Reacher Project

Project Description

The project consists of solving the Reacher environment which is a double-jointed arm that moves to target locations.

- State Space: consists of 33 continuous variables corresponding to the position, rotation, velocity, and angular velocities of the arm
- Action space: consists of 4 continuous variables between -1 and 1
- Reward: The agent gets a reward of +0.1 each time his hand is the goal location

The environment has 2 versions, one composed of a single agent, and the other of 20 agents. Although numerous attempts were made to solve the environment compose of a single agent, I found it easier and faster to solve the environment composed of 20 agents.

The environment is considered solved when the average score over the 20 agents is higher than 30 on average for 100 episodes.

Implementation Description

Since the action space is continuous, it seems wise to use the Deep Deterministic Policy Gradient method as it is more adequate for a continuous action output.

Deep Deterministic Policy Gradient is composed of an actor that outputs an action using the state as inputs and a critic that outputs an "evaluation of the future expected rewards" using the state and action as input. Both are updated using a separate target that gives the architecture more stability, since the target is updated through soft-updates.

Deep Deterministic Policy Gradient introduces the use of a replay buffer to sample experiences from it randomly to avoid having correlated observations when training, hence invalidating the independent and identically distributed samples hypothesis.

The Deep Deterministic Policy Gradient paper suggests using Batch Normalization technique as well to scale input features, but I managed to obtain good results without the later.

Deep Deterministic Policy Gradient is an off-policy algorithm that explores actions by adding a noise to the policy output. In the algorithm paper Ornstein-Uhlenbeck process was used, however when testing multiple noises a decaying constant noise was sufficient.

Learning Algorithm

The agent is composed of an actor and a critic. At each learning step, the next action is computed by inputting the next state to the target network of the actor. The critic target network takes as input the next action and the next state and outputs the expected future rewards. By summing the current reward and the discounted future rewards, we get the target Q-value. Using the output value by the critic network, we can optimize the critic network loss. The equations representing this process are the following:

$$next\_action = actor\_target\_net(next\_state)$$

$$Q_{target} = rewards(state, action) + \gamma \cdot critic\_target\_net(next\_state, next\_action) \cdot (1 - done)$$

$$Q_{expected} = critic\_net(state, action)$$

$$w_{critic} = \min_{w}(Q_{target} - Q_{expected})^2$$

As for the actor network, the predicted action is computed using the state. The actor network loss is the negative expected future rewards computed by the critic network.

$$predicted\_action = actor\_net(state)$$

$$w_{actor} = \max_{w} critic\_net(state, predicted\_action)$$

The actor target and the critic target networks are updated using a soft-update process. The target networks weights are updated at each learning step using the following formula:

$$w\_ = \tau \cdot w + (1 - \tau) \cdot w\_$$

To comply with the fixed target logic, τ is very small (the value 5.10^{-3} is used in this project).

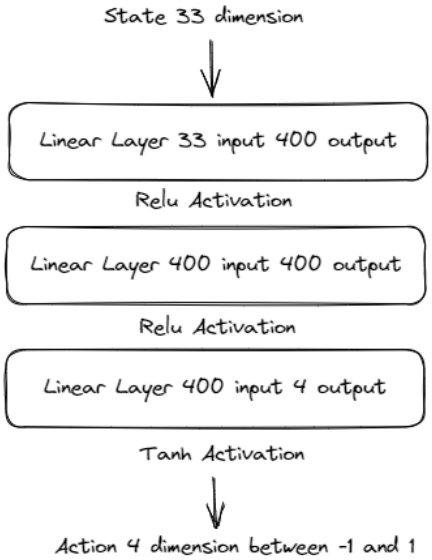
Since, future reward should count less then present rewards, we use a discount factor γ of 0.995.

Last but not least, since this is an off-policy method we choose a decaying noise to add to each action that starts at 0.2 with a decaying factor of 0.995 at each time step.

