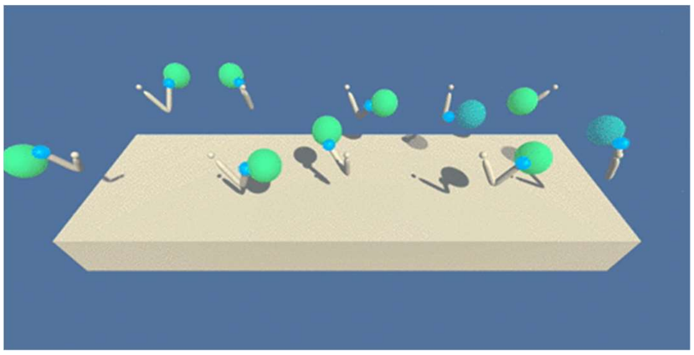
Deep Reinforcement Learning Nanodegree

Project 2 – Continuous Control

1. Introduction



In the Reacher environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

The Unity environment contains 20 identical agents, each with its own copy of the environment. The agents must get an average score of +30 (over 100 consecutive episodes, and over all agents). Specifically,

- After each episode, add the rewards that each agent received (without discounting), to get a score for each agent. This yields 20 (potentially different) scores. Then take the average of these 20 scores.
- This yields an **average score** for each episode (where the average is over all 20 agents).

2. Deep Deterministic Policy Gradient Algorithm

A multi-agent Deep Deterministic Policy Gradient (DDPG) algorithm is implemented to solve the Reacher environment.

The DDPG algorithm is "*a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces*" (Lillicrap et al., 2016).

2.1. Actor-Critic Method

Actor-Critic methods leverage the strengths of both policy-based and value-based methods:

- Using a policy-based method, an agent (actor) learns how to act by directly estimating the optimal policy and maximizing rewards through gradient ascent.
- Using a value-based method, an agent (critic) learns how to estimate the value (i.e. future cumulative reward) of different state-action pairs

Actor-Critic agents combine policy-based and value-based methods to accelerate the learning process. Action-Critic agents are more stable than value-based agents, while requiring fewer training samples than policy-based agents.

In DDPG, the actor agent is trained using the Deterministic Policy Gradient algorithm and produces a deterministic policy. The critic agent learns to evaluate the deterministic policy and is updated using TD-error.

2.2. Network Architecture

The actor network, shown in Figure 2-1, comprises of three layers:

- An input layer with 33 neurons corresponding to 33 possible observation states
- Two hidden fully-connected layers
  - Layer 1 with 400 neurons activated by a `ReLU` function
  - Layer 2 with 300 neurons activated by a `ReLU` function
- An output fully-connected layer with 4 neurons activated by a `tanh` function, corresponding to action vector with four numbers indicating the torque applicable to two joints (shoulder and elbow), where each number has a value between -1 and 1

The critic network, shown in Figure 2-2, comprises of three layers:

- An input layer with 33 neurons corresponding to 33 possible observation states
- Two hidden fully-connected layers
  - Layer 1 with 400 neurons activated by a `ReLU` function
  - Layer 2 with 300 neurons activated by a `ReLU` function where the input is the concatenation of layer 1 output and action vector
- An output fully-connected layer with single neuron, activated by a `ReLU` function, corresponding to a scalar Q value

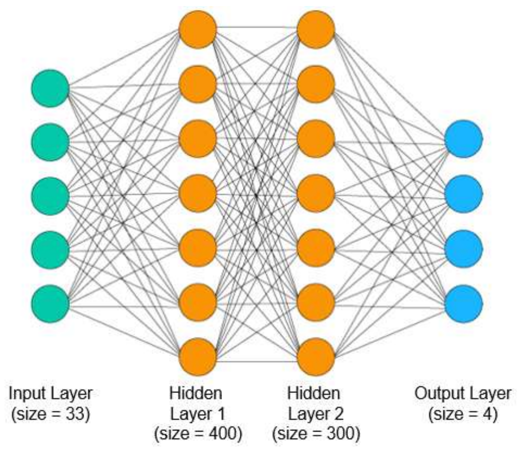


Figure 2-1 Actor Network

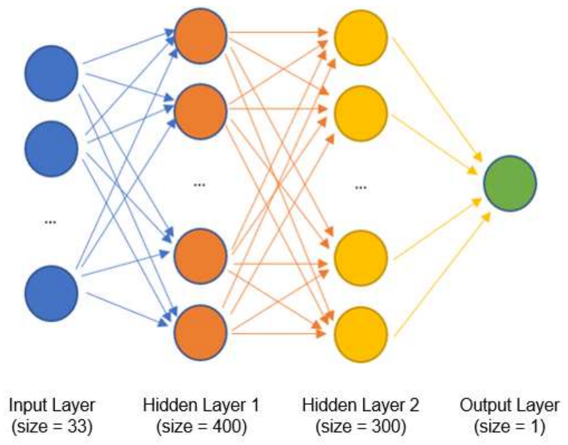


Figure 2-2 Critic Network

Noise is added using an Ornstein-Uhlenbeck process to the action values at each time step, and it decays over time. That is, more noise is introduced earlier in the training process (i.e. higher exploration) and decreases over time as the agent gains more experience (i.e. higher exploitation).

