Deep Reinforcement Learning Exercise 3

Shahaf Yamin 311530943
Ron Sofer 204738637

February 2, 2022

Running Instruction for the script of this assignment.
Default command:

```
python policy_gradients_tf2.py
```

Optional arguments for all questions:

1. "−−N <n>", for n-step Actor-Critic (n>0), default 1.
2. "−−exp <exp_name>", to set experiment name, and save results under this name.
3. "−−n_episodes <i>" to set the number of max episodes, default 5000.
4. "−−gpu <x>" to set CUDA_VISIBLE_DEVICES environment variable, and use gpu indexed with x.
5. "−−explore_start <t>, to perform exploring start for t episodes before starting at initial state. Default 0.

- In Question 1 add the following arguments:
  1. "−−Q Q1"
  2. "−−dest <d>", where d∈ {acrobot, cartpole, mount}. This argument is the task we want to perform.
  3. We suggest using "−−N 16", when d is mount or cartpole.
  4. If d is mount, use "−−explore_start <t>", and "−−n_episodes <i>" to assist convergence. We suggest using t=4900, i=10000.
- In Question 2 add the following arguments:
  1. "−−Q Q2"
  2. "−−dest <d>", where d∈ {acrobot, cartpole, mount}. This argument is the task we want to perform.
  3. "−−src <s1>", where s1∈ {acrobot, cartpole, mount}. This argument is the pretrained network we want to load.
  4. We suggest using "−−N 16", when d is mount or cartpole.
  5. If d is mount, use "−−explore_start <t>" to assist convergence. We suggest using t=2000.
- In Question 3 add the following arguments:
  1. "−−Q Q3"
  2. "−−dest <d>", where d∈ {acrobot, cartpole, mount}. This argument is the task we want to perform.
  3. "−−src <s1> <s2>", where s1,s2∈ {acrobot, cartpole, mount}. These arguments are the pretrained networks we want to load.
  4. We suggest using "−−N 16", when d is mount or cartpole.
  5. If d is mount, use "−−explore_start <t>" to assist convergence. We suggest using t=500.

1 Introduction

In this work, we are going to examine the adaptation ability of our agent to a new task after he is trained over a different task. This technique is a well known method in supervised learning and it is called Transfer-Learning. In order to evaluate the agent adaptation ability, first, we are going to train three different agents independently against the following tasks:

- CartPole-V1
- Acrobot-V1
- MountainCarContinuous-V0

As a next step, we are planning to test the adaptation capabilities of our agents against different task than the one they were train on (We will give further details in the following sections). In the following subsections we will give a brief introduction for each task.

1.1 CartPole

The goal of CartPole is to balance a pole connected with one joint on top of a moving cart. We define the state space $\mathcal{S} \subseteq \mathbb{R}^4$ where each state $s \in \mathcal{S}$ contains the following information $s =$ [cart position, cart velocity, pole angle, pole angular velocity]. where each dimension belongs to the following sets:

- *cart position* $\in [-4.8, 4.8]$
- *cart velocity* $\in [-\infty, \infty]$
- *pole angle* $\in [-0.418, 0.418]_{radians}$
- *pole angular velocity* $\in [-\infty, \infty]$

The action space $\mathcal{A}$ is discrete $\mathcal{A} = \{push\ cart\ to\ the\ left, push\ cart\ to\ the\ right\} = \{0, 1\}$.
The reward is +1 for every step taken. An episode is terminated if:

- The pole angle goes beyond ± 12 degrees or ± 0.2094 rad.
- The cart position is larger than 2.4.

We define the score as the total reward (= total number of steps made) received in one episode. We evaluate our agent overall performance by using a metric which we term as "avg score", where we define "avg score" as the average score over 100 consecutive episodes. Our goal is to maximize the "avg score" value, and achieve "avg score" of at least 475.

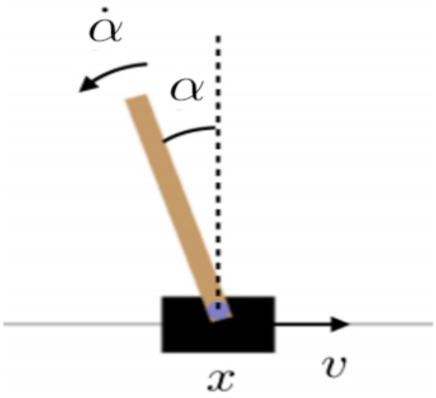


Figure 1: CartPole Illustration.

