Multi-agent DDPG

Alexandre Popoff

May 5, 2019

Multi-agent deep deterministic policy gradient (MADDPG) agents are implemented to solve Udacity's tennis environment. In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of 0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. At each episode, each agent has its own score, then the maximum score is the final score of the episode. The environment is considered solved when the average over the last 100 episodes is at least 0.5. The environment is solved in 639 episodes.

1 Algorithm

The MADDPG algorithm is given in [1] and it is reproduced below. In the tennis environment, each agent has its own Q function and deterministic policy $\mu : o \mapsto a$ (for an observation o, $\mu(o) = a$ is the action taken by the agent). In order to promote exploration, for `start_steps`=1000 episodes, a random action is chosen uniformly from the action space. It introduces a new hyperparameter `start_steps` but without it, the algorithm hardly converges. After this initial exploration, a random noise with decreasing standard deviation is added to $\mu(o)$ so that the action taken by the agent i is $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$. The Q functions are centralized (i.e. they take a vector x and all the actions a_i as input; in this report, x is (s_1, s_2) with s_i the state of the ith agent). Each transition (x, a, r, x') is stored in each agent's memory. *In fine*, it is the same memory but it is kept general for further tests on prioritized replay buffer. But so far, it does not seem to lead to better results. In the original DQN algorithm, a prioritized replay buffer samples a transition with a probability proportional to the TD error and the action taken is the actual action that maximizes Q. In DDPG, the action is estimated by a neural network, not the true optimal action. This could be the reason why a prioritized replay buffer does not seem to give faster convergence in this case.

In this environment, the two agents must collaborate in order to reach the maximum score so that they basically want to learn the same Q function. In practice, implementing directly the same Q function for both agents does not seem to give faster convergence. Instead, each agent i learns its own Q_i but a soft-update on the neural network weights between Q_1 and Q_2 is implemented. The idea is that each agent also learns from the other agent since they want to learn the same centralized Q function. In practice, it seems to accelerate convergence.

Algorithm 1 Multi-agent deep deterministic policy gradient algorithm

for *episode*$= 1$ *to M* **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state x; **for** *episode*$= 1$ *to max-episode-length* **do**
    for each agent i, select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state x'
    Store (x, a, r, x') in replay buffer $\mathcal{D}$
    $x \leftarrow x'$
    for *agent* $i = 1$ *to* N **do**
      Sample a random minibatch of S samples (x^j, a^j, r^j, x'^j) from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, \ldots, a'_N)|_{a'_k = \mu'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\mu}(x^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j)\nabla_{a_i}Q_i^{\mu}(x^j, a_1^j, \ldots, a_i, \ldots, a_N^j)|_{a_i=\mu_i(o_i^j)}$$

    end
    Update target network parameters for each agent i:

$$\theta'_i \rightarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end
end

2 Neural network

The deterministic policy μ is a neural network with a state as input, two fully connected hidden layers with $(64, 64)$ hidden units (ReLU activation function). The output is the action taken by the agent (tanh activation function on the last layer). In this environment, ReLU activations seem to give more stable results compared to tanh activation functions. See below the definition of μ where $fc1 = fc2 = 64$.

```
self.mu =  nn.Sequential(
                              nn.Linear(state_dim, fc1),
                              nn.ReLU(),
                              nn.Linear(fc1, fc2),
                              nn.ReLU(),
                              nn.Linear(fc2, action_dim),
                              nn.Tanh()
                              )
```

The Q function takes a state as input, followed by a fully connected layer with 64 units and a ReLU activation function which gives the activation a_1. In the second layer, the activation a_1 and the action is concatenated to form the input of the second fully connected layer with 128 units and a

ReLU activation function. The last layer is a fully connected layer with 1 unit as ouput. See below the forward function for Q.

```
def forward(self, state, action):
    x = self.fc1(state)
    x = F.relu(x)
    x = torch.cat([x, action], dim=1)
    x = self.fc2(x)
    x = F.relu(x)
    x = self.fc3(x)

    return x
```

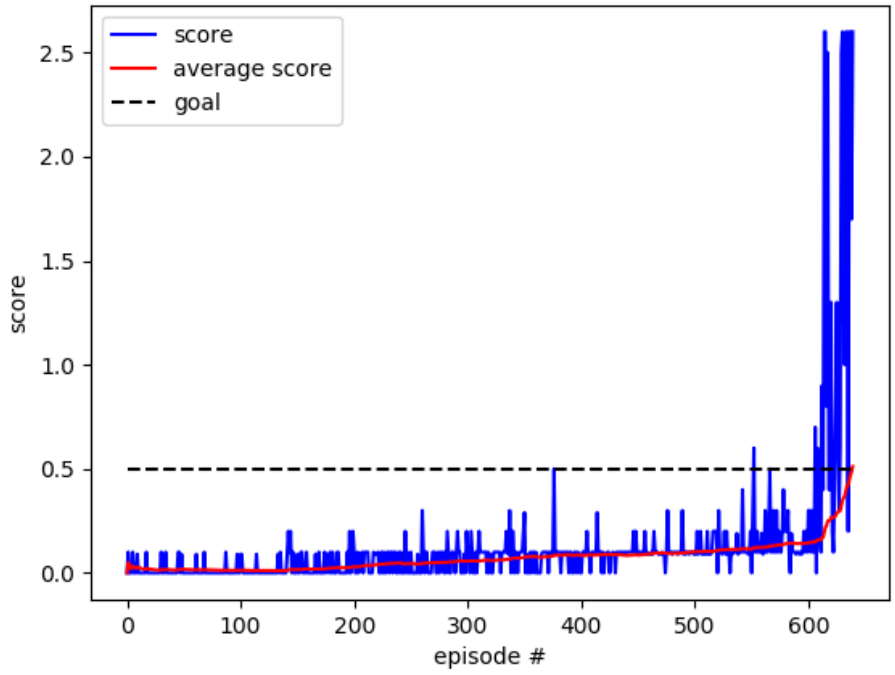
3 Hyperparameters

Below the list of hyperparameters.

Hyperparameter	Value
M (maximum budget)	1000
`start_steps` (initial period for uniform exploration)	1000
Buffer size	10^6
Minibatch size	128
Adam lr for both agents, for both Q and μ	10^{-3}
τ (soft-update constant between $Q_{\theta'}$ and Q_θ, $\mu_{\theta'}$ and μ_θ)	0.01
γ (discount)	0.99
λ (soft-update constant between Q_1 and Q_2, μ_1 and μ_2)	0.01
Gradient clip (for μ only)	0.5

4 Plot of rewards

The environment is generally solved in $600-700$ episodes. Below, the agents solved the environment in 639 episodes.



5 Future work

As mentioned in [1], we should ask the question of the scalability of the algorithm. When the number of agent is small, it is always possible to use a centralized Q function. In a more general setting, the centralized learning could take into account only a certain neighborhood of a given agent.

Further study could focus on reducing variance. As mentioned in this report (from limited samples), a proportional prioritized replay buffer does not seem to improve performance. Instead, in [1], the author suggest to train agents with an ensemble of policies as described in the paper.

The present work could be extended to the Udacity's soccer environment where four agents play soccer. Each team should have the same centralized learning and decentralized action as implemented in this report.

References

[1] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.