PROJECT REPORT

UDACITY

DEEP REINFORCEMENT LEARNING NANODEGREE

Project 2: Continuous Control

AUTHOR:
Thomas Teh

Date: October 29, 2018

1 Deep Deterministic Policy Gradient (DDPG)

I have implemented DDPG, as introduced by Lillicrap et al. (2015), to solve this problem. The pseudo-code for DDPG is given in Algorithm 1.
The action-value function describes the expected return after taking an action a_t in state s_t and thereafter following policy π:

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= \mathbb{E}_{\pi}[R_t|s_t, a_t] \\ &= \mathbb{E}_{\pi}\left[\sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)|s_t, a_t\right] \end{aligned}$$

The action-value function follows the Bellman equation:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}[r(s_t, a_t) + \gamma \mathbb{E}_{\pi}[Q^{\pi}(s_{t+1}, a_{t+1})]]$$

Given that the target policy is deterministic, we can described it as $\mu : \mathcal{S} \rightarrow \mathcal{A}$

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{\pi}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))]$$

In DDPG, we use non-linear function approximator to learn both the $Q(s_t, a_t)$ action-value function and the policy $\mu(s_t)$.
Firstly, the action-value function is learned using Q-learning, whereby $\mu(s_t) = \arg\max_{a \in \mathcal{A}} Q(s_t, a_t)$. Similar to the DQN algorithm introduced by Mnih et al. (2015), the action-value function $Q(s_t, a_t)$ is learned by minimizing the following loss function:

$$L(\theta^Q) = \mathbb{E}_{\pi'}\left[(Q(s_t, a_t) - y_t)^2\right]$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$$

Secondly, the actor network is learned by using the DPG algorithm introduced by Silver et al. (2014). The actor is updated by the following:

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

One final issue on DDPG is exploration. Exploration is introduced to the algorithm by including some noise in the action.

$$a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_{\sqcup}.$$

$\mathcal{N}_t$ is an Ornstein-Uhlenbeck process

$$d\mathcal{N}_t = \theta_{ou}(\mu_{ou} - \mathcal{N}_t)dt + \sigma_{ou} dW_t$$

where θ_{ou} is the mean-reversion rate, μ_{ou} is the long-term mean, $sigma_{ou}$ is the volatility and W_t is a Brownian motion.

Initialization:
Randomly initialize critic network $Q\left(s, a|\theta^Q\right)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$.
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
Selection action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$ according to the current policy and exploration noise.
Execute action a_t and observe reward r_t and observer 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 $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.
Update the critic network by minimizing:

$$L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2.$$

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
end

Algorithm 1: Deep Deterministic Policy Gradient (DDPG)

2 Modification to DDPG

In the project, I specifically tackled the problem with 20 agents. DDPG was originally designed for a single agent. However, since all 20 agents have the same tasks, the tuple (s_t, a_t, r_t, s_{t+1}) resulting from an action a_t taken by the different agents are stored in the experience replay buffer. While this does not constitute a full distributed learning algorithm such as A3C or D4PG, it was sufficient to do so in order to solve the problem.

3 Architectures of the Actor and Critic Network

The actor network, as shown in Figure 1 consists of the following:

- Input layer: The input to the network are the states of the environments observable by the agents.
- Hidden layer: The hidden layer is made up of 256 neuron units with exponential linear units.
- Output layer: The network outputs the action taken by the agent, which consists of 4 numbers, corresponding to the torque applicable on to two joints.

Figure 1: Architecture of the actor network.

The critic network, as shown in Figure 2 consists of the following:

- Input layer: The input to the network are the states of the environments observable by the agents.
- Hidden layer: Hidden layer 1 takes in the states observable by the agent as input. Hidden layer 2 takes both the output from hidden layer 1 and the action as inputs. Hidden layer 3 takes the output of hidden layer 2 as input.
- Output layer: The network outputs the value function of the state-action input.

