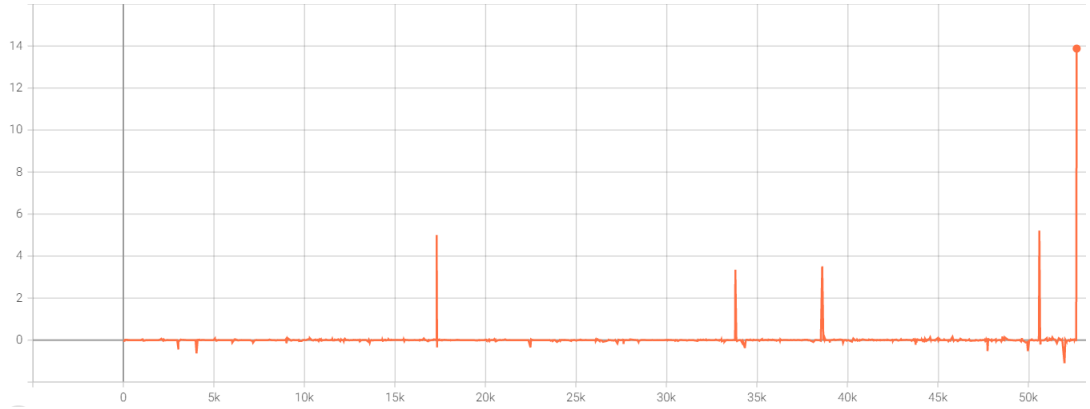


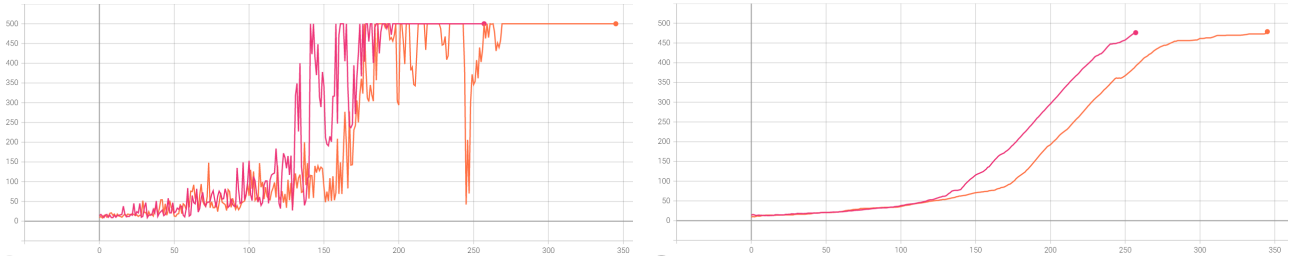
Figure 21: CartPole To MountainCar – Actor network loss per step

2.3 Comparison with section 1

In this section, we will compare the training results we got for the CartPole and MountainCar environments from section 1 by training individual networks to those we got from section 2 by fine tune an existing model.

2.3.1 CartPole

The next figure shows the cumulative reward along the two training runs. The network that learned individually is marked in orange color and the fine tune model marked in pink.



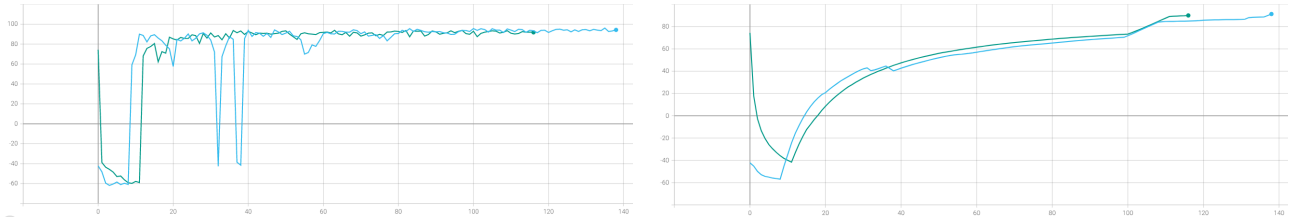
(a) (b)

Figure 22: CartPole comparison - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

As can be seen in the figures above, the fine-tuned CartPole model was converged faster, although the two runs seem to converge in a similar manner.

