Report

Learning Algorithm Description:

I opted Double Deterministic Policy Gradient (DDPG) to solve the given Reacher environment. The implementation consists of both actor and critic with two neural networks (NN) each (designed with same network architecture), one for Q updates and other for Q targets using the Bellman equation. The actor is used to learn the best action for each state in a deterministic manner. On the other hand, critic learns to deterministically evaluate the optimal action value function by using the actors best believed action. The NN architecture for both actor and critic has 2 hidden layers each besides input and output layer. The first hidden layer is a linear layer with dimensions (33, 128). The second hidden layer is a fully connected layer with dimensions (128,128) which is then finally fed to an output layer of dimensions (128,4).

The agent's actor and critic Q networks namely Qlocal and Qtarget are initialized first with same uniform random weights. The weights of output layers for both actor and critic are initialized randomly with values between -3e-3 and 3e-3. The agent then interacts with the Reacher environment, obtaining the tuples (state, action, reward, next state). Experience Replay is then used over these tuples, by storing them as samples into a finite buffer. A batch of samples from this buffer are randomly sampled each time and passed into actor and critic's Qlocal and Qtarget networks. The actor Qtarget selects the best possible action for the next states in the samples and critic Qtarget utilizes them to Qvalues inside the Bellman equation, thereby computing mean squared critic loss with critic Qlocal values. Similary, the actions predicted by actor's Qlocal network for the current states are also used in computing the actor's mean difference loss with critic Qlocal values. Thus, both actor and critic loss functions are computed and back-propagation is performed to minimize their respective losses with model parameters and also updating the actor and critic's Q network weights using their individual Adam Optimizers with learning rates 2e-4 and 2e-4 respectively. Qtarget network parameters for both actor and critic are then updated at each learning iteration using the equation below:

$$\theta\_target = \tau * \theta\_local + (1 - \tau) * \theta\_target$$

Here, τ is a update hyper-parameter (tuned to 1e-3) in my alogorithm. Also, the agent to perform some exploration in the reacher environment we add some noise to the action selected each time using Ornstein-Uhlenbeck process with parameters $\theta = 0.15$ and $\sigma = 1$.

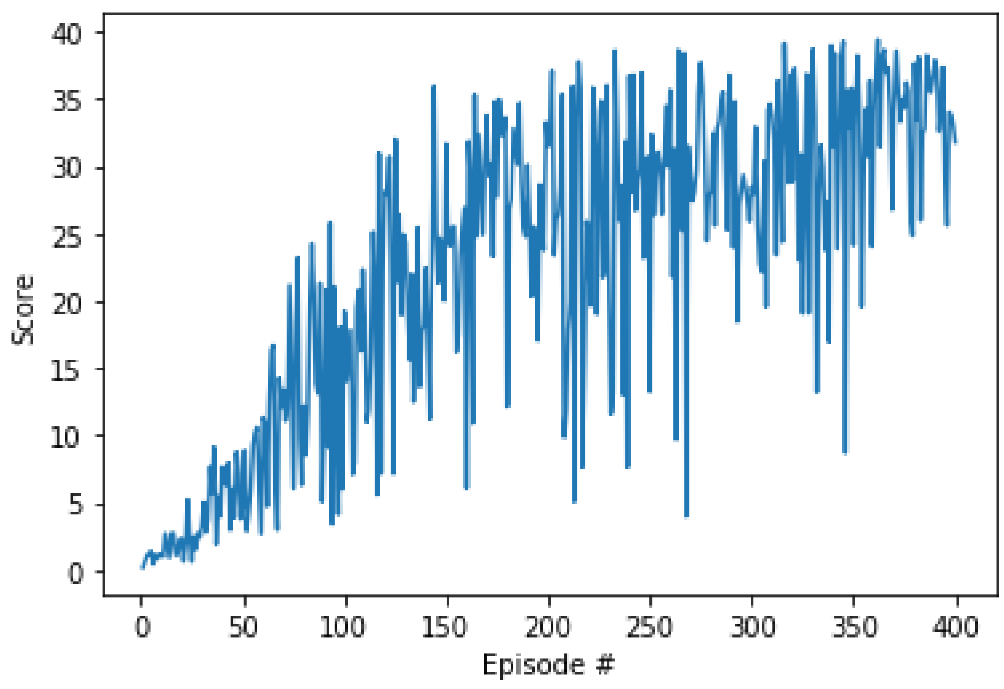
In brief, the DDPG algorithm can be stated as follows:

1. Initialize DDPG network Actor and Critic, local and target networks with some initial approximation
2. Select action according to current policy and exploration noise ($a_t = \mu(s|\theta^{\mu}) + \mathcal{N}_t$)
3. By interacting with the environment, obtain the $tuple(s_t, a_t, r_t, s_t')$
4. A mini-batch of random samples are selected and sent into DDPG network
5. Update critic by computing Loss: $L = \frac{1}{N}\sum_i\left(r_i - Q(s_i, a_i|\theta^Q)\right)^2$ if episode has ended or $L = \frac{1}{N}\sum_i\left((r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}|\theta^{\mu})|\theta^Q)) - Q(s_i, a_i|\theta^Q)\right)^2$
6. Update actor policy using sampled policy gradient $\frac{1}{N}\sum_i\left(\nabla Q(s_i, a_i|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})\right)$
7. Using Adam Optimizer with learning rate, by minimizing the loss with respect to model parameters update both actor and critic target networks
8. Repeat from step 2 until converged

Hyper-parameters Selection:

To the architecture designed in model.py, soft updates of Q-target network with tau=1e-3. OUNoise is chosen here for the exploration of learning agent with parameters $\theta = 0.15$ and $\sigma = 1$. A batch size of 128 samples are randomly selected each time and used for the actor and critic loss function. Gradient clipping is also performed after critic loss computation to prevent exploding gradients. Adam Optimizer is then used at every step for both actor and critic with learning rate (LR=2e-4) after performing back propagation at every step. The discount factor for episodic rewards (gamma) is set to 0.99. Number of training episodes considered are 400. After performing hyper-parameter tuning and fixing the hyper-parameters to above mentioned values, the agent is able to solve the environment with an average score of 31.64 over 100 consecutive episodes. The weights of the learnt model are saved under the filename "model.pt" in the same folder.

Reward Plot:



Future Ideas:

Based on the designed NN architecture and hyper-parameter tuning, I could solve the environment with an average reward of 31.64 over 100 episodes. However, I have only used one agent in this problem. I want to consider multiple agents during training and observe the behaviour and complexity of training accordingly.