Report

Udacity project 2

Continuous control with polciy gradient method DDPG

Environment

We want to let the arm controlled to be at the goal position.

Observations include (position, velocity, rotation, angular velocity...)

```
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :
    goal_speed -> 1.0
    goal_size -> 5.0
Unity brain name: ReacherBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 33
        Number of stacked Vector Observation: 1
        Vector Action space type: continuous
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
```

Reward is set to be +0.1 for each step the arm is in position.

Total timestep for each episode is 1000. In order to solve the environment, the agent must get an average score of +30 over 100 consecutive episodes.

Architecture used

For the agent there are actor and critic parts.

```
class Actor(nn.Module):
    """Actor (Policy) Model."""

    def __init__(self, state_size, action_size, seed, fc1_units=500, fc2_units=400, fc3_units=200):
        super(Actor, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, fc3_units)
        self.fc4 = nn.Linear(fc3_units, action_size)
```

```
        self.reset_parameters()

    def reset_parameters(self):
        self.fc1.weight.data.uniform_(*hidden_init(self.fc1))
        self.fc2.weight.data.uniform_(*hidden_init(self.fc2))
        self.fc3.weight.data.uniform_(*hidden_init(self.fc3))
        self.fc4.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, state):
        """Build an actor (policy) network that maps states -> actions."""
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        return F.tanh(self.fc4(x))
```

```
class Critic(nn.Module):
    """Critic (Value) Model."""

    def __init__(self, state_size, action_size, seed, fcs1_units=500, fc2_units=300):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
            fcs1_units (int): Number of nodes in the first hidden layer
            fc2_units (int): Number of nodes in the second hidden layer
        """
        super(Critic, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fcs1 = nn.Linear(state_size, fcs1_units)
        self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
        self.fc3 = nn.Linear(fc2_units, 1)
        self.reset_parameters()

    def reset_parameters(self):
        self.fcs1.weight.data.uniform_(*hidden_init(self.fcs1))
        self.fc2.weight.data.uniform_(*hidden_init(self.fc2))
        self.fc3.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, state, action):
        """Build a critic (value) network that maps (state, action) pairs -> Q-values."""
        xs = F.relu(self.fcs1(state))
        x = torch.cat((xs, action), dim=1)
        x = F.relu(self.fc2(x))
        return self.fc3(x)lass Critic(nn.Module):
"""Critic (Value) Model."""
```

Hyperparameters

```
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 128
```

