Deep Reinforcement Learning Algorithm for Robotic Arm

Shivoh Chirayil Nandakumar[1]

[1]

830 Words — Student ID:nil

Keywords: DDPG, Continuous action space — 16th February, 2021

1 Introduction

This report provides you an insight on how i implemented the deep deterministic policy gradient(DDPG) algorithm and trained the agent for providing the required torque for the two joints of the Robotic arm to reach a target location. Each Action vector consists of 4 values which represents the torque required for two joints. And the value of the torque varies from -1 to 1. The state space is of 33 dimensions.It includes the information regarding the position and orientation of the robotic arm and the information related to target location.Its worth noting that action space and state space are continuous rather than discrete. All the code for this project was developed in Python 3 mainly using the Pytorch library.

2 Methods

The DQN algorithms are useful in discrete action spaces. But, as the dimension of action space increases the complexity to obtain the optimal action from the Q value, even for a single state becomes computationally very expensive and when considering continuous action spaces, it becomes nearly impossible. So, various methods are explored to solve this conundrum. One of the method which was successful in solving continuous acion spaces was DDPG.Thus,I used the Deep deterministic policy gradient or DDPG algorithm to train the agent for this Project.Its an actor-critic method which has many similarities with the DQN algorithm. Just like DQN algorithm, it has a separate target deep neural network which lags from the local deep neural network. But, unlike the DQN, soft updating is done for the target network. Moreover,similar to DQN, experience replay was used to enhance the efficiency of training. It should be noted that, there are two target network, one for the actor and one for the critic. Similarly, there are two local networks.

The DDPG solves the continuous action space issue by obtaining an approximate optimal action which maximises the Q-value from a target policy network. The fundamental idea regarding actor-critic method is that, the critic provides better direction for the actor to take action. Specifically, the critic will take in the state and actions to give out the optimal Q value.and the actor takes in the state to give out the best action to take, hence the name deterministic policy gradient. The deep neural network of Critic is trained by minimising the loss function using stochastic gradient descent[1]. The loss function here is the MSBE or mean square bellman Equation as shown below.

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$

Fig. 1. Optimal Action-Value function

The loss function for training the (Critic) neural network is as shown below.

$$L(\phi, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d) \sim \mathcal{D}} \left[\left(Q_\phi(s,a) - \left(r + \gamma (1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s')) \right) \right)^2 \right],$$

Fig. 2. Loss function

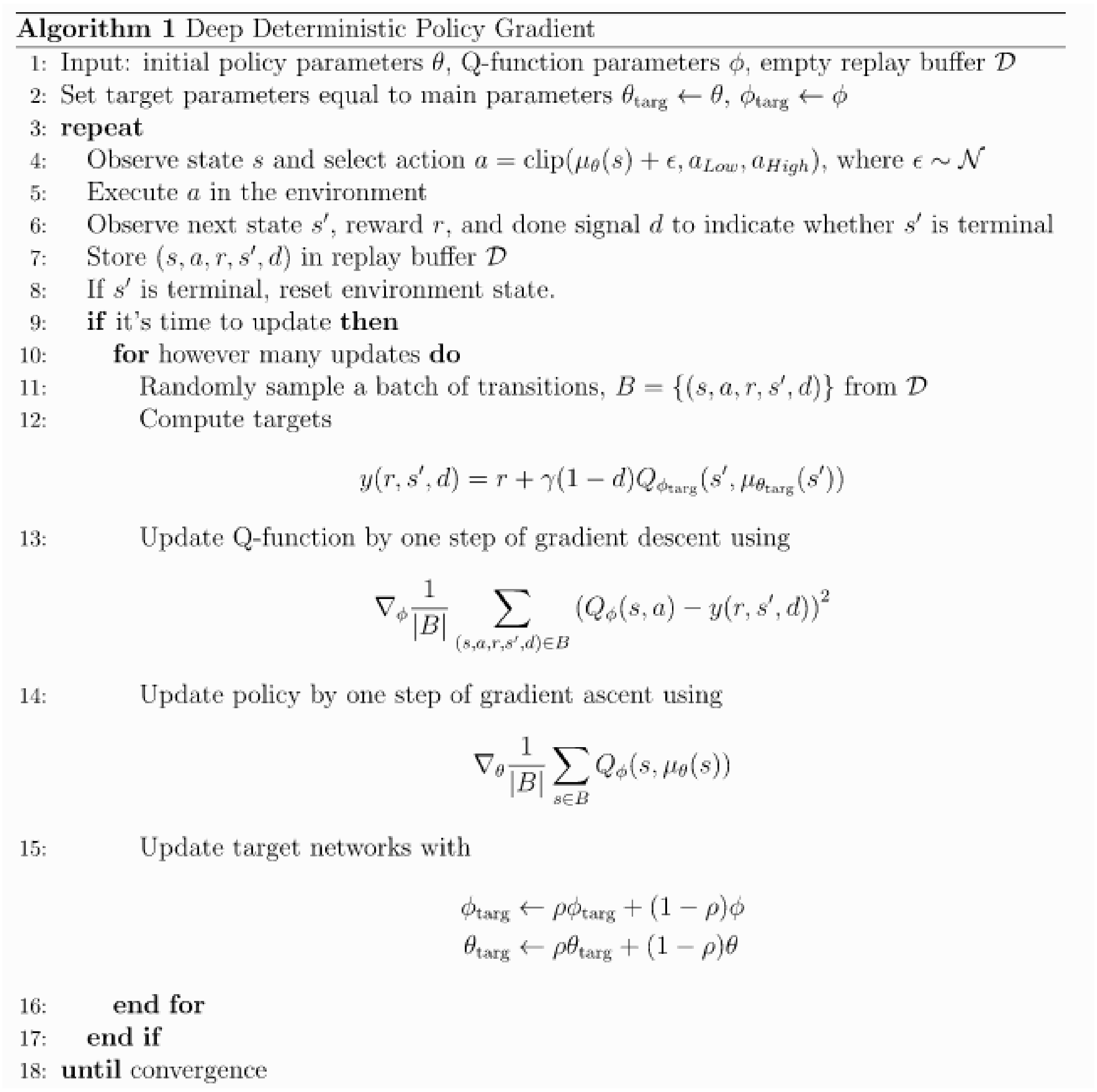
Here the current action values are compared with a target action values.The target action values are updated using a technique of soft update.The θ_i represent the current weight in the neural network and the θ_i^- represent the weights used in the target network. We should minimise the value between target action value and the current action value.

