

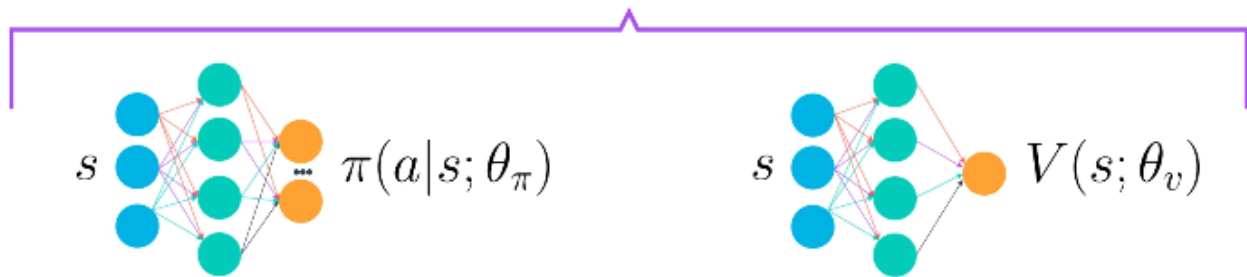
Udacity Deep Reinforcement Learning Nanodegree

Project 2: Continuous Control

Introduction

This report details the implementation of a reinforcement learning algorithm to solve the continuous control problem in Unity's Reacher environment. The solution presented here uses the Deep Deterministic Policy Gradient (DDPG) algorithm, which is a form of model-free actor-critic reinforcement learning that operates over continuous action spaces.

Actor-Critic



DDPG is a method that is similar to Deep Q Learning, and uses two neural networks to complete a task – the Actor to determine the best action from the policy deterministically,

$$a^* = \operatorname{argmax}_a Q(s, a),$$

and the Critic to determine the action value function,

$$Q(s, \mu(s; \theta_\mu); \theta_Q).$$

The algorithm runs as follows (from Lillicrap et al. 2016):

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process \mathcal{N} for action exploration
 Receive initial observation state s_1
 for t = 1, T **do**
 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

end for
end for

Implementation Details

As in the original Lillicrap implementation, we also use a replay buffer containing tuples of previous transitions, that we periodically revisit to improve sample efficiency. In addition, a soft update is used in order to slowly blend a target and the regular network weights. In this solution, both the actor and the critic are composed of four-layer neural networks, with a fully connected layer, followed by a batch normalization layer, followed by two additional fully-connected layers.

In order to encourage the agent to explore, we impose noise on the actions that are chosen by the actor, so that the actions that are actually taken have an additional term dictated by an Ornstein-Uhlenbeck process. This is a stationary Gauss-Markov process, and is a form of mean-reverting random walk. We also impose a decay factor on the process, to reduce the amount of exploration as time goes on and the agent's policy becomes more feasible. Since the action output can only take values between -1 and 1, the outputs of the actor are clipped to be between -1 and 1.

Results

The DDPG training solved the environment in 354 episodes (with the window of episodes 254-354 averaging more than +30 reward). The plot of the rewards is shown below. In an earlier run, the solution was actually found in approximately 270 episodes, but an error occurred while saving the network weights that forced a reversion to an earlier copy of the actor and critic networks (saved at episode 200). Unfortunately, not all the decayed parameters were returned to their state at episode 200, so the score dropped dramatically upon reloading. Still, as shown in episodes 200-300, starting with network weights that were previously trained somewhat did help the agent get a “warm start.” This segment could be compared to episodes 0-100, which showed significantly less improvement in score, since it started from random network weights.