2.3.2 MountainCar

In the following figures the network that learned individually is marked in green color and the fine tune model marked in gray.



(a) **(b)**

Figure 23: MountainCar comparison - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

From the figures above, it can be seen that the fine-tune MountainCar model reached the peak score first, but because of instability around episodes 30-40, the individual network model converged faster.

3 Transfer learning

In this section, we implemented a simplified version of the Progressive Networks to the CartPole environment using the Acrobot,MountainCar environments actor networks from section 1, and the same for the MountainCar environment using the Acrobot,CartPole . According to the Progressive Networks method, each new layer in the target network was learned by the following expression.

$$h_i^{(k)} = f(W_i^{(k)} \cdot h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} \cdot h_{i-1}^{(j)}) \tag{1}$$

Note that since the MountainCar actor network from section 1.3 has more layers than the two others networks (CartPole and Acrobot environments), we transferred the hidden layers from the end of the network until the layers over to one of the models.

3.1 {Acrobot, MountainCar} → CartPole

In this training, we used the same networks architectures and hyperparameters values we used in section 1 for the CartPole environment, which can be seen in figure 1 and table 1

3.1.1 Training Results

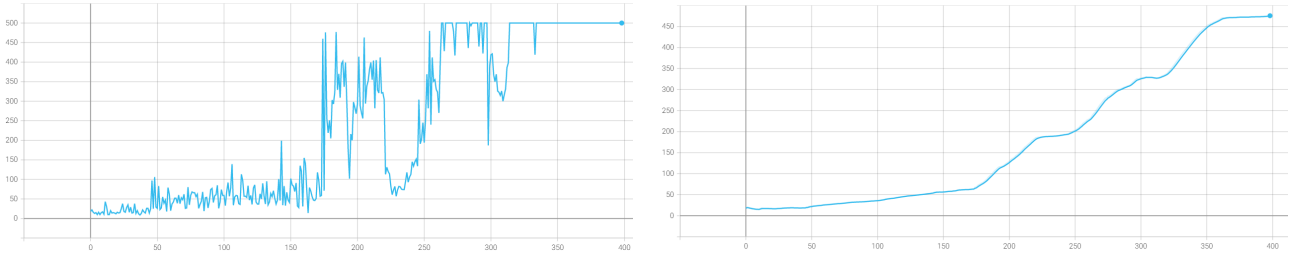
As can be seen in the next figure, it took 399 episodes for the agent to reach an average reward above 475 in consecutive 100 episodes. The time until convergence in this case was 2.8 minutes, and the cumulative reward in the test episode was 500.

```
Episode 396 | Reward: 500.00 | Average over 100 episodes: 474.45
Episode 397 | Reward: 500.00 | Average over 100 episodes: 474.45
Episode 398 | Reward: 500.00 | Average over 100 episodes: 474.45
Episode 399 | Reward: 500.00 | Average over 100 episodes: 477.58

Great!! You win after: 399 Episodes
Average reward in the last 100 episodes: 477.58
Training complete after 2.8 minutes
Total reward in the test: 500.00
```

Figure 24: {Acrobot, MountainCar} → CartPole summary of training

The next figure shows the resulted reward per episode as well as the average Reward in the consecutive 100 episodes



(a) **(b)**

Figure 25: {Acrobot, MountainCar} → CartPole - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

The next figure shows the Actor network loss per step.

