Project 2 : Continuous Control

Adithya Subramanian
Deep Reinforcement Learning Nanodegree, Udacity
8/1/2019

Implementation description :

1. TD_3_agent.py:
    a. Agent class contains information about the current state representation, allowed actions,. The RMSprop optimizer, OUNoise, Actor networks - local & target, The Critic Networks - local & target and replay buffer object is initialized in this class.
    b. The step function of the agent class adds the current state, action, reward, and next state pair into the replay buffer by calling its add function and updates the weights of q-networks every 4 steps if the size of the buffer is greater than the batch size.
    c. The act function returns the optimal actions as per the current actor network's deterministic policy.
    d. The learn function is responsible for computing the mean squared error between the expected Q-value and target Q-value. The target is computed by finding the minimum between the two Q- values learnt Q1 and Q2.
    e. The Actor is learnt in contrast to the critic is updated less frequently than the Critic in the learn function.
    f. The soft_update function updates the target weights using τ parameter and local network weights.
    g. The class ReplayBuffer is the class definition for the buffer memory it is initialized with a deque of pre-defined buffer size which will store a named tuple.
    h. The add function in ReplayBuffer adds the newly visited state transition to the buffer memory.
    i. The sample function samples a complete transition using a uniform sampling technique.
    j. The OUNoise class is the class definition for Ornstein-Uhlenbeck process it is initialized with mean and variance specified during the initialization of the TD3 Agent.
    k. The reset function of the OUNoise class resets in the current state to the mean specified while the object was initialized.

    l. The sample function of the OUNoise class samples a noise at random and then updates the internal state.
2. Model.py :
    a. The actor model used for transforming the state information vector into the action vector is written in the class Actor.
    b. The critic model used for transforming the state information vector and action vector is written in the class Critic.
    c. The networks used in the code are one's which was used in the actual implementation of the Author.
3. CC.ipynb :
    a. The TD3 function runs multiple episodes of the agent's interactions with the environment for a certain number of transitions. The network stores the average reward collected by the agent over the 100 episodes. Once the agent accumulates over the 40+ reward in 100 episodes the environment is considered to solved and model's weight are saved.

Learning algorithm :

Introduction :

The nanodegree lectures have enlightened us on DDPG algorithm which can be used in the continuous actions spaces and has been seen to achieve great performance. On other the other it has also been observed that DDPG is quite sensitive to the hyperparameters of the model. Similar to DQN DDPG also overestimates the Q-values, which then leads to the policy breaking. Twin Delayed DDPG (TD3) on the other avoids these issue by first adding noise to the target action making it harder for the policy to exploit Q-value errors by smoothing out Q along changes in action, second by learning two Q-functions instead of one it uses the smaller of the two Q-values to form the targets in the Bellman error loss functions. Finally, it updates the policy less frequently than the Q-function.

Explanation:

Unlike DDPG TD3 simultaneously learns two Q-functions, Q_ϕ_1 and Q_ϕ_2, by mean square Bellman error minimization. The Q- functions learnt by comparing MSE on the same target as shown Equation 1 and this target is decided by choosing minimum between the two as shown in Equation 2 but the policy is learnt by maximizing Q_ϕ_1 similar that of DDPG as shown in Equation 3.

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')),$$

Equation 1.

$$L(\phi_1, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}} \left[\left(Q_{\phi_1}(s, a) - y(r, s', d) \right)^2 \right]$$

$$L(\phi_2, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}} \left[\left(Q_{\phi_2}(s, a) - y(r, s', d) \right)^2 \right]$$

Equation 2.

$$\max_{\theta} \mathop{\mathrm{E}}_{s\sim\mathcal{D}} \left[Q_{\phi_1}(s, \mu_\theta(s)) \right]$$

Equation 3.

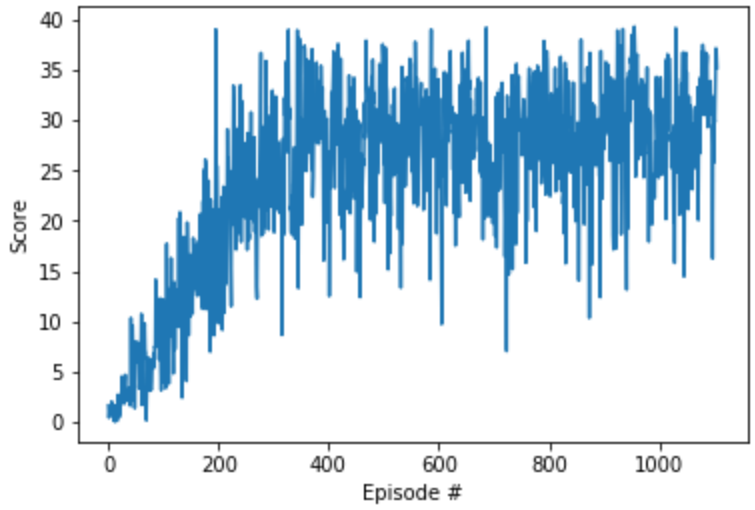
However, the actor is updated is less frequently than the critic. The idea behind using the minimum among them as a target is to make sure that we choose an estimate which is less prone to overestimation. Although the authors have also mentioned that the algorithm is still vulnerable to underestimation but underestimation is not as much as a problem as that of the overestimation.

To prevent the policy to exploit narrow peaks for certain actions and avoid a suboptimal solution we add noise to the target policy actions and then clip to them so that they lie in the range of a_{Low} and a_{High}. This technique acts a regularization technique for the algorithm which smoothens the target policy. The target action for the state s' is given by Equation 4.

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

Equation 4.

Reward Plot:



The rewards acquisition is much more stable than that of DDPG but is a lot more slower than DDPG as it is make use of the sharp peaks early as that of the DDPG hence making it less exploitative.

Future work:

1. Implementing Soft-actor-critic which was another algorithm which was published almost at same time of that of TD(3) for solving the task.
2. Implementing multi-agent DDPG for using multiple agents simultaneously.

THANK YOU