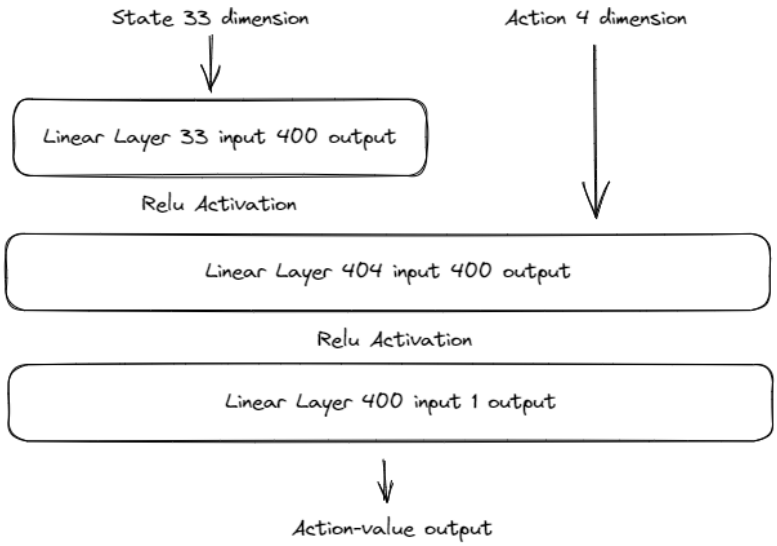
The optimizer of the actor and critic networks is Adam with a learning of 5.10^{-4} and 10^{-3} respectively.

For training we sample a batch of 128 experiences from a buffer keeping the last 10^6 experience tuples.

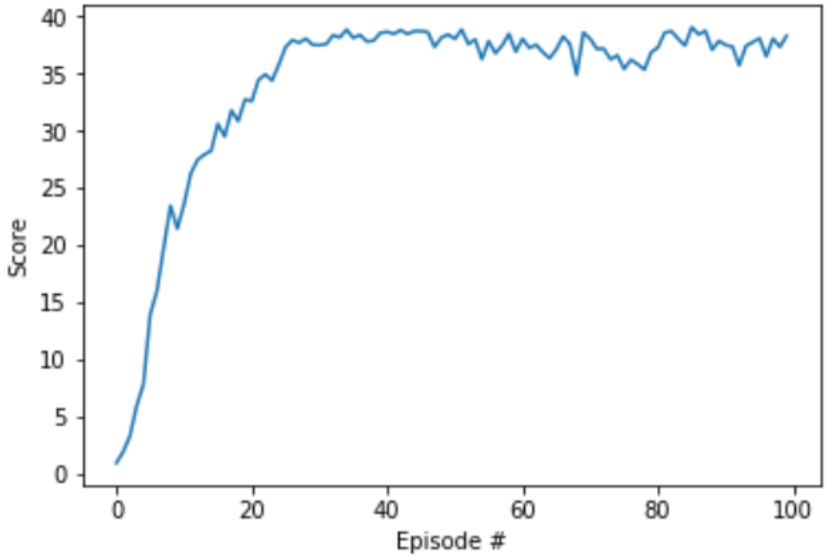
Actor architecture



Critic architecture



Rewards Plot



The score at each time step is the sum of the rewards obtained by the 20 agents divided by the number of agents. The scores over the last 100 episodes were kept in order to compute their average and check whether the environment was solved. As we see in the graph above a score of 30 was reached in less than 20 episodes. As we reached the 100^{th} episode, the average score over the last 100 episodes was well above 30 (close to 34) ! The environment was solved in only 100 episodes.

Ideas for Future Work

- Add a normalization layer for input features as suggested in the Deep Deterministic Policy Gradient paper
- Experiment difference noise processes effects on the learning curve such as Ornstein-Uhlenbeck process or even gaussian process
- Test hard-update instead of soft-update for the target networks by updating the networks only after a certain number of time steps
- Test different Actor and Critic networks architectures with different weights initialization techniques
- Perform a random search for the agent hyperparameters for a better score and convergence