To stabilize the training, three techniques are used:

a. **Experience Replay**
   During training, the sequence of observations can be highly correlated. A (circular) replay buffer is used to store each experience tuple (S, A, R, S') as the agent interacts with the environment. Using experience replay to sample a small batch of tuples from the replay buffer randomly out-of-order breaks the correlation between consecutive experience tuples. This allows the agent to learn more from individual tuples multiple times, recall rare occurrences, make better use of experiences and stabilize the action space - prevent action values from oscillating or diverging catastrophically.

b. **Fixed Target**
   Introduced for DQN, the parameters of the network is shifted based on a constantly moving target, and this can potentially lead to harmful correlations. Two separate networks with identical architectures are used, one with fixed targets and another learns the Q value. The target network weights are updated less often (or more slowly) than the online network. This decouples the target from the parameters, making the learning algorithm more stable, and less likely to diverge or fall into oscillations. Since there are two neural networks for the actor and critic, there are two target neural networks as well, one for actor and one for critic.

c. **Soft Target Updates**
   The target actor and critic networks are updated such that at each update step, 0.01% of the local network weights are mixed with the target network weights. That is, 99.99% of the target network weights are retained and 0.01% of the local network weights are added.

2.3. Hyper-parameters

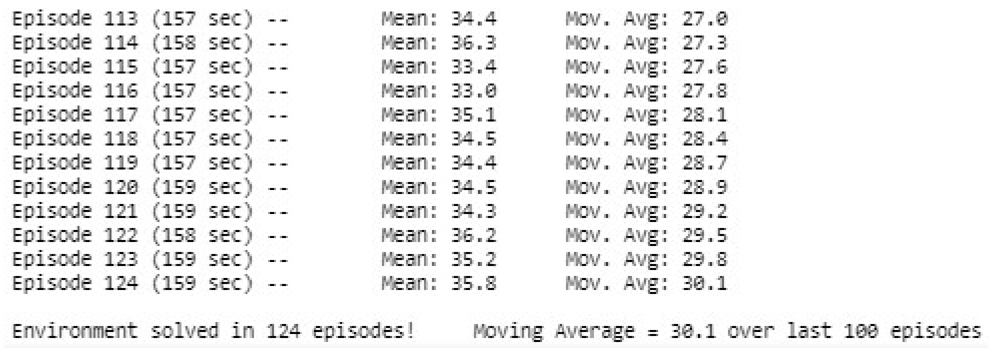
The hyper-parameters used in the DDPG algorithm are shown in Table 1.

Hyper-parameter	Name in code	Value
Discount factor, γ	GAMMA	0.99
Replay buffer size	BUFFER_SIZE	10^5
Minibatch size	BATCH_SIZE	256
Soft update parameter, τ	TAU	10^{-3}
Learning rate of actor	LR_ACTOR	10^{-3}
Learning rate of critic	LR_CRITIC	10^{-3}
Target update rate	LEARN_INTERVAL	20
Initial epsilon, ε_0	EPSILON	1.0
Epsilon decay, δ	EPSILON_DECAY	10^{-6}

Table 1 Hyper-parameters

3. Results

The model training completes once the score – averaged over the past 100 episodes – exceeds 30.0. The model training completes in 124 episodes as shown in Figure 3-1.



```
Episode 113 (157 sec) --       Mean: 34.4      Mov. Avg: 27.0
Episode 114 (158 sec) --       Mean: 36.3      Mov. Avg: 27.3
Episode 115 (157 sec) --       Mean: 33.4      Mov. Avg: 27.6
Episode 116 (157 sec) --       Mean: 33.0      Mov. Avg: 27.8
Episode 117 (157 sec) --       Mean: 35.1      Mov. Avg: 28.1
Episode 118 (157 sec) --       Mean: 34.5      Mov. Avg: 28.4
Episode 119 (157 sec) --       Mean: 34.4      Mov. Avg: 28.7
Episode 120 (159 sec) --       Mean: 34.5      Mov. Avg: 28.9
Episode 121 (159 sec) --       Mean: 34.3      Mov. Avg: 29.2
Episode 122 (158 sec) --       Mean: 36.2      Mov. Avg: 29.5
Episode 123 (159 sec) --       Mean: 35.2      Mov. Avg: 29.8
Episode 124 (159 sec) --       Mean: 35.8      Mov. Avg: 30.1

Environment solved in 124 episodes!     Moving Average = 30.1 over last 100 episodes
```

Figure 3-1 Training Average Score Over 100 Episodes

Figure 3-2 depicts the model scores by episode.

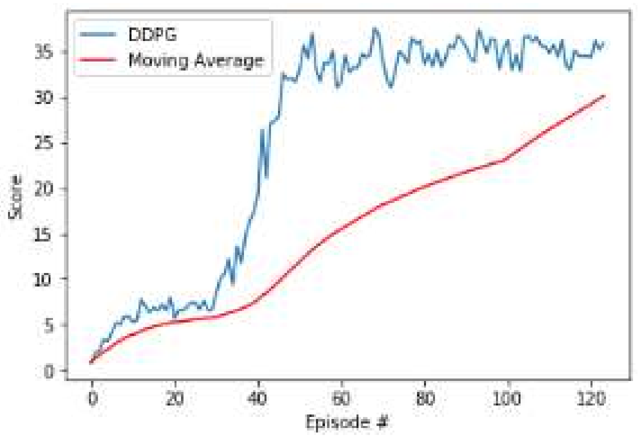


Figure 3-2 Scores by Episode

4. Potential Improvements

- Hyper-parameter tuning (e.g. number of layers and neurons, learning rates, batch and buffer sizes, noise levels)
- Investigate prioritized experienced replay
- Investigate alternate algorithms: PPO, A3C, D4PG

5. Reference

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. Continuous Control With Deep Reinforcement Learning, ICLR 2016. [Online: https://arxiv.org/pdf/1509.02971.pdf].