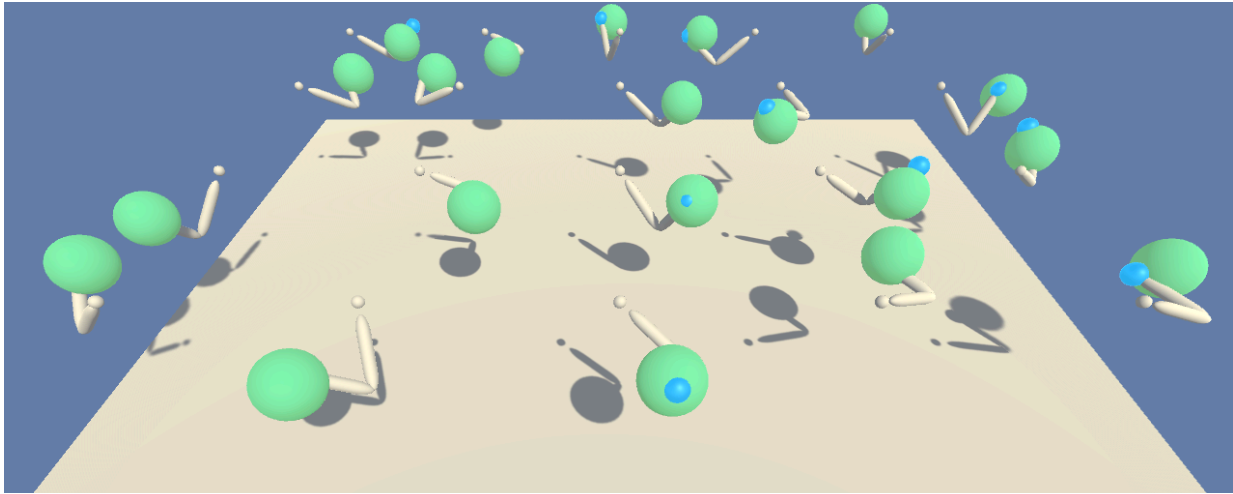


The Reacher – Continues control

Using Deep reinforcement learning for Continues action space control

Mohamed Dawod

dawod.moh@gmail.com



Context:

In this project we want to train a double-jointed arm to move to a specific target location. We give a reward of +0.1 for each step the agent's hand is in the goal location. The goal of the agent is to maintain its position at the target location as many time steps as possible.

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

This problem is an episodic task and to solve this environment we use 20 parallel agents which must get an average rewards of +30 over 100 consecutive episodes.

The problem:

Deep Q Network (DQN) can only handle discrete and low dimensional action spaces. However, our task needs a continues (real value) and high dimensional action spaces to control the arm movement. Therefore, DQN can't be used to solve this problem as DQN by nature cannot be straight forwardly applied to continues domain since it relies on finding the action which can maximize the action-value function.

One solution to adapt DQN to a continuous domain is by simply discretize the action space. However, this has many limitations as the number of actions increases exponentially with the number of degrees of freedom. For instance, a 7 degree of freedom system (a human arm) with coarsest discretization $(-k, 0, L)$ for each joint leads to an action space with dimensionality $3^7 = 2187$. Therefore, in this project we will implement the model-free, off-policy actor-critic algorithm Deep Deterministic Policy Gradient (DDPG). DDPG can learn policies using low-dimensional observations (e.g. Cartesian coordinates or joint angles) or learning from pixels.

DDPG is using an actor-critic (two neural network) approach based on the DPG algorithm (Silver et al., 2014). The first network (The Actor) which basically tries to approximate the optimal policy. The second network (The Critic) tries to estimate the reward from following that approximate optimal policy.

The first network uses the DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping the states to a specific action. The critic $Q(s,a)$ is learned by using the Bellman equation as in Q-learning. The actor is updated by following the chain rule to expected return from the state distribution J with respect to the actor parameters:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta} [\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}] \\ &= \mathbb{E}_{s_t \sim \rho^\beta} [\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t}]\end{aligned}$$

Fig 1. Silver et al. (2014) proved that this is the policy gradient, the gradient of the policy's performance.

This process becomes a "(Actor)try the policy" then "(critic)evaluate the policy action value" then "improve the policy" loop. The improving step comes by the actor and critic network updating through their loss function. In addition to that there are several elements that are running behind the scene. This method uses Ornstein Uhlenbeck noise, a replay buffer, target networks, and a soft updating.

