Continuous Control with Deep Reinforcement Learning

Atul Balaji - EE16B002

March 4, 2019

1 Introduction

The principal aim of this paper is to build on the success of the Deep Q-Network (DQN) and implement a deep reinforcement learning approach for a continuous action space. The DQN works well in discrete and low-dimensional action spaces (as in Atari Games). However, for physical control tasks such as pole balancing, the action space is high dimensional and cannot be solved by discretization of actions, since the number of actions blows up exponentially with the number of degrees of freedom, rendering the DQN alone ineffective due to the curse of dimensionality. This paper uses **Deep Deterministic Policy Gradients** (DDPG), a model-free, off-policy method to learn successful policies using low-dimensional observations like coordinates, angles, pixels, etc.

2 Theory

According to the Bellman Equation:

$$Q^{\pi}(s_t, a_t) = E_{r_t, s_{t+1}}\left[r(s_t, a_t) + E_{a_{t+1}}[Q^{\pi}(s_{t+1}, a_{t+1})]\right] \tag{1}$$

By making our target policy deterministic, we can avoid the expectation over the action space. The policy can be described as a function $\mu : \text{S} \leftarrow \text{A}$. The above equation now becomes:

$$Q^{\mu}(s_t, a_t) = E_{r_t, s_{t+1}}\left[r(s_t, a_t) + Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right] \tag{2}$$

Now, the expectation depends only on the environment. This allows us to learn Q^{μ} off-policy, using transitions generated from a different policy. In Q-learning, we use the greedy policy $\mu(s) = \underset{a}{argmax} Q(s, a)$.

3 Algorithm

In the continuous action space, we cannot find the greedy policy required in Q-learning, since we need to optimize a_t at every time step. So, we use an **actor-critic** approach based on Deterministic Policy Gradients.
The actor function is the deterministic policy $\mu(s|\theta^{\mu})$. The critic $Q(s, a)$ is learned using the Bellman Equation.
This paper makes modifications to DPG to enable it to use neural network function approximators, similar to DQN in order to learn in large state and action spaces online.
As in the case of DQN, a replay buffer is used to store (s_t, a_t, r_t, s_{t+1}). At each time step, both the actor and critic are updated by sampling a minibatch from the buffer.

3.1 Soft Updates

Implementing Q-learning with neural networks is generally unstable and prone to divergence because the network $Q(s, a|\theta^{Q})$ which is being updated, is also being bootstrapped to calculate the target.
The above problem is solved in the DQN by cloning Q to the target network every C steps. In this paper, a similar approach is used, called **soft target updates**.

- We make copies of the actor and critic networks: $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$, which are used as the target networks.
- We perform slow updates to these network weights as:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta',\ \tau << 1 \tag{3}$$

  This ensures that the target values change slowly, thereby increasing stability.

3.2 Batch Normalization

In different environments, the scale of various observations differ. This makes it difficult to have a common set of hyper-parameters. For this purpose, batch normalization is used. It normalizes the mean and variance of each dimension across the samples in a minibatch, to have unit mean and variance.