Report

Implementation of the DDPG algorithm to solve the Reacher Continuous Control Environment.

Model Architecture:

The model is based on Udacity's DDPG Bipedal environment algorithm.

I used two Neural networks each for actor and critic, namely-

actor_target, actor_local, critic_target, critic_local.

Actor Networks:

- Input Layer – state_size
- Hidden Layer – 512 units - ReLU
- Hidden Layer – 256 Units - ReLU
- Hidden Layer – 64 Units - ReLU
- Hidden Layer – 16 Units - ReLU
- Output Layer – action_size – Tanh(Outputs 4 values for action between -1 and +1)

Critic Networks:

- Input Layer – state size
- Hidden Layer – 512 units - ReLU
- Hidden Layer – 256 Units + action_size - ReLU
- Hidden Layer – 64 Units - ReLU
- Hidden Layer – 16 Units - ReLU
- Output Layer – 1 Unit – ReLU(Outputs a Q_value given state S and action A)

Hyperparameters:

- Learning Rate: 0.001 for both types of Networks
- Batch Size = 128
- Buffer Size = 1e6
- Gamma = 0.99
- Tau = 0.001
- Ornstein-Uhlenbeck noise parameters (0.15 theta and 0.2 sigma.)

Algorithm Details:

- Create an instance of an Agent with non-default input parameters number of agents, state_size and action_size.
- In the beginning of an episode, the algorithm generates 20 random initial states of the environment, one for each agent.
- The 20 agents then pass them through the actor_local network. This generates an action per agent.
- This action is passed to the environment to generate, next_state, reward, done for each agent.
- Each agent then uses its step function to add the state, action, reward, next_state, done that it generated in the previous steps to an instance of the Replay_Buffer, we call memory, as an experience.

- If there are more than 128 experiences in memory, then each agent will randomly sample 128 experiences and update the actor-critic networks.
- Therefore, at each time-step we are updating the actor-critic networks 20 times.
- The rewards for all steps in an episode are added and stored in a list called scores as per the index of the agent.
- The mean of scores in then the average score for that episode.

Updating the networks:

Here I will describe, how the actor-critic networks are updated every step by an agent.

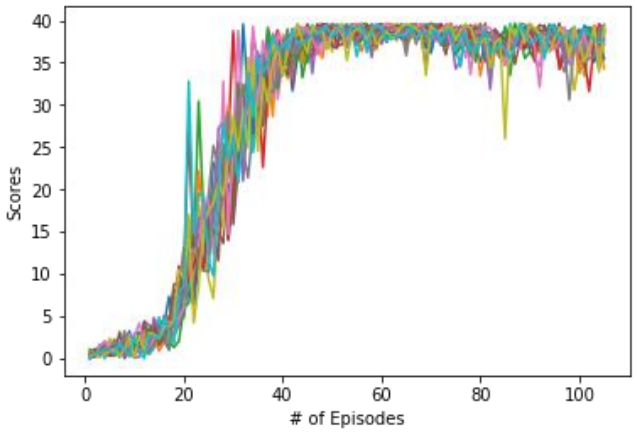
- The agent samples 128 experiences.
- The next_states are passed to the actor_target network to predict actions after the next_states, as actions_next.
- The next_states and actions_next are passed to the critic_target network, to predict Q_values for the next_states as Q_targets_next.
- Q_targets is calculated using rewards + (gamma* Q_targets_next).
- Q_expecteds is generated by passing states and actions through the critic_local network.
- The Q_targets calculated above will be used to train the critic_local network, such that given the states and the actions, the critic_local network should generate Q_targets. This is done by minimizing the Mean_Squared_Error Loss between Q_expecteds and Q_targets. I also clipped the gradients of the critic_local network before optimizing it.
- The actor_local network is then updated as below.
- States are passed to the actor_local network to predict actions as actions_pred.
- Actions_pred are then passed to the updated critic_local network to generate Q_values.
- We intend to optimize the parameters of the actor_local network by maximizing the Q_values. Hence, we minimize -1*Q_values.
- After the local networks are updated, we do a soft_update of the target network parameters, such that the updated version is 99.9% of its previous version and 0.1% of the updated local network parameters.

Observations:

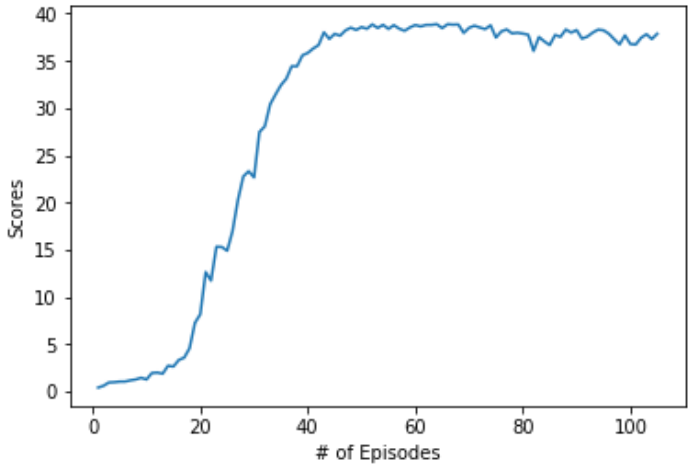
When I first attempted this project, I tried the one-agent version, but got very disappointing results. Then I amended my Agent class and the main algorithm for the 20-agent version and I got impressive results.

I made few minor changes to the network, added gradient-clipping and increased the batch size from 64 to 128.

Following is the plot for all agents –



Plot for mean score of the agents –



As we can see, I got stable and >30 scores after around episode 40. My environment was solved in 105 episodes with a score of +30.13. That means, the average of scores from episode 6 to 105 was +30.13.

Ideas for future work:

I plan to implement the Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG) methods as they have achieved better performance. I also plan to test D4PG algorithm for both the Reacher and the Crawler environments.