DRLND-P3-Report

Ali Alizadeh

1 Introduction

In this report, I summarize my final modeling decisions as to succeed in solving the Tennis environment. My approach in this project is to use the Deep Deterministic Policy Gradients (DDPG) algorithm [1] modified to be applicable for multi-agent setup.

2 Description of Learning Algorithm

DDPG algorithm uses actor-critic framework with replay buffer as described in the following pseudo-code [1].

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

My implementation follows the standard DDPG algorithm described above with some modifications to make it multi-agent and suitable for competing environment.

2.1 Multi-agent DDPG

Multi-agent DDPG algorithm differs from the standard DDPG algorithm described above in a way that the experience replay buffer is fed with multiple agents (suitable for distributed traiing) while the agents interact. The multi-agent framework in this project uses 2 interacting agents with different state vectors from the environment.

Experience replay buffer in multi-agent DDPG setup for distributed training is fed with interactions of two interacting agents.

```
# Save experience / reward
for i in range(self.num_agents):
    self.memory.add(state[i,:], action[i,:], reward[i], next_state[i,:], done[i])
```

3 Hyper-parameters optimization

The following hyper-parameters are being used in my implementation:

- Batch size of 128
- Experience replay buffer size of 100000 randomly samples of actions
- Discount factor of 0.99
- Actor's learning rate of 0.001
- Critic's learning rate of 0.001
- Soft update parameter of target parameters of 0.001
- Weight decay of 0
- Action noise with Ornstein-Uhlenbeck process with mean (μ) of 0 and mean reversion rate (θ) of 0.15 and variance (σ^2) of 0.1

3.1 Model Architecture

The actor-critic framework is used in this implementation, where:

- Actor network is a mapping from state to action values. The actor network takes a state vector of 24 elements as input and returns 2 elements as an action vector. A neural network model with two hidden layers of128 neurons and RELU activation functions is mapping the input to the 2-dimensional output layer with tanh activation function (bounding the range of actions between -1 and $+1$). Batch-normalization is also applied ot the input and hidden layers.

- Critic network maps the state and action vectors as inputs (state-action pairs) to a scalar Q-value as output. A DNN with a 24-dimensional input layer is connected to the first hidden layer with 128 neurons followed by a batch-normalization layer. The output of batch-normalization layer is fed to a hidden layer with 128 neurons with RELU activation functions for both hidden layers. The output is achieved through a single neuron and linear activation to give single real number.

4 Results and Discussion

The training log of the multi-agent DDPG algorithm applied to Tennis algorithm is shown in Fig. 1. As shown in Fig. 1, the orange line shows the learning performance for the average score of 100 consecutive episodes which shows that it has crossed the success threshold determined by red line in the plot.

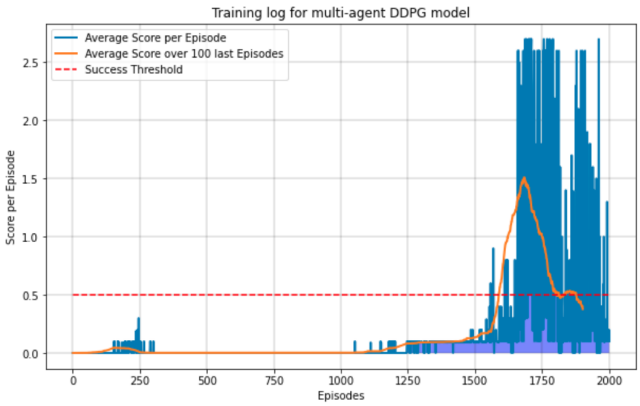

Figure 1: Training log of the agent's performance in Tennis environment

5 Future Work

I can list future works as follows:

- Automated hyper-parameters optimzation.
- If using DDPG, utilization of prioritized experience replay.
- According to literature, utilization of more PPO [1], A3C [2], and D4PG [3].
- Share information between competing agents for actor network parameters.

[1] https://arxiv.org/pdf/1707.06347.pdf
[2] https://arxiv.org/pdf/1602.01783.pdf
[3] https://openreview.net/pdf?id=SyZipzbCb

References

[1] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.