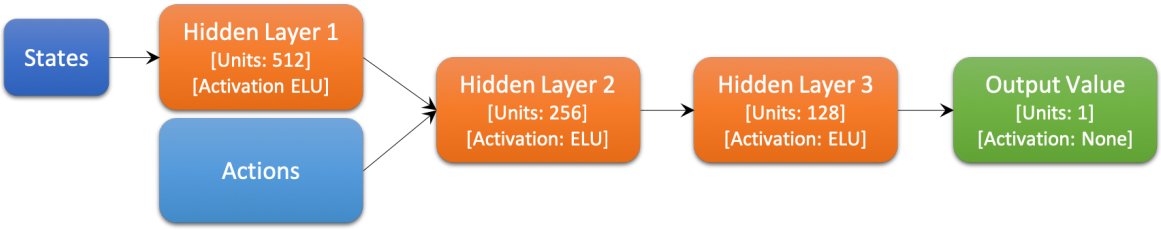

Figure 2: Architecture of the critic network.

While in (Lillicrap et al., 2015), the authors mentioned the application of batch normalization with ReLU activation functions for their actor-critic networks. I found that the training process is more stable by using exponential linear units (ELUs) introduced by Clevert et al. (2015). The ELUs retain most of the characteristics of ReLUs but they have an implicit regularization effect which made training much more stable.

4 Hyperparameters

The hyperparameters used in training the DDPG algorithm are listed below:

Hyperparameter	**Value**
Actor learning rate, α^{μ}	0.0010
Critic learning rate, α^{Q}	0.0010
Discount factor, γ	0.9950
Buffer size	3,000,000
Minibatch size	128
Soft update weights, τ	0.0010
Update frequency	Every 20 time steps
Number of updates	10 updates
Long term mean for noise, μ_{ou}	0.00
Mean reversion rate for noise, θ_{ou}	0.15
Volatility for noise, σ_{ou}	0.20

5 Results

The average rewards during the training process is shown in Figure **??**. The DDPG algorithm achieves a mean score of 34.50 (exceeding the required 30), across 100 episodes and all 20 agents after 737 episodes.

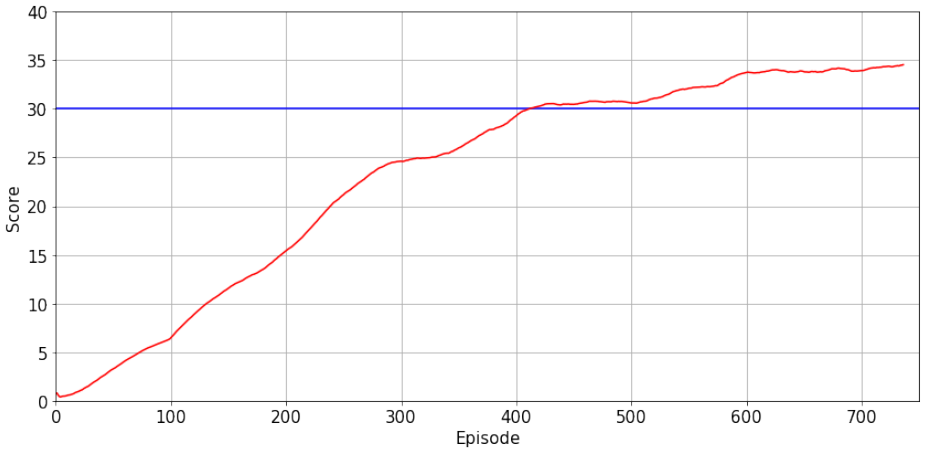

Figure 3: Average reward vs number of episodes

6 Ideas for Future Work

The solution of the problem can potentially improved by applying different algorithms. The applicable of DDPG in this problem is not parallelized across the 20 agents. Essentially, each agent would contribute to the memory for experience replay and then the parameter optimization update is done based on those replays. For future work, I would explore implementations that can be parallelized such as A3C and D4PG in order to speed up the training process. Also, for comparison purposes, I would implement PPO as well.

References

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*. pages 5

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. pages 2, 5

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529. pages 2

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *ICML*. pages 2