1.2 Acrobot

The Acrobot is a planar two-link robotic arm in the vertical plane (working against gravity), with an actuator at the elbow, but no actuator at the shoulder. The Acrobot is so named because of its resemblance to a gymnast (or acrobat) on a parallel bar, who controls his/her motion predominantly by effort at the waist (and not effort at the wrist). In this case, The goal is to swing the end of the outer-link to reach the target height. We define the state

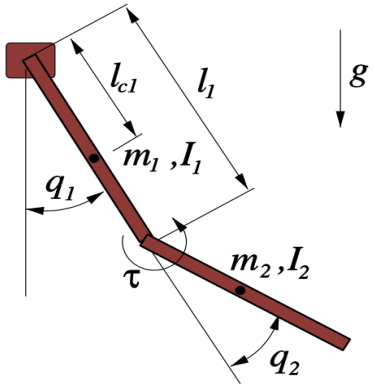


Figure 2: AcroBot Illustration.

space $\mathcal{S} \subseteq \mathbb{R}^6$ where each state $s \in \mathcal{S}$ contains the following information $s = [cos(q_1), sin(q_1), cos(q_2), sin(q_2), \text{angular velocity of } q_1, \text{angular velocity of } q_2]$. where each dimension belongs to the following sets:

- $cos(q_1) \in [-1, 1]$

- $sin(q_1) \in [-1, 1]$
- $cos(q_2) \in [-1, 1]$
- $sin(q_2) \in [-1, 1]$
- angular velocity of $q_1 \in [-4\pi, 4\pi]$
- angular velocity of $q_2 \in [-9\pi, 9\pi]$

The action space $\mathcal{A}$ is discrete and it determines the torque direction we apply to the elbow. $\mathcal{A} = \{torque\ left,\ no\ torque, torque\ right\} = \{-1, 0, 1\}$.
All steps that do not reach the goal (termination criteria) incur a reward of -1. Achieving the target height and terminating incurs a reward of 0. An episode is terminated if:

- The target height is achieved. As constructed, this occurs when $-cos(q_1) - cos(q_2 + q_1) > 1.0$
- Episode length is greater than 500.

We define the score as the total reward (= total number of steps made) received in one episode. We evaluate our agent overall performance by using the "avg score" metric. Our goal is to maximize the "avg score" value, and reach an "avg score" of at least -90.

1.3 MountainCarContinuous

In this environment, there is an underpowered car whos goal is to climb a one-dimentional hill to reach a goal, while using minimum energy necessary. The goal is on top of the hill on the right-hand side of the car, represented by the flag. We define the state space $\mathcal{S} \subseteq \mathbb{R}^2$ where each state $s \in \mathcal{S}$ contains the

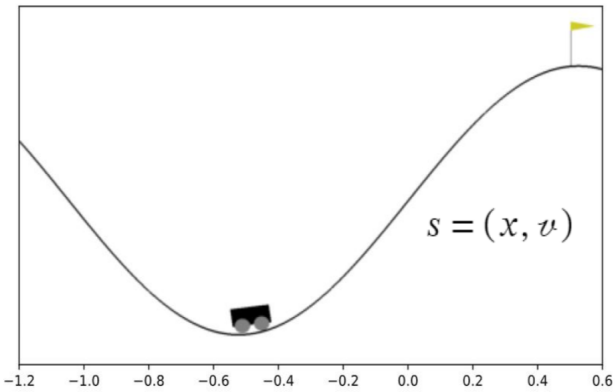


Figure 3: Mountain-Car Illustration.

following information $s = [x, v]$. where each dimension belongs to the following sets:

- $x \in [-1.2, 0.6]$
- $v \in [-0.07, 0.07]$

The action space $\mathcal{A}$ is continuous and it determines the force we apply on the car, $\mathcal{A} = [-1, 1] \subseteq \mathbb{R}$. All steps that do not reach the goal (termination criteria) incur a reward of -$|a|$, where a was the action taken. Reaching the goal and terminating incurs a reward of 100. An episode is terminated if:

- The goal was reached.
- Episode length is greater than 200.

We define the score as the total reward (= summation of rewards along the trajectory) received in one episode. We evaluate our agent overall performance by using the "avg score" metric. Our goal is to maximize the "avg score" value, and reach an "avg score" of 90.

2 Question 1

In this question, we have trained 3 independent agents to solve the aforementioned tasks. In order to transfer the knowledge between those agents in the next questions, we have chosen to use the same network architectures (Actor and Critic) between all tasks. The architecture is presented in Fig.4.