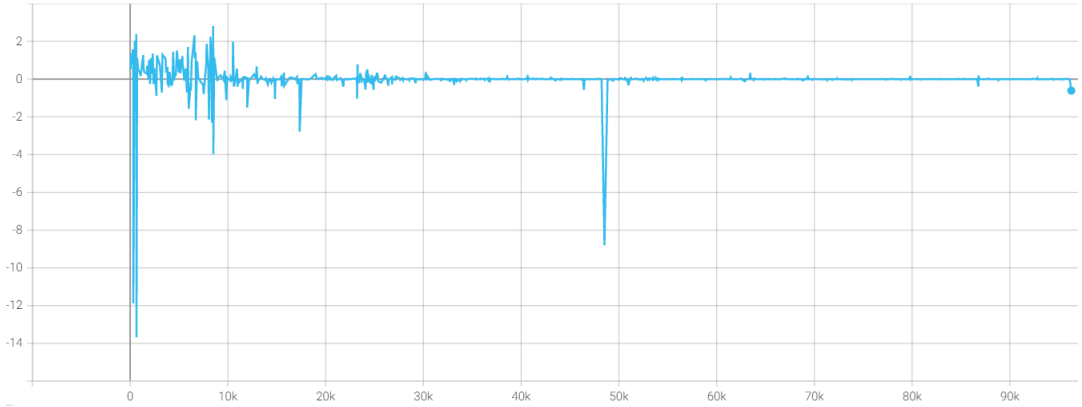


Figure 26: {Acrobot, MountainCar} → CartPole – Actor network loss per step

3.2 {Acrobot, CartPole} → MountainCar

In this training, we initially used the same networks architectures and hyperparameters values we used in section 1 for the MountainCar environment, but this training was not shown any signs of convergence. Consequently, we decided to try to change the actor network architecture without touching the learning rates which are already very low. Finally, after some trials and errors, we got to the simpler and compatible new actor network's architecture which led to good results. The changes we made relative to the network in figure 10 are to remove the second hidden layer, and to change the number of neurons in the first hidden layer from 64 to 12
The new architecture can be seen in the following figure.

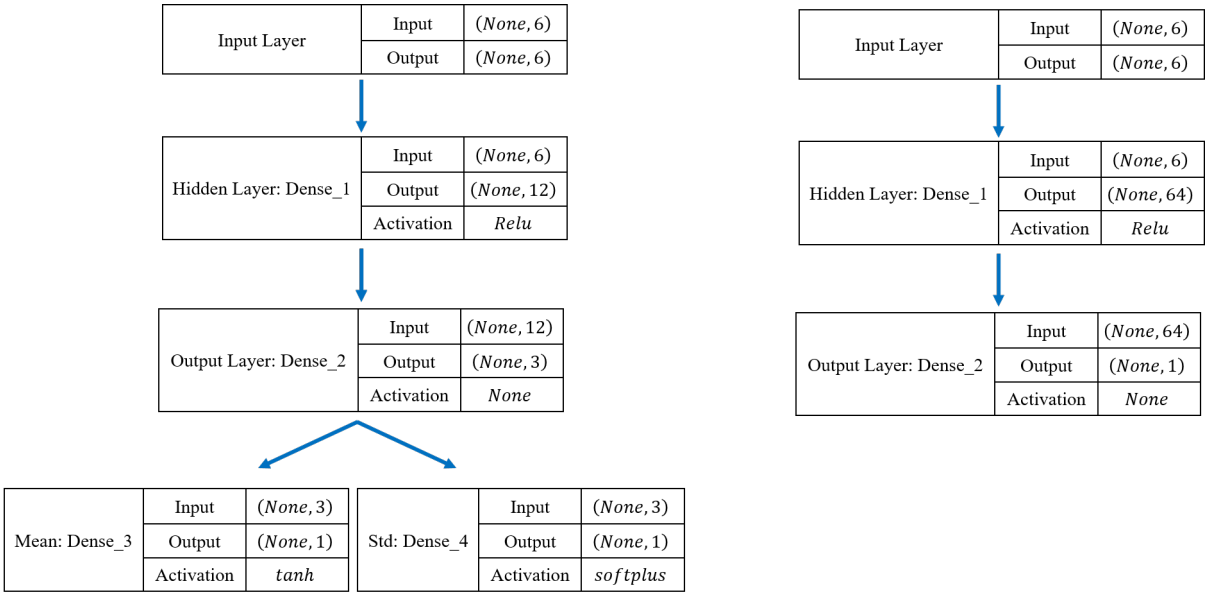


Figure 27: MountainCar Transfer learning - Actor network architecture in the left side and Critic network architecture in the right side

