Deep Reinforcement Learning

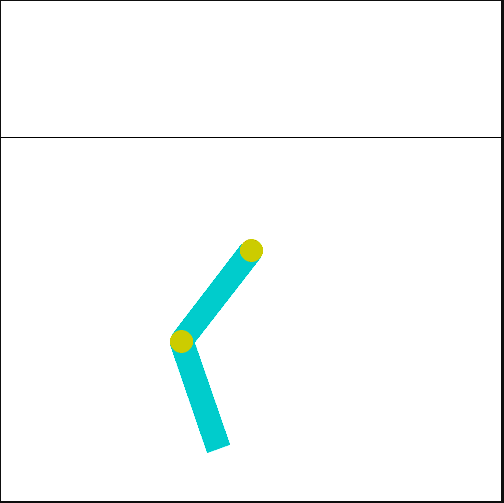
Assignment 2

Dor Shmuel - 312179971 | Yuval Cohen - 314978198 | 14/12/2022

# Section 1 – Training individual networks

## Actor-Critic algorithm for “Acrobot-v1”

**Description**: two blue links connected by two green joints. The joint in between the two links is actuated. The goal is to swing the free end of the outer-link to reach the target height (black horizontal line above the system) by applying torque on the actuator. [1]

****

**Action space shape**: Discrete(3)

**Observation shape**: (6,)

# Section 2 – Advantage Actor-Critic

## 2.3 – Implementation of Actor Critic

In the following, we will describe the relevant information for the implementation of actor-critic algorithm in the cart-pole v1 environment (as detailed in assignment 1).

**Architectures**

1. Architecture of the actor – Policy Network:

class PolicyNetwork:

    def \_\_init\_\_(self, state\_size, action\_size, learning\_rate, name='policy\_network'):

        self.state\_size = state\_size

        self.action\_size = action\_size

        self.learning\_rate = learning\_rate

        with tf.variable\_scope(name):

            self.state = tf.placeholder(tf.float32, [None, self.state\_size], name="state")

            self.action = tf.placeholder(tf.int32, [self.action\_size], name="action")

            self.A\_t = tf.placeholder(tf.float32, name="discounted\_advantage")

            self.I\_factor = tf.placeholder(tf.float32, name="I\_factor")

            tf2\_initializer = tf.keras.initializers.glorot\_normal(seed=0)

            self.W1 = tf.get\_variable("W1", [self.state\_size, 12], initializer=tf2\_initializer)

            self.b1 = tf.get\_variable("b1", [12], initializer=tf2\_initializer)

            self.W2 = tf.get\_variable("W2", [12, self.action\_size], initializer=tf2\_initializer)

            self.b2 = tf.get\_variable("b2", [self.action\_size], initializer=tf2\_initializer)

1. Architecture of the critic – Value Network:

class ValueNetwork:

    def \_\_init\_\_(self, state\_size, learning\_rate, name='state\_value\_network'):

        self.state\_size = state\_size

        self.learning\_rate = learning\_rate

        with tf.variable\_scope(name):

            self.state = tf.placeholder(tf.float32, [None, self.state\_size], name="state")

            self.A\_t = tf.placeholder(tf.float32, name="discounted\_advantage")

            self.R\_t = tf.placeholder(tf.float32, name="total\_rewards")

            self.I\_factor = tf.placeholder(tf.float32, name="I\_factor")

            tf2\_initializer = tf.keras.initializers.glorot\_normal(seed=0)

            self.W1 = tf.get\_variable("W1", [self.state\_size, 64], initializer=tf2\_initializer)

            self.b1 = tf.get\_variable("b1", [64], initializer=tf2\_initializer)

            self.W2 = tf.get\_variable("W2", [64, 16], initializer=tf2\_initializer)

            self.b2 = tf.get\_variable("b2", [16], initializer=tf2\_initializer)

            self.W3 = tf.get\_variable("W3", [16, 1], initializer=tf2\_initializer)

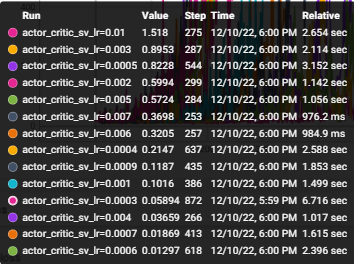
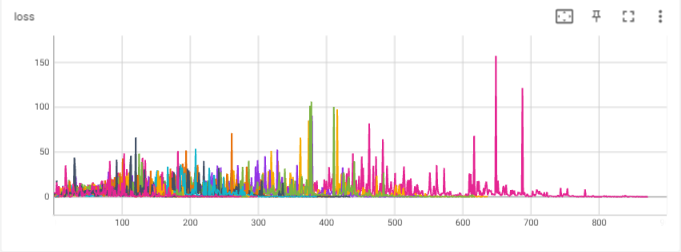
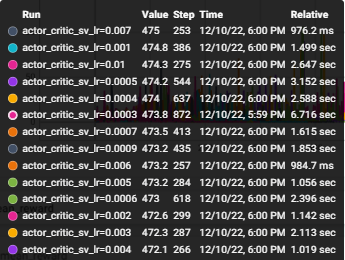
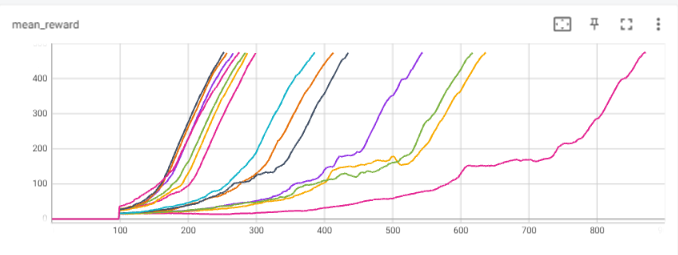
            self.b3 = tf.get\_variable("b3", [1], initializer=tf2\_initializer)