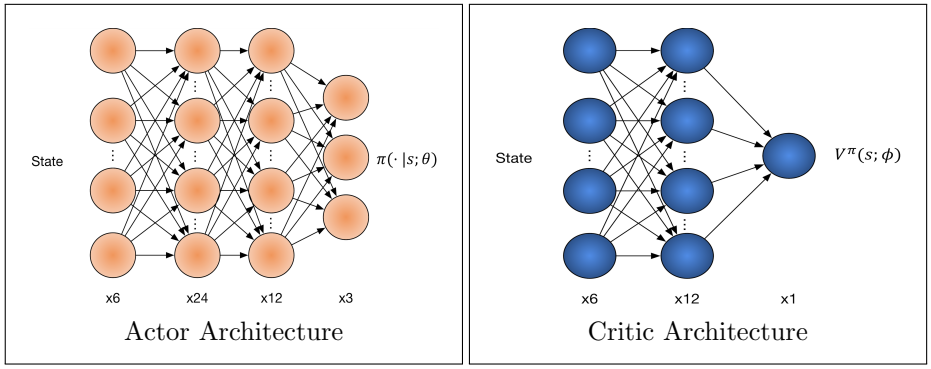


Figure 4: Actor and Critic Network Architecture

The final settings we used for solving each environment:

- Cartpole environment: we used N-steps TD error (as the Advantage estimator), when N equals 16:

$$\delta_t = \sum_{k=0}^{N-1} \gamma^k \cdot R_{t+k+1} + V^\pi(S_{t+N}; \phi) - V^\pi(S_t; \phi) \tag{1}$$

.

- Acrobot environment: we used 1-step TD error.
- Mountain-Car environment: we first used a discretization of the action-space. From $\mathcal{A} = [-1, 1]$ to $\mathcal{A} = \{-1, 1\}$. 1-step TD error, and exploring-starts for K episodes before using the default starting state.

We use the same learning-rates ($\eta_{Actor}, \eta_{Critic}$), discount-factor ($\gamma$), loss functions ($L_{Actor}, L_{Critic}$) and Adam optimizer for all tasks:

- $\eta_{Actor} = 0.0004$
- $\eta_{Critic} = 0.0012$
- $\gamma = 0.99$
- $L_{Actor} = Categorial\ Cross\ Entropy\ Loss$ (multiplied by the Advantage)
- $L_{Critic} = Mean\ Squared\ Error\ Loss$

Rationales behind these settings is presented in the following subsections, under each task separately.

2.1 Cartpole

We use an actor-critic (REINFORCE with baseline) framework explained in the section's introduction. Although we already implemented an actor-critic based agent that mange to solve this environment in the previous exercise, we have chose to use different network architecture for this agent in order to improve the network capability to generalize. We have made this change since our main goal is to be able to transfer knowledge between the tasks. In addition, we already observed in previous tasks that increasing the network size may cause training instability issues. Thus, as part of the solution of this task, we applied a technique that empirically improves the training stability. Our technique was to use a 16-step estimated TD error as our Advantage function instead of the traditional 1-step TD error estimation. It should be mentioned that we have tried to use 1-step TD error estimation but that cause instability issues, which lead our algorithm to fail as we can see in Fig. 5. Now, we summarize the training statistics and duration in Table 1:

Table 1: Cartpole Train Statistics and Duration

N	Episodes	Time
1	Failed	
16	871	**6h 21m**

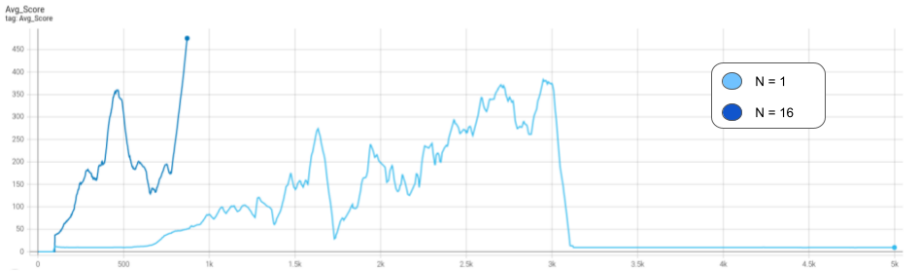


Figure 5: CartPole N=1 Vs N=16

2.2 Acrobot

We use an actor-critic (REINFORCE with baseline) framework explained in the section's introduction, with 1-step estimated TD error as our Advantage function (the baseline function). We tried to use 16-step TD error as well we've seen empirically that using more steps in this task comes with more episodes until convergence, and therefore, more time, this phenomena is shown in Fig. 6. We believe the reason behind this to be that in Acrobot, the reward signal is always non-positive and mostly negative. Therefore, when increasing the size of N (for increasing the quality of the TD error estimation δ) we receive a pessimistic estimation of the state's value. We can observe the N-step td-error estimation at the following equation.

$$\delta_t = \sum_{k=0}^{N-1} \gamma^k(-1) + V^\pi(s_{t+N};\phi) - V^\pi(s_t;\phi) \tag{2}$$