In regarding to policy optimisation, the gradient ascent method is used to find the maximum of the Q values. Since the action space is continuous,it is assumed that, the Q function is differentiable with respect to the actions thus, the gradient ascent method is used to solve this optimisation problem as shown below.

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

Fig. 3. Optimisation problem for Policy update

As given in the DDPG paper[2], the psudocode of the DDPG algorithm is attached below.



Algorithm 1 Deep Deterministic Policy Gradient

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
3: repeat
4:   Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:   Execute $a$ in the environment
6:   Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:   Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:   If $s'$ is terminal, reset environment state.
9:   if it's time to update then
10:    for however many updates do
11:      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:      Compute targets
             $y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$
13:      Update Q-function by one step of gradient descent using
             $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$
14:      Update policy by one step of gradient ascent using
             $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$
15:      Update target networks with
             $\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi$
             $\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$
16:    end for
17:  end if
18: until convergence
```

Fig. 4. DDPG Pseudo Code

2.1 Deep Neural network Architecture

For this project, i used two neural networks, one for the actor and other for the critic. Each has almost similar architecture, but with subtle differences. Actor consists of hidden layers with 256,128,64,32 and 16 units and the output layer has size equal to the dimension of the action space. Each hidden layer has an ReLU activation function except the output layer. The output layer has a tanh activation function. The output of the actor model is the action space for the agent. further, noise (Ornstein-Uhlenbeck process) was added to the action values to make the training more robust. Adam optimiser was used for optimising the weights of the Neural Network. In case of critic model, the first hidden layer consists of 256 units along with number of units corresponding to the size of action space. the subsequent layers consists of

128,64,32 and 16 units respectively.The output of the critic model is the Q-value for the given state action pair.

3 Results

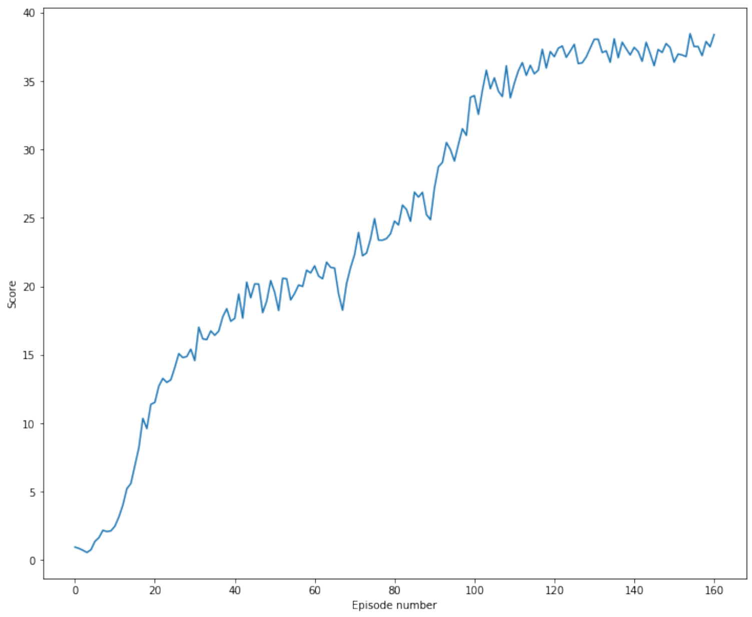


Fig. 5. The successful training session of DDPG Agent

The agent was successfully trained to obtain a score of 30+ for 100+ consecutive episodes in less than 152 episodes.

4 Future Work

Fine tuning various hyper parameters of the system can possibly improve the efficiency of the system. Adding more hidden layers and changing the number of the units of the layers found to have impact on the training speed of the agent. Prioritised experience replay can possibly improve the training speed. Further, various other deep learning algorithms which are suitable for high dimensional action spaces such as Proximal Policy Optimisation (PPO) and Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG) should achieve better performance. Further, Distributed Distributional Deterministic Policy Gradients (D4PG) algorithm should be explored to find out if its able to produce more robust agent.

5 References

1. OpenAI. *Deep Deterministic Policy Gradient.* 2019.
2. Silver, D. *Continuous control with deep reinforcement learning.* 2019.