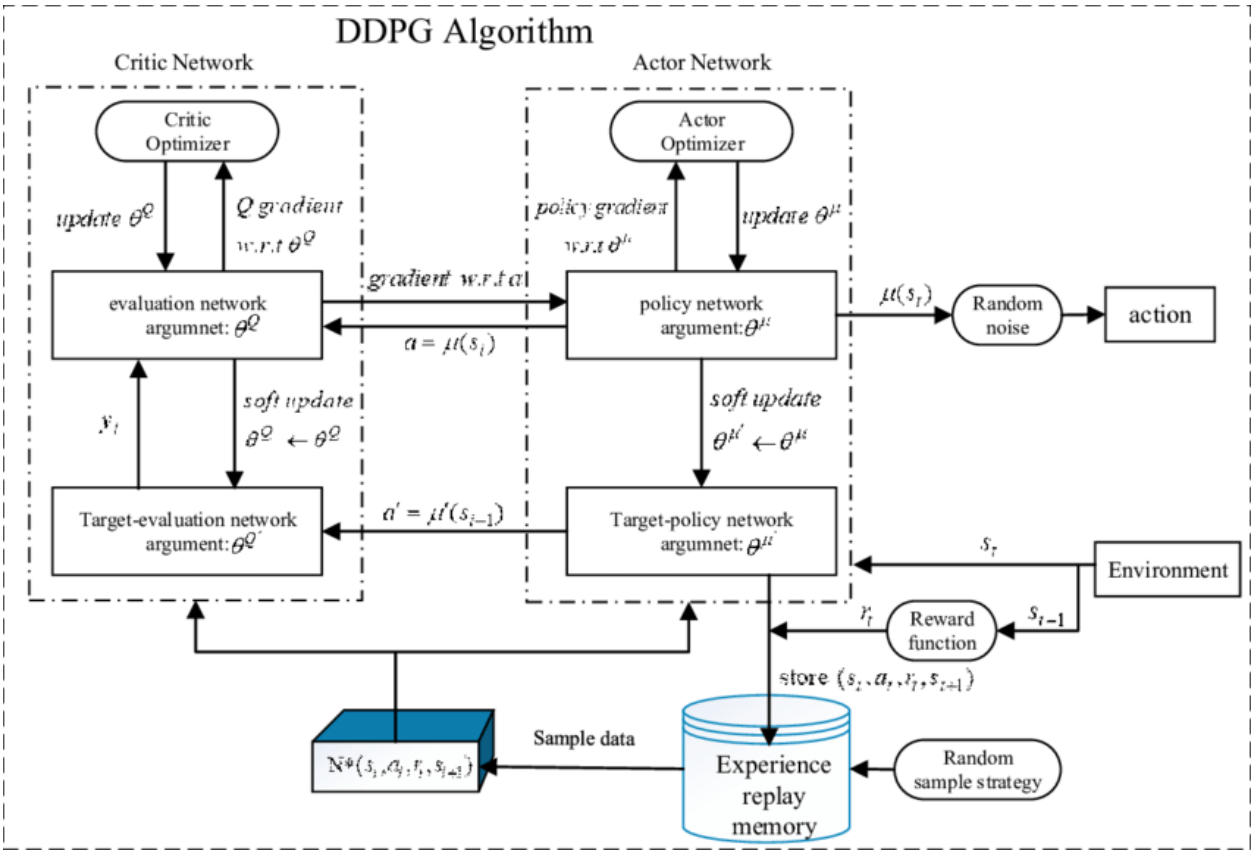
```
GAMMA = 0.99
TAU = 1e-3
LR_ACTOR = 1.5e-4
LR_CRITIC = 1.5e-4
WEIGHT_DECAY = 0.0001

## Actor NW
fc1_units=500, fc2_units=400, fc3_units=200
## Critic NW
fcs1_units=500, fc2_units=300

## OU process
mu=0
theta=0.1
sigma=0.2
```

DDPG

DDPG is a combination of actor-critic method and experience replay(from DQN).



(This image was cited from:

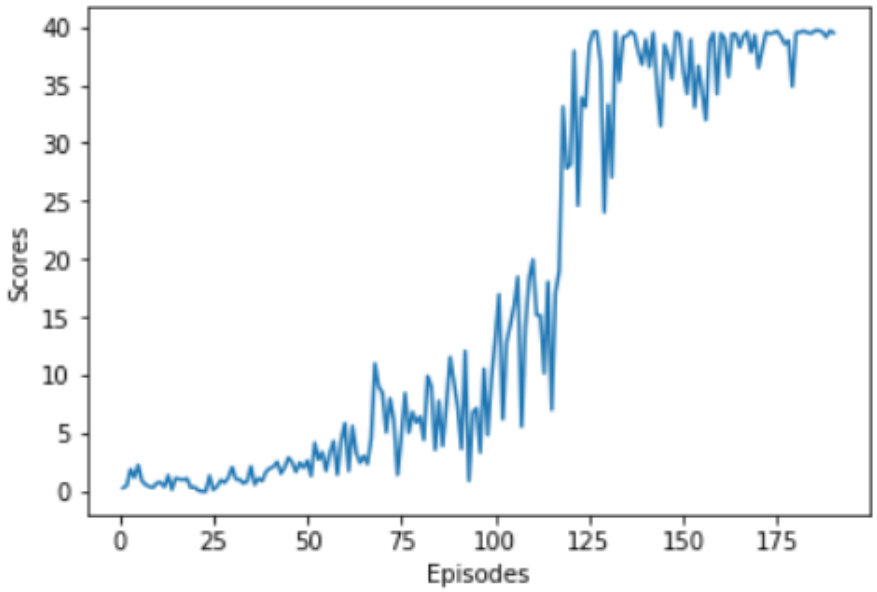
Guo, Baosu & Zhang, Yu-Xiu & Zheng, Weiqing & Yanhua, Du. (2020). An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning. Sensors. 20.

426. 10.3390/s20020426.)

Findings, Results and graphs

In the training, we discover that this fine tuning is truly similar to GAN's training. We fine tune the actor network with different fc numbers and layers of Actor and Critic Networks.

The agent reaches a reward over 30 after 125 episodes.



Future improvements

Using other algoeithms: SAC, PPO may be good for the task.