3.2.1 Training Results

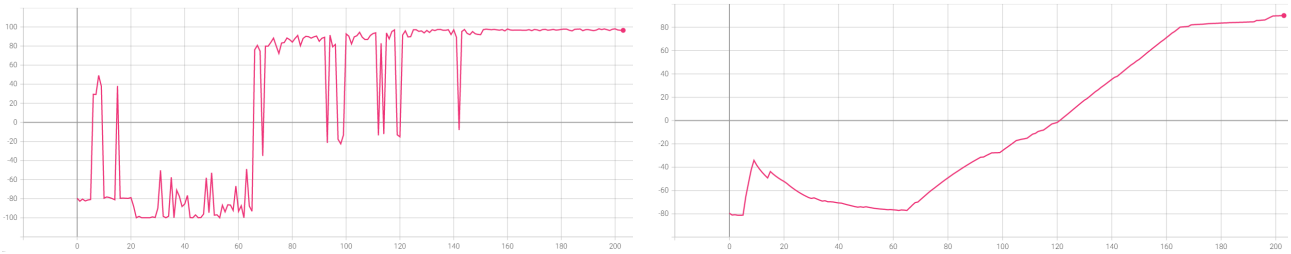
As can be seen in the next figure, it took 204 episodes for the agent to reach an average reward above 90 in consecutive 100 episodes. The time until convergence in this case was 4.6 minutes, and the cumulative reward in the test episode was 97.83.

```
Episode 201 | Reward: 98.00 | Average over 100 episodes: 89.77
Episode 202 | Reward: 96.85 | Average over 100 episodes: 89.83
Episode 203 | Reward: 96.51 | Average over 100 episodes: 89.97
Episode 204 | Reward: 96.45 | Average over 100 episodes: 90.04

Great!! You win after: 204 Episodes
Average reward in the last 100 episodes: 90.04
Training complete after 4.6 minutes
Total reward in the test: 97.83
```

Figure 28: {Acrobot, CartPole} → MountainCar summary of training

The next figure shows the resulted reward per episode as well as the average Reward in the consecutive 100 episodes



(a) **(b)**

Figure 29: {Acrobot, CartPole} → MountainCar - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

The next figure shows the Actor network loss per step.

