2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA) | 978-1-6654-2490-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CAIBDA53561.2021.00025

Reinforcement Learning with Deep Deterministic Policy Gradient

Haining Tan*
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
*haining.tan@mail.utoronto.ca

Abstract—**This study reviews the major developments of Deep Deterministic Policy Gradient (DDPG) in the field of reinforcement learning. It is innovated by Deep Q-network ideas and can finally handle some much challenging problems that operate over continuous action space. The main idea of DDPG is to use an actor-critic architecture (shown in Figure 5) to learn much more competitive policies. It allows the model to use neural network function approximators to learn in large state and action spaces. Due to its strong capacity, DDPG has many useful applications to real world problems in the field like robotics and control systems. But like most of the model-free reinforcement learning methods, the requirement for a large number of training steps is still a major difficulty for DDPG.**

Keywords-component; Reinforcement learning, Q learning, Deep Q-network, Deep Deterministic Policy Gradient, continuous space, function approximator, experience replay.

I. Introduction

Over the past years, as more and more breakthroughs are made in deep reinforcement learning, it has gained much popularity and has been applied to solve many real-world problems. One famous research done by a group of people in DeepMind Technologies is to design the first deep reinforcement learning model to play multiple Atari 2600 games, which can learn control policies from high-dimensional sensory input successfully [1]. Following that first milestone achievement, many outstanding improvements in different deep reinforcement learning methods have been made. Minh et al. [1] designed a Deep Q-Network (DQN) method, which shows a significant improvement over the original Q-learning algorithm.

Although DQN can solve complex tasks with high-dimensional state spaces, it can only handle discrete and low-dimensional action spaces. Then based on the deterministic policy gradient (DPG) algorithm, Lillicrap et al. [2] presented a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional continuous action space, which is known as Deep DPG (DDPG). As a consequence, it outperforms many other algorithms and becomes one of the most popular methods used in many applications. This study will review those major developments in deep reinforcement learning in chronological order in the methodology section and present some of their state-of-art applications to some real problems in the discussion section.

II. Methodology

A. Technical background

One essential technical background in reinforcement learning is the definition of a Markov Decision Process (MDP). It contains the information of: $\mathcal{S}$ (State space), A (Action space), $\mathcal{P}$ (Transition probability), $\mathcal{R}$ (Immediate reward distribution), γ (Discount factor with $0 \leq \gamma < 1$).

We aim to find the optimal policy π which maximizes the expected return defined by the value function V^{π}:

We aim to find the optimal policy π which maximizes the expected return defined by the value function V^{π}:

$$V^{\pi}(s) = E_{\pi}[G_t \mid S_t = s] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} R_{t+k} \mid S_t = s\right] \quad (1)$$

The foundation of many reinforcement learning algorithms is the fact that the value functions satisfy a recursive relationship, which is known as the Bellman equation:

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s)\left[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid a,s) V^{\pi}(s')\right] \quad (2)$$

A closely related version for state-action value function Q^{π}:

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'} cP(s' \mid a,s) \sum_{a'} \pi(a' \mid s') Q^{\pi}(s',a') \quad (3)$$

B. Q Learning

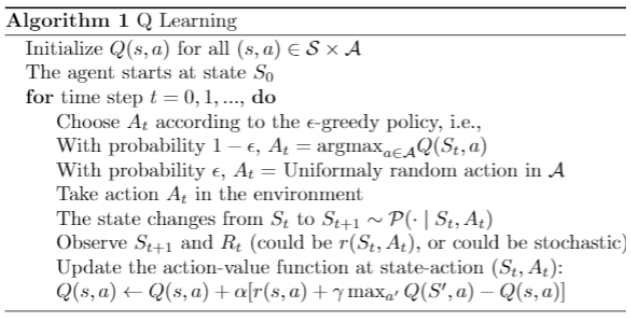
```
Algorithm 1 Q Learning
  Initialize $Q(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$
  The agent starts at state $S_0$
  for time step $t = 0, 1, ...,$ do
      Choose $A_t$ according to the $\epsilon$-greedy policy, i.e.,
      With probability $1-\epsilon$, $A_t = \text{argmax}_{a \in \mathcal{A}} Q(S_t, a)$
      With probability $\epsilon$, $A_t$ = Uniformaly random action in $\mathcal{A}$
      Take action $A_t$ in the environment
      The state changes from $S_t$ to $S_{t+1} \sim \mathcal{P}(\cdot \mid S_t, A_t)$
      Observe $S_{t+1}$ and $R_t$ (could be $r(S_t, A_t)$, or could be stochastic)
      Update the action-value function at state-action $(S_t, A_t)$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \gamma \max_{a'} Q(S',a) - Q(s,a)]$
```

Figure 1. Algorithm 1 Q Learning

The ϵ-greedy is a simple mechanism for managing the exploration-exploitation trade-off. It ensures that most of the time (probability $1-\epsilon$), the agent exploits its incomplete knowledge of the world by chooses the best action, which corresponds to the highest action-value, but occasionally (probability ϵ) it explores other actions.

