**REPORT**

**INTRODUCTION**

The project demonstrates how policy-based methods can be used to learn the optimal policy in a model-free Reinforcement Learning setting using a Unity environment, in which a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector is a number between -1 and 1.

**Learning Algorithm**

I have chosen Deep Deterministic Policy Gradients (DDPG) to solve the problem. Deep Deterministic Policy Gradient (DDPG) lies under the class of Actor-Critic Methods but is a bit different than the vanilla Actor-Critic algorithm. The actor produces a deterministic policy instead of the usual stochastic policy and the critic evaluates the deterministic policy. The critic is updated using the TD-error and the actor is trained using the deterministic policy gradient algorithm. In fact, it could be seen as approximate Deep-Q Network (DQN) method. The reason for this is the critic in DDPG is used to approximate the maximizer over the Q-values of the next state, and not as a learned baseline.

**MODEL**

The actor network architecture is comprised of two fully connected hidden layers of 256 and 128 units each with Leaky ReLU activations. In order to help speed up learning and avoid getting stuck in a local minimum, batch normalization was introduced. The hyperbolic tan activation was used on the output layer for the actor-network as it ensures that every entry in the action vector is a number between -1 and 1. Adam was used as an optimizer for both actor and critic networks.

The critic network is similar to actor network with exception of tan activation function. An additional fully connected layer of 128 units is used to output the required Q values.

Since we are dealing with 20 agents, I went ahead with updating the weights after every 20 steps and for every such step, updating the weights 10 times. There are also a few techniques which contributed significantly towards stabilizing the training:

**Fixed targets:** Originally introduced for DQN, the idea of having a fixed target has been very important for stabilizing training. Since we are using two neural networks for the actor and the critic, we have two targets, one for actor and critic each.

**Soft Updates:** In DQN, the target networks are updated by copying all the weights from the local networks after a certain number of epochs. However, in DDPG, the target networks are updated using soft updates where during each update step, 0.01% of the local network weights are mixed with the target networks weights, i.e. 99.99% of the target network weights are retained and 0.01% of the local networks weights are added.

**Experience Replay:** This is the other important technique used for stabilizing training. If we keep learning from experiences as they come, then we are basically observed a sequence of observations each of which are linked to each other. This destroys the assumption of the samples being independent. In ER, we maintain a Replay Buffer of fixed size (say N). We run a few episodes and store each of the experiences in the buffer. After a fixed number of iterations, we sample a few experiences from this replay buffer and use that to calculate the loss and eventually update the parameters. Sampling randomly this way breaks the sequential nature of experiences and stabilizes learning. It also helps us use an experience more than once.

All of the above mentioned techniques were incorporated. The entire implementation was done in PyTorch.

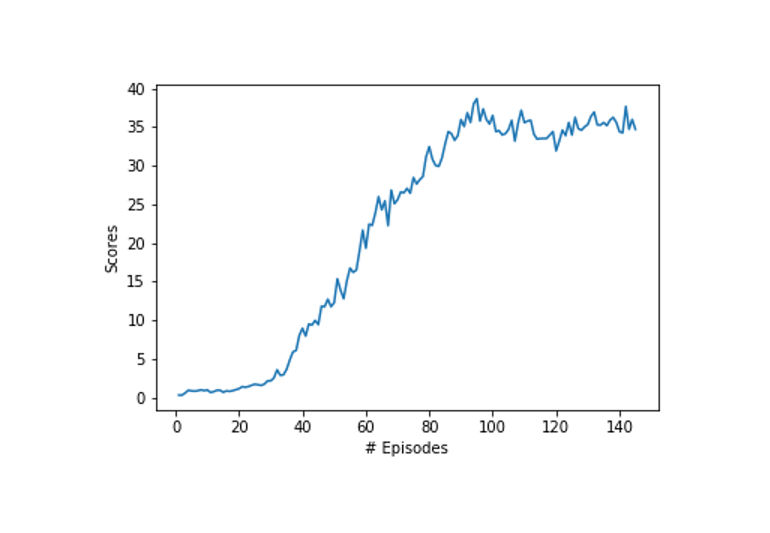
**Hyperparameters:**

There were many hyper parameters involved in the experiment. The value of each of them is given below:

|  |  |
| --- | --- |
| **Hyper parameters** | **value** |
| Replay buffer size | 1e6 |
| Batch size | 1024 |
| gamma | 0.99 |
| tau | 1e-3 |
| Actor learning rate | 1e-4 |
| Critic learning rate | 3e-4 |
| Update interval | 20 |
| Update times per interval | 10 |
| Number of episodes | 2000 |
| Max number of time steps per episode | 1000 |
| Leak for LeakyReLU | 0.01 |

**Results:**

The best performance was achieved by **DDPG** where the reward of +30 was achieved in **145** episodes



**Ideas of Improvement:**

* Other algorithms like TRPO, PPO, A3C, A2C that have been discussed in the course could potentially lead to better results as well.
* The Q-prop algorithm, which combines both off-policy and on-policy learning, could be good one to try.