Report for Udacity DRLND Project 2

Continuous Control: Reacher

Introduction

In this 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 your 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 task is episodic, and in order to solve the environment, your agent must get an average score of +30 over 100 consecutive episodes.

Deep reinforcement learning Methods

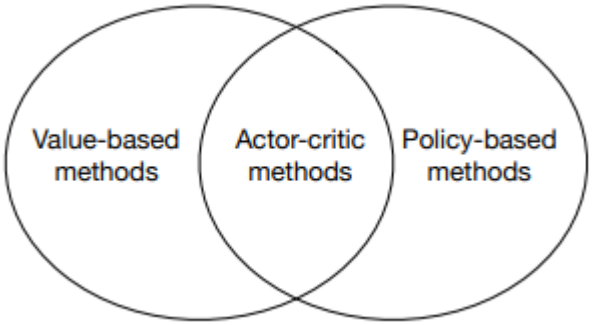
Policy-based & value-based methods

With value-based methods**,** the agent uses its experience with the environment to maintain an estimate of the optimal action-value function. The optimal policy is then obtained from the optimal action-value function estimate. Policy-based methods directly learn the optimal policy, without having to maintain a separate value function estimate.

Actor-critic methods

In actor-critic methods, we are using value-based techniques to further reduce the variance of policy-based methods. Basically actor-critic are a mixed version of the

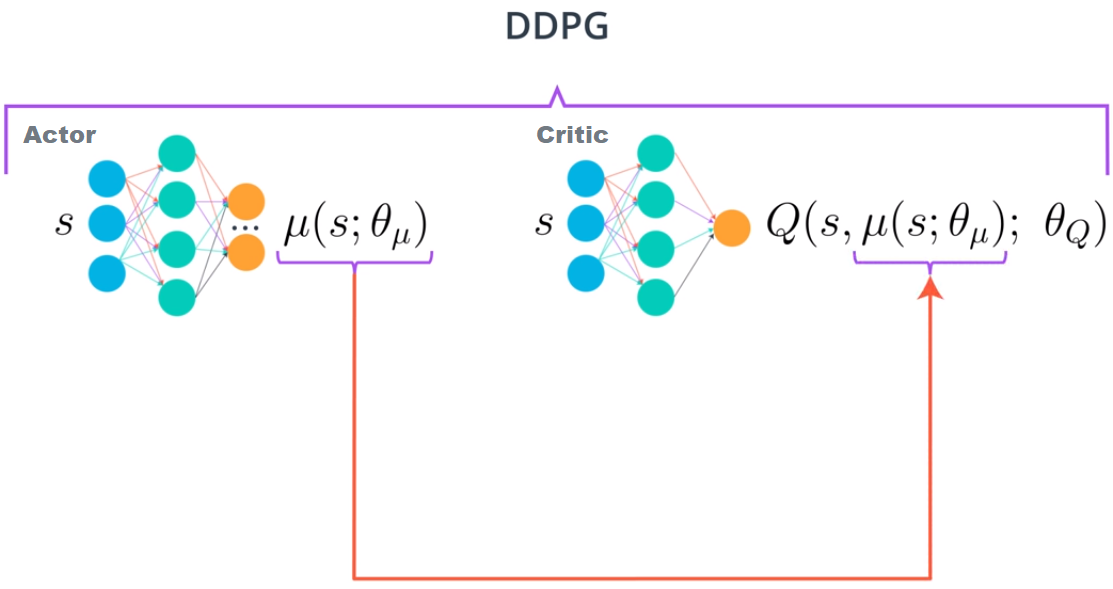
policy-base and value- based methods, in which the actor estimates the policy and the critic estimates the value function.



Deep Deterministic Policy Gradient (DDPG)

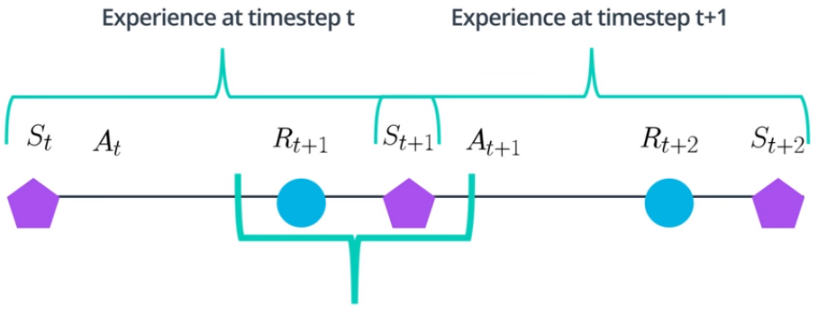
DDPG short for Deep Deterministic Policy Gradient, is a model free off policy actor critic algorithm, combining DPG with DQN. Recall that DQN (Deep Q- Network) stabilizes the learning of the Q-function by using experience replay and a frozen target network. The original DQN works in discrete action space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy.