From this equation we can see that the first term is constant for most of the

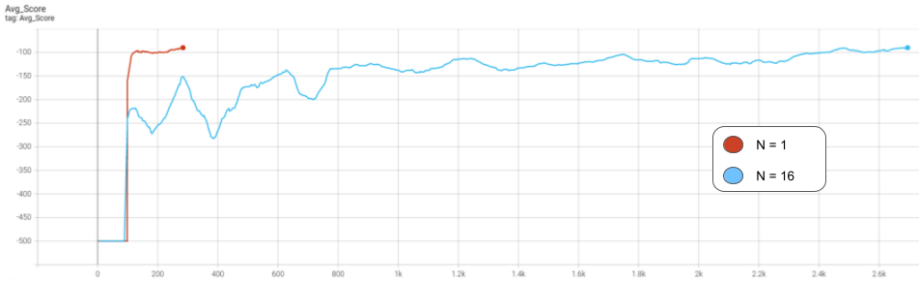


Figure 6: Acrobot N=1 Vs N=16

states, actually it is constant for all the states in our trajectory until the last N states. We can consider the initial V^π to be some sort of noise, and by adding large negative values we induce a negative bias which implies "bad state" to the current state value estimation, while if we use 1-step estimation - our agent will tend to be less pessimistic, since the initial "noise" will have more influence on

the value estimation. Another aspect of this phenomena is the trade-off between exploration and exploitation, one may say that the noisy estimation increase the "curiosity" of our agent. This in turn lead us to search for new unexplored states, and that increases our agent chances for find a "good" enough policy.
Now, we summarize the training statistics and duration in Table 2:

Table 2: Acrobot Train Statistics and Duration

N	Episodes	Time
1	284	**2h 11m**
16	2695	15h 13m

2.3 Mountain Car

We use an actor-critic (REINFORCE with baseline) framework explained in the section's introduction, while same as CartPole, we used 16-step estimation of the TD error. This task is considered ill conditioned, due to its reward signal, which raises an exploration challenge. This problem raises when our agent doesn't reach the goal soon enough. Under this scenario, since the agent is not aware to the flag/goal it will probably figure out that it is better not to move at all and wait at the bottom of the hill. Therefore, the agent will stop looking for the goal. We shall mention that this solution is a sub-optimal solution which many algorithms suffer from converging to. We shall also note that this reward is unusual with respect to most published work, where the goal was to reach the target as fast as possible, hence favouring a bang-bang strategy.
As part of this work, we've tried to overcome this challenge in many ways. Thriving for stability and consistency, our solutions were

- Action Discretization.
- 'Guidance' via reward shaping.
- Exploring-Starts.

2.3.1 Action Discretization

Since the agent "pays" the absolute value of his action in every step, as expected we receive a preference toward actions that are closer to 0. As we can see at the following continuous action histogram in Fig.7. We simplified the action space by forcing the agent to choose whether 1 or -1, hence the reward is constant for each step, and the agent can't converge to the "do-nothing" policy. It should be mentioned that this step by itself is not enough for the agent to reach the goal. Although both trials failed, we chose to continue with discrete action space due to the fact that in continuous action space, the agent usually converge to the "do-nothing" policy. We shall note that we have tried several attempts to use the continuous action-space, while using energy-based technique to enforce

stochastic policy instead of the "do-nothing" policy, but these attempts also failed since the first moment of our policy's Gaussian distribution still converged to zero while in order to preserve high energy, our agent has chosen to use a large second moment. To visualize the policy at every time-step - we've generated a thousand samples from a Gaussian distribution with the network's output as first and second moments. These samples were passed through a tanh function to map the results to [-1,1]. The policy is shown in Fig.7, with and without Energy-based loss. As explained earlier, the energy-based loss causes the second

Figure 7: Policy at time-step t, with and without Energy-based loss. Notice the support of each distribution

moment to be large, but yet the first moment is close to 0 at every time step, without visible dependency on the input state. Therefore these two trials have failed to reach the goal, and we used a discrete action space instead.

2.3.2 'Guidance' via reward shaping

As a next step, we have tried to give our agent guidance towards the "correct" behavior via an additional hand crafted reward which we can see in the following equations:

$$s_t = (x_t, v_t) \quad d_t = \begin{cases} -1 & , x_t < -0.5 \\ 1 & , x_t >= 0.5 \end{cases} \quad M_{v_t} = \begin{cases} -1 & , d_t \cdot v_t < v_0 \\ 1 & , d_t \cdot v_t \geq v_0 \end{cases}$$
$$r_t^i = M_{v_t} \cdot a_t \cdot d_t$$
$$\hat{r}_t = r_t + r_t^i$$

