Deep Reinforcement Learning Nanodegree Project 3 Report

Ohn Kim

Learning Algorithm

I used DDPG algorithm to solve this problem.

Actor-Critic is a mix of policy-based and value-based methods. Policy based agent (actor) determines what action to take, and value-based agent (critical) determines the value of the current state and action. The representative algorithm of Actor-critical learning, DDPG (Deep Deterministic Policy Gradients), is learned in the following ways:

1. Actor makes an action based on state.
2. Critical is taught to predict the reward based on state, action and to create a value like the actual reward.
3. The actor receives an expected reward by delivering the action he created from state to critical and learns to maximize the expected reward.

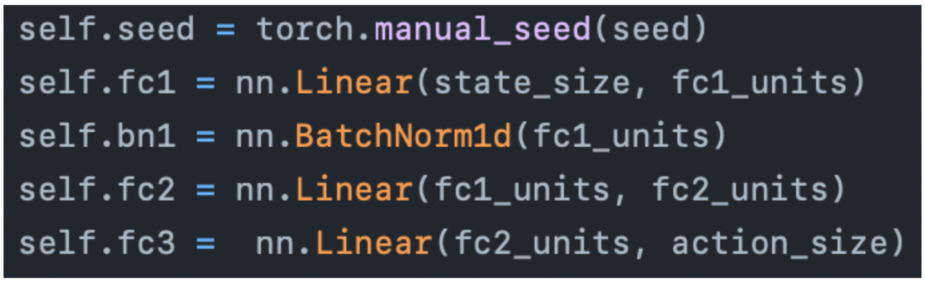
Hyperparameters

- BUFFER_SIZE = int(1e6) # replay buffer size
- BATCH_SIZE = 256 # minibatch size
- GAMMA = 0.99 # discount factor
- TAU = 2e-3 # for soft update of target parameters
- fc1 = 400 # Number of units in the first hidden layer
- fc2 = 300 # Number of units in the second hidden layer
- Actor LR = 1e-3 # actor learning rate
- Critic LR = 1e-3 # critic learning rate
- WEIGHT_DECAY = 0 # L2 weight decay
- EPSILON = 1.0 # for epsilon in the noise process (act step)
- EPSILON_DECAY = 1e-6 # epsilon decay rate
- GRAD_CLIPPING = 1.0 # gradient clipping

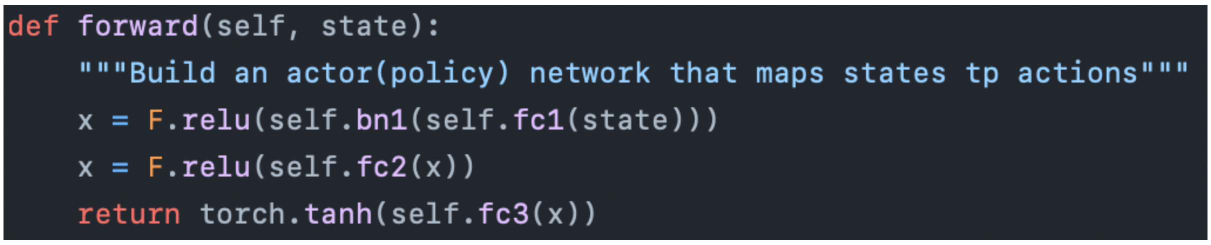
Model architecture

The structure of the actor and critic model is as follows. The fully connected layer is connected by ReLU, and in the case of the actor, tanh activation function was applied at the end.

Actor

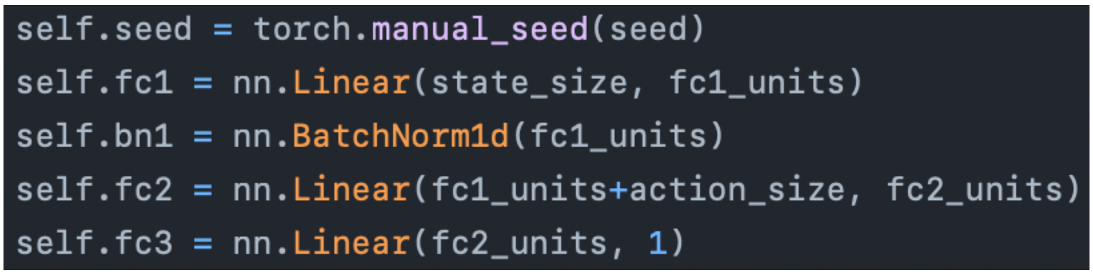


```python
self.seed = torch.manual_seed(seed)
self.fc1 = nn.Linear(state_size, fc1_units)
self.bn1 = nn.BatchNorm1d(fc1_units)
self.fc2 = nn.Linear(fc1_units, fc2_units)
self.fc3 =  nn.Linear(fc2_units, action_size)
```

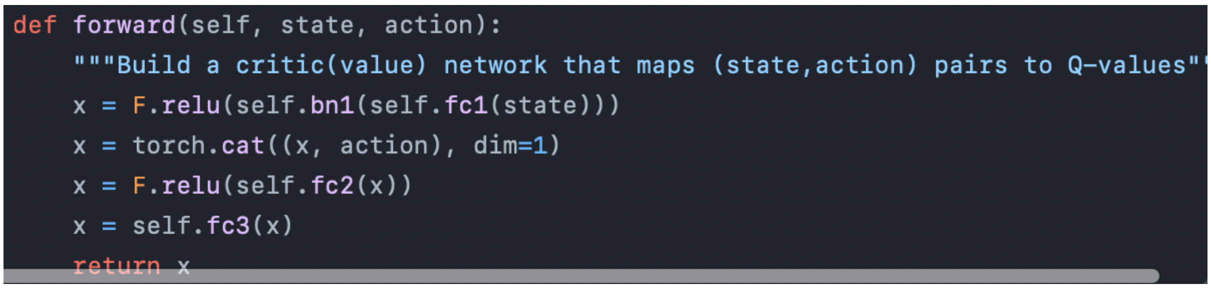


```python
def forward(self, state):
    """Build an actor(policy) network that maps states tp actions"""
    x = F.relu(self.bn1(self.fc1(state)))
    x = F.relu(self.fc2(x))
    return torch.tanh(self.fc3(x))
```

