Report

Learning Algorithm:

I used similar agent from my implementation of continuous control project. The agent uses DDPG(Deep Deterministic Policy Gradient) learning algorithm to train itself. DDPG uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a target policy network.

The Actor directly maps states to actions (the output of the network directly the output) instead of outputting the probability distribution across a discrete action space. The Critic or the Q network predicts the Q value given the state and action values as input. The target networks are time-delayed copies of their original networks that slowly track the learned networks. Using these target value networks greatly improve stability in learning.

As used in Deep Q learning (and many other RL algorithms), DDPG also uses a replay buffer to sample experience to update neural network parameters. During each trajectory roll-out, we save all the experience tuples (state, action, reward, next_state) and store them in a finite-sized cache — a “replay buffer.” Then, we sample random mini-batches of experience from the replay buffer when we update the value and policy networks.

Hyperparameters:

I had to tweak the hyperparameters from my implementation especially I had to increase the replay buffer because the experiences are becoming too similar too fast and I had to decrease the learning rate of the actor to stop the agent from crashing. With these new hyperparameters the model achieved the required reward just within 885 episodes.

replay buffer size = 10000000
minibatch size for training = 128
discount factor GAMMA = 0.99
for soft update of target parameters TAU = 1e-3
LR for the actor = 1e-4
LR for the critic = 1e-3

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:

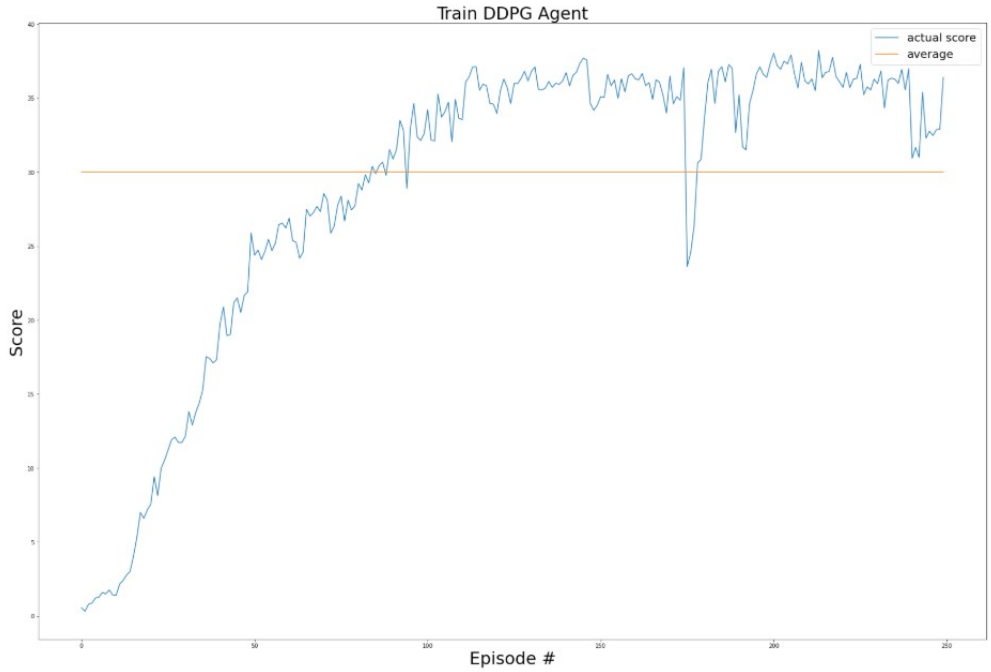
$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    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

plot of average rewards:



Network deep learning Model:

ACTOR:

First layer: 24 X 256

Second layer: 256 X 128

Final layer: 128 X 2

CRITIC:

First layer: 24 X 256

Second layer: 258 X 128

Third layer: 128 X 64

Final layer: 64 X 1

Ideas for Future Work:

We can clip the gradients of the critic network to prevent the agent from crashing.

OpenAI’s recent paper on MADDPG can be used to exploit the multi agent training environment for better use.