Where v_0 represents a small positive value, we used 10^{-3}. The logic behind this reward is that we want to push our agent towards our desired behaviour, we are doing this by giving him good feedback if he does what we expect him to do in this state and vice versa. In addition, one can look at this reward-shaping method as a private case of imitation learning where we try to "teach" our agent to follow our "expert" behaviour via the reward signal. We can see in Fig 9, that our agent in this case have managed to understand the logic behind our "expert" decision and he constantly achieve better and better guidance reward. We shall mention that we didn't like this method very much since this method

induce our "human" bias over the agent's policy. Therefore, we have looked for another way to overcome the exploration challenge. In Fig.8 we demonstrate how the parameters from the equation above behave. We would like the agent to push left either when he is in the right side ($d_t = 1$) and his velocity is negative or relatively small ($M_t = -1$), or when he's on the left side ($d_t = -1$) and his velocity is negative and relatively large ($M_t = 1$). On the other hand, we want the agent to push right whenever he is in the right side ($d_t = 1$) and his velocity is positive and relatively large ($M_t = 1$), or whenever he is in the left side ($d_t = -1$) and his velocity is positive or relatively small negative. Fig.

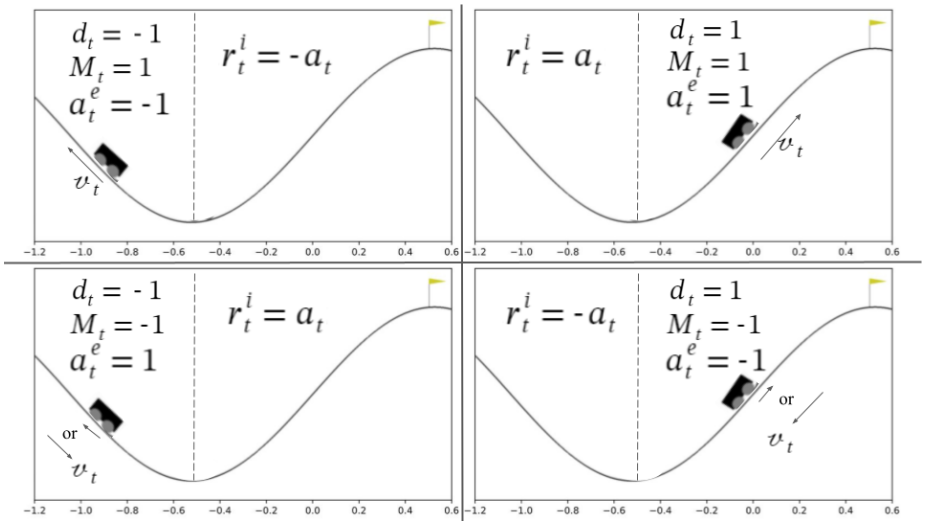


Figure 8: Guidance signal intuition, we denote a_t^e as the "expert's" action at time step t. We should note that the guidance signal is maximized when the agent follow the "expert's policy"

9 illustrates the agent performance while using our hand-crafted reward signal. As we can see, as we further the agent learns, his capability of following the "Guidance" signal increases. In addition, we can see that the length of the intervals between the "Guidance" signal violations is with positive correlation with the time-step. Moreover, we conclude that by following the "Guidance" signal, the agent has managed to solve the task successfully, and **very fast** (only 138 episodes, and 1h 38m).

2.3.3 Exploring-Starts

As a last resort, we have tried to overcome the exploration challenge by using the well-known exploring start method. This method randomly changes the starting position and velocity. As a first step, in order to fairly evaluate our agent performance against this task, we have tried the exploration start method for a predefined constant interval at the beginning of the training procedure.

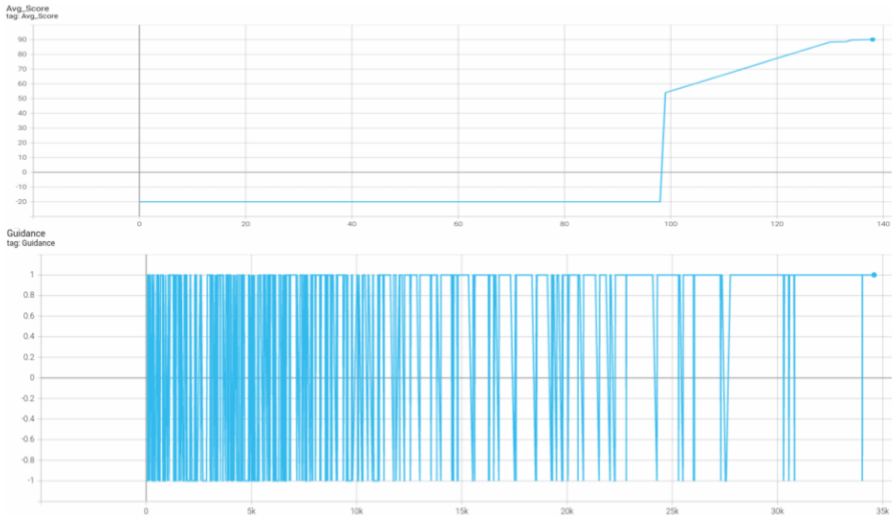


Figure 9: Agent's performance by following the Guidance signal.