The process run as follows:

At the start the actor and the critic network are randomly initialized. During an episode time's step, the actor is given the current state and returns an action value, which is then added to the Ornstein Uhlenbeck noise. This action is given to the environment and returns the reward and the new state. These values are then stored in the Replay buffer in form of a tuple (State, action, reward, next state). Once the replay buffer has enough transitions, a random sample of these experiences is taken and used to help the critic network. The critic network tries to predict the next reward given the new next state, which can be thought of as the expected Q-value. Then both networks are updated, first the critic then the actor.

Key solution

In addition to the importance to the hyperparameters tweaking the main key solution to this project relies on using the correct application of Ornstein Uhlenbeck Noise method. Specifically on how we apply the noise sample.

Deep Neural Network Model

The algorithm uses two deep neural networks (actor-critic) with the following structure:

- Actor
 - Hidden: (State_size, 400) - ReLU
 - Hidden: (400, 300) - ReLU
 - Output: (300, 4) - TanH
- Critic
 - Hidden: (State_size, 400) - ReLU
 - Hidden: (400 + action_size, 300) - ReLU
 - Output: (300, 1) - Linear

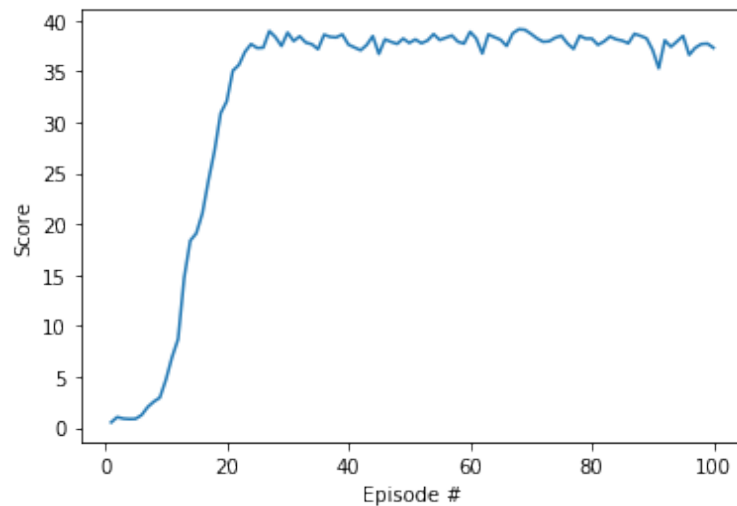
Hyperparameters

<code>BUFFER_SIZE = int(1e6)</code>	<code># replay buffer size</code>
<code>BATCH_SIZE = 512</code>	<code># minibatch size</code>
<code>GAMMA = 0.99</code>	<code># discount factor</code>
<code>TAU = 1e-3</code>	<code># for soft update of target parameters</code>
<code>LR_ACTOR = 1e-4</code>	<code># learning rate of the actor</code>
<code>LR_CRITIC = 3e-4</code>	<code># learning rate of the critic</code>
<code>WEIGHT_DECAY = 0</code>	<code># L2 weight decay</code>
<code>ACTOR_HL_SIZE= [400, 300]</code>	<code>#Actor Hidden layers</code>
<code>CRITIC_HL_SIZE= [400, 300]</code>	<code>#Critic Hidden layers</code>

Results

Episode 10	Current Score: 4.70	Average Score: 1.75
Episode 20	Current Score: 32.06	Average Score: 20.33
Episode 30	Current Score: 38.81	Average Score: 37.35
Episode 40	Current Score: 37.62	Average Score: 38.06
Episode 50	Current Score: 37.79	Average Score: 37.69
Episode 60	Current Score: 38.88	Average Score: 38.20
Episode 70	Current Score: 38.68	Average Score: 38.32
Episode 80	Current Score: 38.22	Average Score: 38.08
Episode 90	Current Score: 37.09	Average Score: 38.04
Episode 100	Current Score: 37.31	Average Score: 37.38

Environment solved in 100 episodes!	Average Score: 37.38
-------------------------------------	----------------------



Future Work

Idea 1: DDPG is still an active area of research, therefore I'm willing to try to improve the algorithm by implementing the following concepts:

- 1- [Prioritized Experience Replay](#)
- 2- [Exploration with Parameter Noise](#)

Idea 2: Exploring different reinforcement learning algorithms like:

- 1- [Asynchronous Actor-Critic Agents \(A3C\)](#)
- 2- [Trust Region Policy Optimization \(TRPO\) and Proximal Policy Optimization \(PPO\)](#)

References

Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In ICML, 2014.