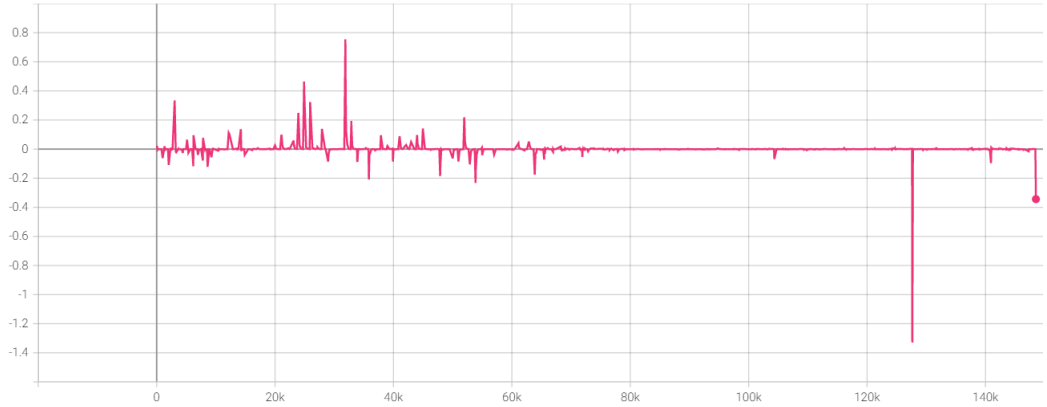


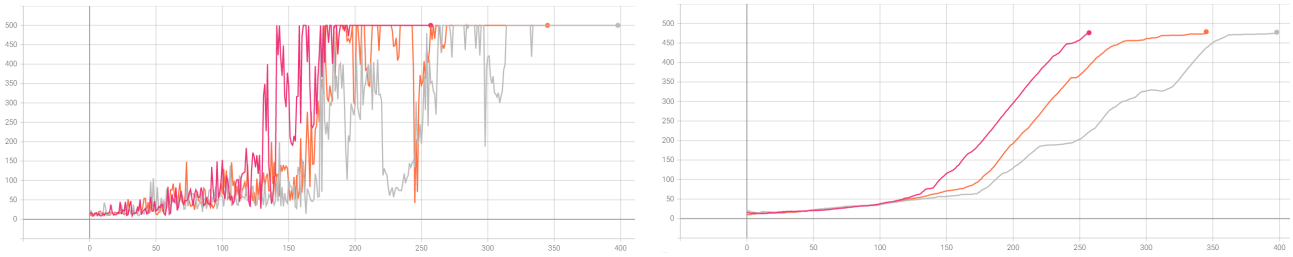
Figure 30: {Acrobot, CartPole} → MountainCar – Actor network loss per step

3.3 Comparison with previous sections

In this section, we will compare the training results we got for the CartPole and MountainCar environments from section 1 by training individual networks to those we got from sections 2 and 3 by fine tune an existing model and using transfer learning.

3.3.1 CartPole

The next figure show the cumulative reward along the different training runs for the CartPole environment. The network that learned individually is marked in orange color, the fine tune model marked in pink, and the transfer learning in gray.



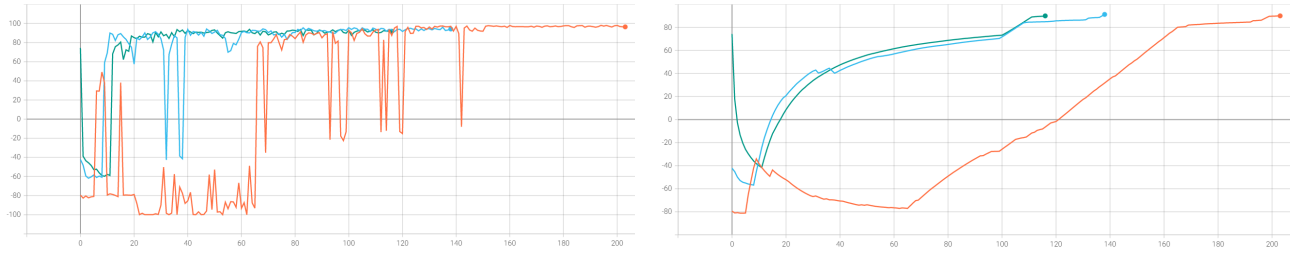
(a) **(b)**

Figure 31: CartPole comparison - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

As can be seen in the figure above, for the CartPole environment we got the fastest convergence by fine-tuning the existing model, but recall we did this without freezing the weights from the Acrobot environment, so it makes this learning more similar to the individual training except for the initial values of the network variables. The progressive network model took more time to converged relative to the other models, but still fast relative to the DQN algorithm we studied in Assignment 1.

3.3.2 MountainCar

The next figure show the cumulative reward along the different training runs for the MountainCar environment. The network that learned individually is marked in green color, the fine tune model marked in azure, and the transfer learning in orange.



(a) **(b)**

Figure 32: MountainCar comparison - (a) Reward per episode. (b) Average reward in consecutive 100 episodes

For the MountainCar problem, we got the best results for the individual model we implemented in section 1. However, the fine-tune model was very close to it. We can observe that the progressive network model took more time to converge, but

recall the number of trainable variables in this model was much smaller, which also affected in some way.

4 Code Execution Explanation

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.

Section 1:
CartPole : `python section1_Or_Ofir\CartPole_AC_Or_Ofir.py`
Acrobot : `python section1_Or_Ofir\Acrobot_AC_Or_Ofir.py`
MountainCar : `python section1_Or_Ofir\MountainCar_AC_Or_Ofir.py`

Section 2:
CartPole : `python section2_Or_Ofir\AcrobotToCartPole_Or_Ofir.py`
MountainCar : `python section2_Or_Ofir\CartPoleToMontainCar_Or_Ofir.py`

Section 3:
CartPole : `python section3_Or_Ofir\Transfer_CartPole_Or_Ofir.py`
MountainCar : `python section3_Or_Ofir\Transfer_MountainCar_Or_Ofir.py`