After that, we proceed with a new training paradigm that doesn't involve any special exploration methods. Fig 10 illustrates our method's empirical results against different exploring start parameters. One may see that even an agent that was capable to solve the task multiple consecutive times, suffer from degradation of performance when we are turning off the exploring start mechanism. Also, we evaluate the influence of the number of steps in our TD error estimation which is shown in Fig. 11. Now, we summarize the training statistics and duration in Table 3: In conclusion, from our results we see that our explo-

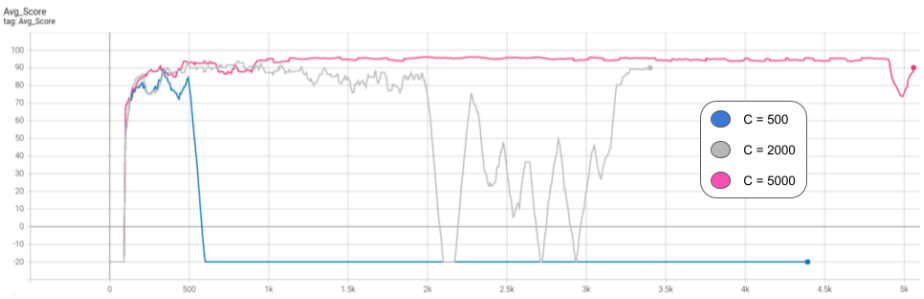


Figure 10: Different Exploring start interval length (C) performance evaluation.

ration method combined with the aforementioned discretization technique has managed to solve the task. In addition, we observe that the exploring start interval length is **crucial** for a successful training paradigm. As we can see, using too short exploring start interval (500 episodes) is not enough for the agent in this scenario, and he fails at solving the task. Moreover, we see that

Figure 11: Different N-step Performance evaluation.

Table 3: Mountain Train Statistics and Duration (N=16)

Exploring-Ends	Episodes	Time
500	Failed	
2000	3401	12h 16m
5000	5060	**8h 47m**

5000 episodes has better performance than 2000 in terms of training overall duration. We explain this phenomena by the fact that our agent has shorter episodes when he is able to successfully solve the task, while also an increased exploring start interval leads to better understanding of the environment's dynamics. Another conclusion is that using too noisy TD error estimation leads to the well-known problem of instability. We solve this problem by increasing the quality of the TD-error estimation by using more steps, which we've found to be also **crucial** for a successful training paradigm. We shall mention that from this point onward based on the aforementioned results, we use the method of action discretization combined with exploring start for this task.

3 Question 2

As part of this question, we have tried to transfer the agent's knowledge from one task to the other while as part of the transfer we reinitialize the last layer's weights. The following subsections summarizes our result and conclusion.

3.1 Transferring to CartPole

We have tried to transfer the knowledge from both Mountain and Acrobot agents that we have trained in the previous question. Fig. 12 shows the different training paradigm and Table 4 summarizes the training statistics. As we can see from the results, using transfer-learning indeed succeed and we've managed to solve the task by using less experience. Intuitively speaking, we explain this

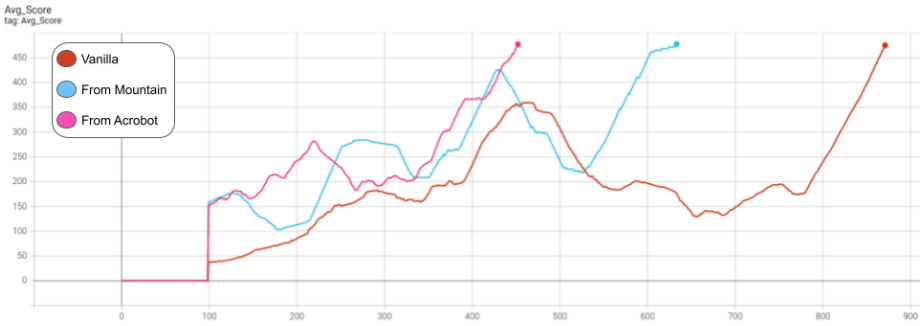


Figure 12: Transfer Learning To CartPole.