Critic



```python
self.seed = torch.manual_seed(seed)
self.fc1 = nn.Linear(state_size, fc1_units)
self.bn1 = nn.BatchNorm1d(fc1_units)
self.fc2 = nn.Linear(fc1_units+action_size, fc2_units)
self.fc3 = nn.Linear(fc2_units, 1)
```



```python
def forward(self, state, action):
    """Build a critic(value) network that maps (state,action) pairs to Q-values"
    x = F.relu(self.bn1(self.fc1(state)))
    x = torch.cat((x, action), dim=1)
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

Batch normalization and ReLU function are applied.

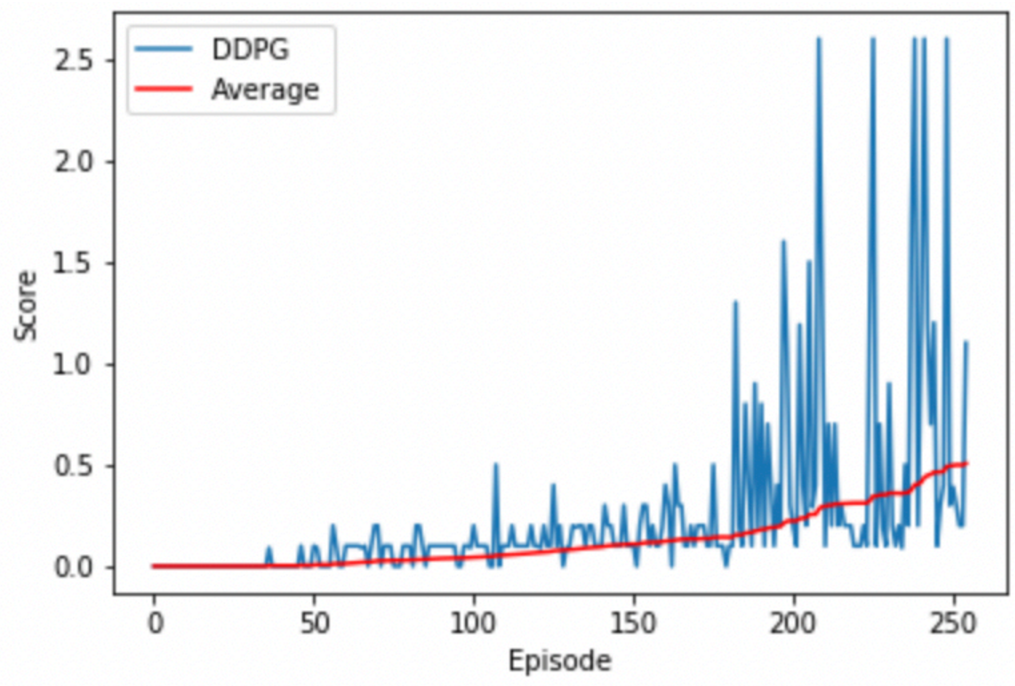
# of episodes needed to solve the environment

```
Episode 10 (4s) Max score: 0.000        Average: 0.000
Episode 20 (4s) Max score: 0.000        Average: 0.000
Episode 30 (4s) Max score: 0.000        Average: 0.000
Episode 40 (4s) Max score: 0.000        Average: 0.002
Episode 50 (4s) Max score: 0.000        Average: 0.004
Episode 60 (4s) Max score: 0.000        Average: 0.011
Episode 70 (25s)        Max score: 0.200        Average: 0.024
Episode 80 (9s) Max score: 0.100        Average: 0.030
Episode 90 (8s) Max score: 0.100        Average: 0.037
Episode 100 (8s)        Max score: 0.090        Average: 0.042
Episode 110 (8s)        Max score: 0.100        Average: 0.054
Episode 120 (15s)       Max score: 0.100        Average: 0.066
Episode 130 (9s)        Max score: 0.090        Average: 0.079
Episode 140 (15s)       Max score: 0.100        Average: 0.094
Episode 150 (9s)        Max score: 0.100        Average: 0.109
Episode 160 (19s)       Max score: 0.190        Average: 0.120
Episode 170 (13s)       Max score: 0.100        Average: 0.134
Episode 180 (5s)        Max score: 0.000        Average: 0.143
Episode 190 (18s)       Max score: 0.190        Average: 0.174
Episode 200 (33s)       Max score: 0.300        Average: 0.223
Episode 210 (146s)      Max score: 1.400        Average: 0.294
Episode 220 (18s)       Max score: 0.100        Average: 0.311
Episode 230 (15s)       Max score: 0.100        Average: 0.352
Episode 240 (16s)       Max score: 0.200        Average: 0.403
Episode 250 (30s)       Max score: 0.300        Average: 0.493

Environment solved in 255 episodes.     Average score: 0.506
```

After a total of 255 episodes, average score is over +0.5

Plot of rewards



Ideas for Future Work

I used the DDPG model to solve this problem. This model could have solved the problem enough, but other algorithms could be used for better performance. Using prioritized experience replay is helpful for better performance. And more optimization of hyperparameters is needed.