DRLND – Continuous Control – Project2

Introduction

The objective of this project is to train an artificial intelligent agent maintain its position at a target location for as many time steps as possible. I have selected the second version of the environment. This version of the environment has 20 agents in it and to solve it, need to consider the presence of many agents. The agent trained must get an average score of +30.

Solution

This scenario being, one with continuous action space having multiple agents, a policy based reinforcement algorithm such as Deep Deterministic Policy Gradient (DDPG) would be the ideal starting point. DDPG is an off-policy algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy.

To solve the Reacher environment, DDPG agent is implemented. DDPG is often classified as an Actor-Critic method. The implementation therefore consists of an Actor and a Critic network.

Actor is used to approximate the optimal policy deterministically. This would return the best possible action. Critic learns to evaluate the optimal action value function by using actor's best action

Actor Network: This network has an input layer which takes the states which are then passed through batch normalization layer to next layer. Two hidden layers with size 400 and 300 were tried for this experiment. The output layer was a fully connected layer with output size set as action size representing action probability. Leaky Relu activation layer was used after first and second layer. Tanh activation was used on the last layer.

Critic network: Network used had an input layer which takes the states which are then passed through batch normalization layer to next layer. Two hidden layers with size 400 and 300 were used here as well. Actions were added in the first hidden layer. This proved effective. The output layer was a fully connected layer with output size of 1. Leaky Relu activation function was used after first and second layer.

The algorithm also has a replay buffer to improve convergence.

In DDPG, there are two copies of network weights or in other words, a regular network for the actor and one for critic, a target network for the actor and one for critic. Unlike DQN, the update from regular to target doesn't happen altogether. This involves soft update were the regular network weights are slowly blended into target network weights. Soft updates enable faster convergence.

Algorithm

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

DDPG makes use of an adaptive noise to boost performance. Here I have used Ornstein-Uhlenbeck Noise process whose implementation was provided.

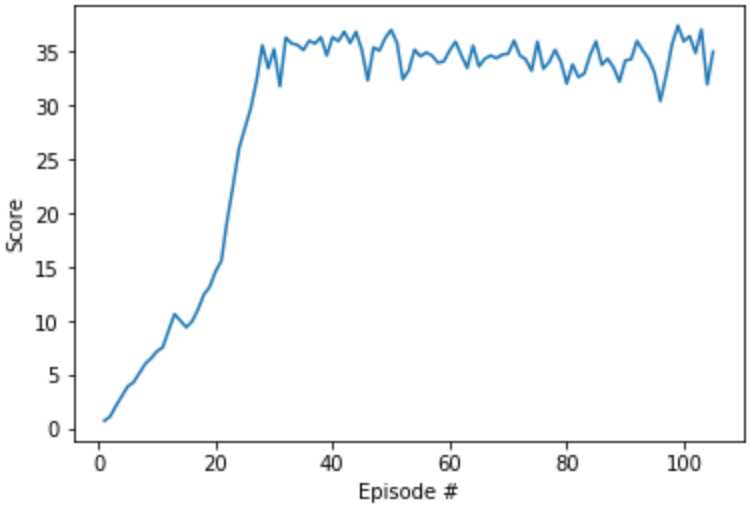
Hyper parameters

After verifying that the DDPG algorithm was working, I embarked on tuning the hyperparameters by running multiple training sessions. Below were the hyperparameter values that provided best results

```
BUFFER_SIZE = 1e5       # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay

LEARN_EVERY_X_TIMESTEPS = 20
UPDATES_PER_LEARN_STEP = 10
EPSILON = 1.0
EPSILON_DECAY = 1e-6
```

Using DDPG algorithm with above mentioned hyper parameters resulted in the environment getting solved in 105 episode(the average score of last 100 episode was above 30).




```
Episode 100     Average Score: 28.55    Score: 35.97
Episode 105     Average Score: 30.19    Score: 35.01
Environment solved in 5 episodes!       Average Score: 30.19
```

Future Enhancements

The results that we obtained by using the DDPG algorithm on continuous action space could further be improved by

- Exploring other Deep RL algorithms such as TRPO, Distributed Distributional Deterministic Policy Gradient(D4PG) which can be used for continuous control
- Further optimization of hyperparameters such as different values for tau in soft update, using huber loss for calculating critic loss, noise decay and additional layers in deep neural network