Table 4: Transfer Learning Cartpole Train Statistics and Duration (N=16)

Weight Source	Episodes	Time
Random Initialization	871	6h and 21m
Acrobot (Transfer)	**452**	**3h and 38m**
Mountain-Car (Transfer)	663	5h and 39m

phenomena by the fact that the agent managed to leverage his past knowledge and apply it for solving this task. In addition, we can see that the knowledge from Acrobot was more influential than the knowledge from MountainCar, In our opinion, this happens because there is higher similarity between the Acrobot and the CartPole environments.

3.2 Transferring to Mountain

Our next task was to transfer the knowledge from Cartpole to Mountain. Fig. 13 shows our results and Table 5 summarizes the training statistics. As we can

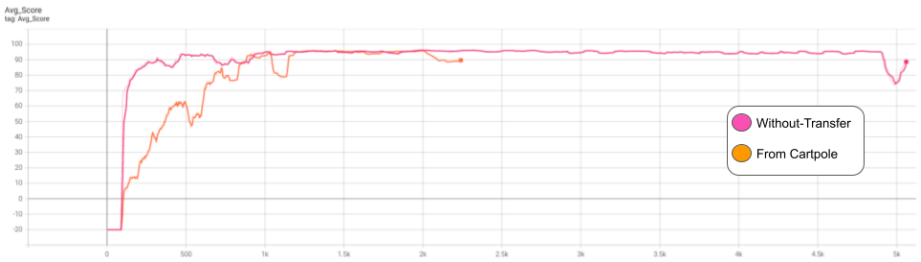


Figure 13: Transfer Learning To Mountain.

see, by using transfer learning we've managed to converge faster, the reason is that now we don't really need to explore for 5000 episodes as we had to do for random weight initialization and using an interval of 2000 episodes for exploring

Table 5: Transfer Learning Cartpole Train Statistics and Duration (N=16)

Weight Source	Episodes	Time
Random Initialization	5060	8h and 47m
Cartpole	**2240**	**7h and 11m**

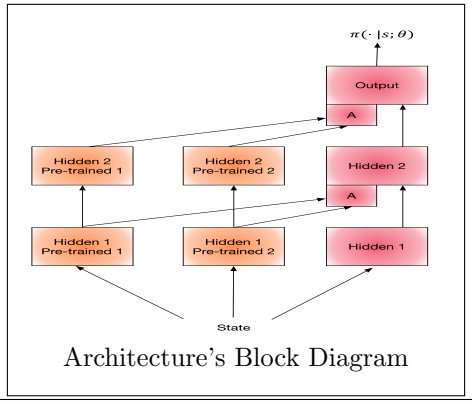
start was sufficient. We believe the reason behind this result lies in the fact that our agent managed to leverage his previous understanding of the CartPole environment during the training procedure over the Mountain Car environment. Since that, he is capable of catching the complex logic behind solving the task faster and doesn't need such a big amount of exploration. We conclude this by observing both the training paradigm and the training statistics, where in both one may observe that transfer-learning has superior performance than the random weight initialization method.

4 Question 3

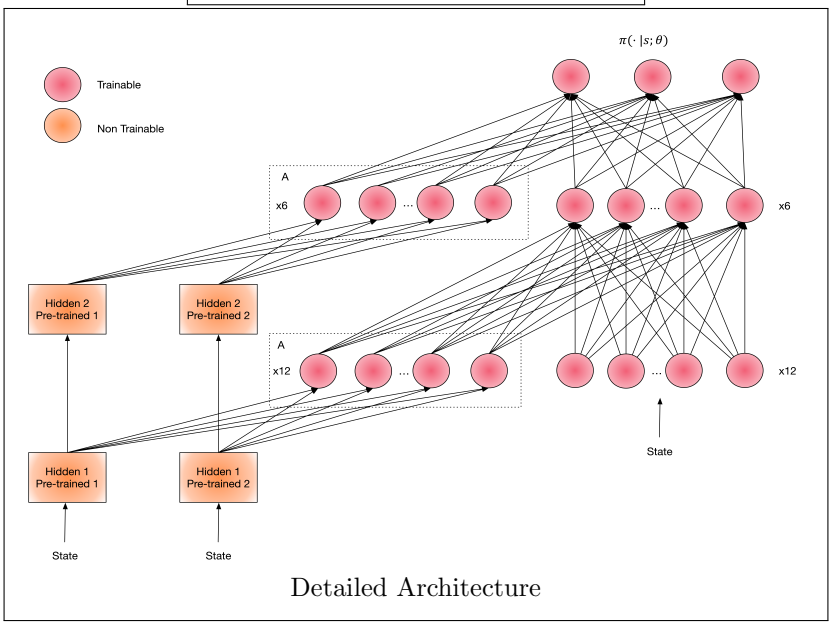
As part of this question, we have tried to transfer the agent's knowledge from 2 different tasks to the last task while as part of the transfer we use a whole new architecture which is a simplified version of Progressive Network. Fig 14 illustrate this architecture. Throughout this question we've used the same architecture as we've used at the previous questions for each pretrained network. We shall note, that the amount of trainable-parameters in our progressive-network architecture (1020) is about twice the amount of trainable-parameters in the architecture of the previous questions (507). Therefore, we receive more expressiveness, i.e., our model is capable of approximating more complex functions. In general, training paradigms for large models tend to suffer from instability issues, as we already seen in previous exercises. By using the pretrained networks, we are capable of using larger models, while preserving the training paradigm's stability. In addition, in the progressive network we use linear layers for weighting the input from each hidden-layer in the pretrained networks. Intuitively, we would like to weight close to 0 irrelevant information from those layers, and weights far from 0 for relevant information. These linear layers are marked as A in Fig.14. The following subsections summarizes our result and conclusion.