In DDPG, we use 2 deep neural networks: one is the actor and the other is the critic:

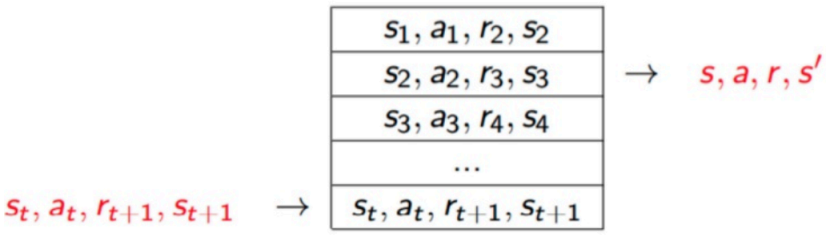


Replay Buffer

When using neural networks, it is usually assumed that samples are independently and identically distributed. In Reinforcement Learning, samples are generated from exploring sequentially in an environment, resulting in the previous assumption that no longer holds. Action A_t is partially responsible for the reward and state at time (t + 1) :



As in DQN, we use a replay buffer to address this issue. Because DDPG is an off-policy algorithm, that is it employs a separate behavior policy independent of the policy being improved upon, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.



Replay Buffer – fixed size

Soft target updates

In DDPG, not only we have a regular network for the actor as well for the critic, we do also have a copy of these regular networks. We have a target actor and target critic, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ respectively that are used for calculating the target values.

We update those target networks using a soft-update strategy. A soft-update strategy consists of slowly blending your regular network weights with your target network weights:

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

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

$\tau < 1$, usually 1% or 0.01.

This means that the target values are constrained to change slowly, greatly improving the stability of learning.

Noise for exploration

A major challenge of learning in continuous action spaces is exploration. An advantage of off-policies algorithms such as DDPG is that we can treat the problem of exploration independently from the learning algorithm. As such, an exploration policy μ' is constructed by adding noise N :

$$\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + N$$

N can be chosen to suit the environment. We use the Ornstein-Uhlenbeck process to model the velocity of a Brownian particle with friction, which results in temporally correlated values centered around 0. This allows exploration efficiency in physical control problems with inertia. Start with Brownian motion (also known as Wiener process): $dW_t = N(0, dt)$ Then add friction: $dx_t = \theta(\mu - x_t)\, d_t + \sigma\, dW_t$

Where θ controls the amount of friction to pull the particle towards the global mean μ. The parameter σ controls the scale of the noise. By discretizing the previous formula, we get:

$$X_{n+1} = X_n + \theta(\mu - X_n)\Delta t + \sigma\Delta W_n$$

DDPG Algorithm

The DDPG algorithm are explained in the paper: CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

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

Because we have two models; the actor and critic that must be trained, it means that we have two set of weights that must be optimized separately. Adam was used for the neural networks with a learning rate of *1e-4* for the actor and critic. For Q, the discount factor of $\gamma = 0.90$. For the soft target updates, the hyperparameter τ was set to *1e-3*.

The neural networks have 2 hidden layers with 350 and 280 units respectively. For the critic Q, the actions were not included the 1st hidden layer of Q. The final layer weights and biases of both the actor and critic were initialized from a uniform distribution *(-3e-3, 3e-3)* to ensure that the initial outputs for the policy and value

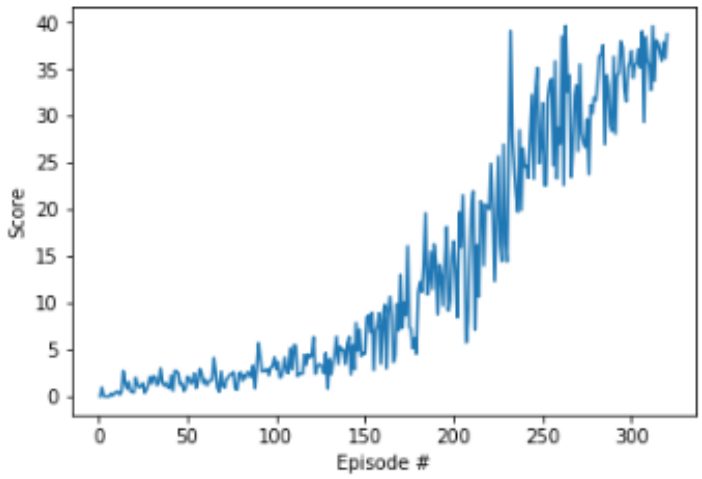
estimates were near zero. The training used mini batches of 500 and the replay buffer was of size 1e6.

For the exploration noise process, the parameters used for the Ornstein-Uhlenbeck process:

$\theta = 0.15$ and $\sigma = 0.18$.

Results

Then agent solved situation (Average Score 30, over last 100 episodes) in 320 episodes.



Environment solved in 320(from 220) episodes! Average Score: 30.16

Future ideas DDPG

The following techniques are worth pursuing for faster convergence or higher peak performance and new experience in Deep Reinforcement Learning:

Hyperparameter search

The learning rates selected may not be optimal and may result in slower convergence.

Architecture search

Batch Normalization in neural network was common solution to increase training speed so that would be probably my next step.

Is it possible that a deeper model or adding / removing units in each of the hidden layers may improve performance? Or combined with exploring and comparing different methods mentioned in course and papers REINFORCE, TNPG, RWR, REPS, TRPO, CEM, CMA-E could also bring better results?

Prioritized experience replay

Prioritized replay of states and actions seems to be helpful to convergence and definitely will improve performance.

Try 20 agents' version

The first version of this environment covered in this report, takes into account only one agent. There is another version which now take over 20 agents in the same environment. This seems to be a great challenge to tackle, leaving place for algorithms like PPO, A3C, and D4PG that use multiple (non-interacting, parallel) copies of the same agent to distribute the task of gathering experience.