**Hyper-parameters Tuning**

Actor-critic algorithm is implemented over the discussed environment, in order to reach a mean reward of 475 points for 100 consecutive episodes. For optimal tuning we set the same seed through all the tuning procedure.

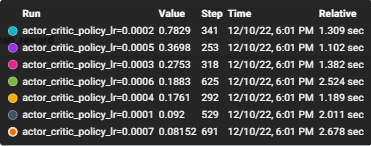
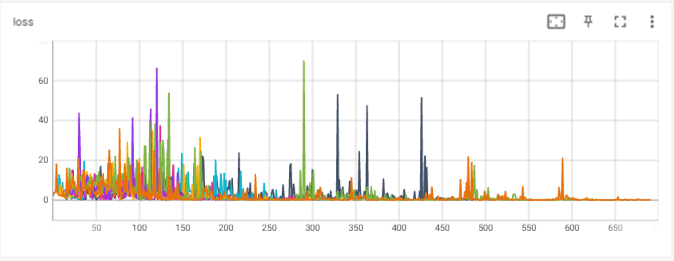
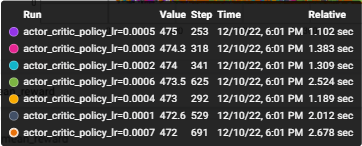
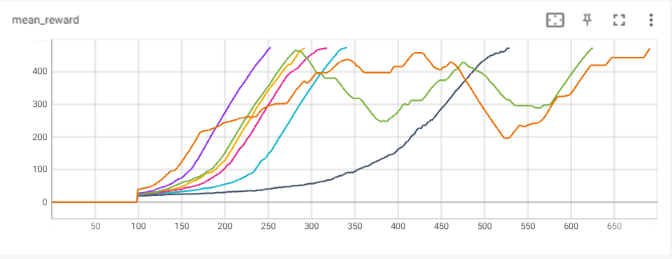
1. **learning rate of critic-value network tuning**:

reference values for tuning: actor learning rate: 0.0005, discount factor: 0.99.  
for learning rate **lr=0.007** the agent converged into the desired goal (mean award = 475) after **253** episodes.



1. **learning rate of actor- policy tuning**:

reference values for tuning: critic learning rate: 0.007, discount factor: 0.99.  
for learning rate **lr=0.0005** the agent converged into the desired goal (mean award = 475) after **253** episodes.



1. **discount factor**:

Chart, line chart

Description automatically generatedreference values for tuning: critic learning rate: 0.007, actor learning rate: 0.0005.  
for discount factor of **df=0.99** the agent converged into the desired goal (mean award = 475) after **386** episodes.

**Optimal hyper-parameters:**

|  |  |  |
| --- | --- | --- |
| Learning rate – policy (actor) | Learning rate – value (critic) | Discount factor |
| 0.0005 | 0.007 | 0.99 |

Text

Description automatically generatedChart, line chart

Description automatically generatedUnder those chosen parameters, we compared the actor-critic, policy gradients with baseline and the original policy gradient algorithms:

**Time for convergence:**

Actor critic: , Trained over GPU : Nvidia RTX 3070

We can observe the following:

1. In the manner of **performances**, we can notice that actor-critic algorithm converges way faster than the other algorithms, reaching the desired goal within lower number of episodes(253 vs 386 vs 772).
2. We can also notice that the mean award over 100 consecutive episodes curve more smother than other algorithms curves indicating that the algorithm is more stable. The stability of the actor-critic model is also reflected in the rewards plot, presenting almost constant reward of 500 for last 100 episodes, and moderate rise in results, without aggressive fluctuations comparing to the other algorithms.
3. The actor-critic also hits the goal score after 165 episodes while reinforce with baseline algorithm achieves the maximal rewards after 183 episodes.
4. Algorithm actor critic converged after 95.3062 seconds, while reinforce with baseline converged after 129.320 seconds (both trained over GPU: Nvidia RTX 3070 8Gb). Since the actor critic algorithm updates the weights of both networks in each step instead of each episode, each operation takes longer.

The improvement of the actor-critic algorithm is obvious since the update of the policy and the value network is done much frequently, in each step and not in each episode end as it is in reinforce algorithms. (Update can be done during episode and not only in the end of it).

**Running the script**:  
To run the above simulation, run the script **actor\_critic.py**.