4.1 Training Progressive Network to solve Cartpole

Our first task was to transfer the knowledge from Acrobot and Mountain-Car agents to solve the Cartpole task. Fig. 15 shows our results and Table 6 summarizes the training statistics. As we can see, using the previous knowledge of the 2 tasks allowed us to converge to a solution, even though the architecture is twice the size of the previous architecture. We believe that in this scenario, the use of more parameters resolved in more training steps needed, although the task was performed fairly in most of the episodes, which made it to train



Architecture's Block Diagram



Detailed Architecture

Figure 14: Progressive Network Architecture

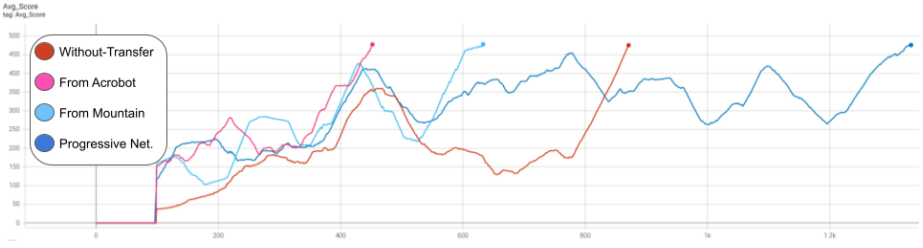


Figure 15: Transfer Learning to Cartpole.

Table 6: Transfer Learning Cartpole-Task Train Statistics and Duration (N=16)

Weight Source	Episodes	Time
Random Initialization	871	6h and 21m
Acrobot (Transfer)	**452**	**3h and 38m**
Mountain-Car (Transfer)	663	5h and 39m
Acrobot, Mountain-Car (PN)	1333	22h and 21m

for a very long time. Because in this environment, the better you play, the more time each episode takes, thus we observe major time gaps between this method and the others. In our opinion, one of the reasons for this phenomena is that this task is considered simple, therefore the use of 'Big-Guns' such as the progressive-network is not preferable here.

4.2 Training Progressive Network to solve MountainCar

Our next task was to transfer the knowledge from Cartpole and Acrobot agents to solve the Mountain Car task. Fig. 16 shows our results and Table 7 summarizes the training statistics. As we can see, using the previous knowledge

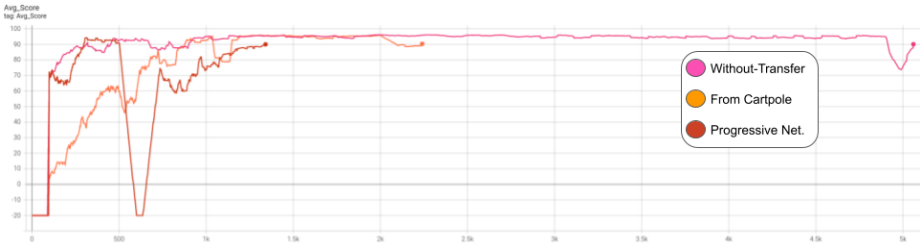


Figure 16: Transfer Learning to Mountain Car.

Table 7: Transfer Learning Mountain-Car-Task Train Statistics and Duration (N=16)

Weight Source	Episodes	Time
Random Initialization	5060	8h and 47m
Cartpole (Transfer)	2240	7h and 11m
Cartpole, Acrobot (PN)	**1341**	**6h and 53m**

of the 2 tasks resolved in faster convergence both time-wise and episode-wise, since in this case we were able to use only 500 episodes as our exploration start interval and yet converge. We also tried to perform 'simple' transfer learning (as in question 2) while using 500 episodes for exploration-start. These attempts failed resolutely. This can imply that the use of 2 pretrained networks as well

as using more complex architecture enabled the agent to exploit the previous knowledge efficiently, and 'understand' the environment better and faster.