Note that the update rule in Q-learning does not mention the policy anywhere. The only thing the policy is used for is to determine which states are visited. This means we can follow whatever policy we want (e.g., ϵ-greedy), and it still converges to the optimal Q-function. This is an off-policy algorithm different from the on-policy algorithm policy gradient that we will focus on later.

C. *Deep Q-Network*

Minih et al. [1] presented a variation of the classic Q-Learning algorithm, which is Deep Q-Network (DQN), with the following contributions: (1) using a deep convolutional network to approximate the optimal action-value function. (2) using the experience replay mechanism to randomize over the data can remove the correlations in the observations and smooth over the changes in the data distribution. (3) only periodically updating the action-values towards the target value.

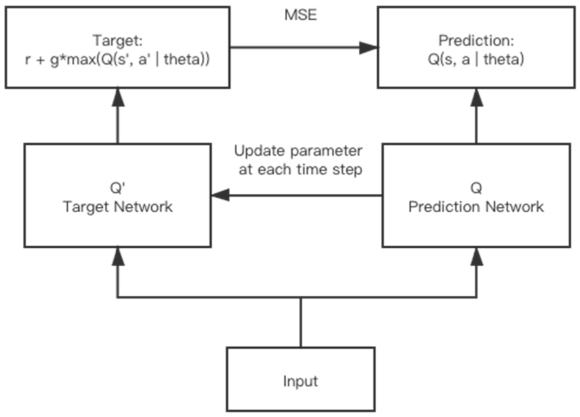

Figure 2. Deep Q Network

DQN keeps track of a number of recent experiences in terms of taking action a at state s to state s' and receiving reward r. The agent randomly samples a mini-batch from the experiences stored before to update the action-value function for each period. This experience-based update rule is called the experience replay mechanism. DQN uses this mechanism to help decorrelate the samples from the environment.

In order to get a more stable training target, DQN uses the stale network parameters when estimating the value function for the next state in an experience. And it only updates the stale network parameters for each interval of discrete many steps.

```
**Algorithm 2** Deep Q Network with Experience Replay
Initialize array M for storing experience
Initialize action-value function $Q$ with weights $\theta$
Initialize target action-value function $Q'$ with weights $\theta$
**for** epoch = 1,2,...,N **do**
    define sequence $s = \{x_1\}$
    **for** time = 1,2,...,T **do**
        $\epsilon$-greedy chooses action $a_t$
        Take action $a_t$
        Store experience $(s_t, a_t, r_t, s_{t+1})$ in M
        Sample random experience $(s, a, r, s')$ from M
        $y = r + \gamma \max_{a'} Q'(s', a \mid \theta')$
        Update Q using gradient descent
        Reset Q' = Q for each constant number of steps
```

Figure 3. Deep Q Network with Experience Replay

D. *Deep Deterministic Policy Gradient*

DQN makes it possible to create agents capable of learning to master a diverse array of challenging tasks. However, while DQN can solve problems with high-dimensional state spaces, it can only handle discrete and low-dimensional action spaces. DQN can not be applied to continuous domains since it requires optimizing the action to minimize the action-value function. It must include an iterative process to find the optimal action at each step in a continuous valued case.

Algorithm 3 DDPG algorithm

```
Randomly initialize critic network $Q(s, a \mid \theta^Q)$ and actor $\mu(s \mid \theta^Q)$ with weights $\theta^Q$ and $\theta^\mu$
Initialize replay buffer M and target network $Q', \mu'$ with weights $\theta^Q$, $\theta^\mu$
for epoch = 1,2,...,N do
    initialize a random process $\mathcal{N}$ for action exploration
    receive initial observation state $s_1$
    for t = 1,2,...,T do
        Select action $a_t = \mu(s_t \mid \theta^Q) + \mathcal{N}_t$ 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 M
        Sample a random minibatch transitions $(s_i, a_i, r_i, s_{i+1})$ from M
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$
        Update critic by minimizing the loss $L = \frac{1}{\#batch} \sum (y_i - Q(s_i, a_i \mid \theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

            $\frac{1}{\#batch} \sum_i \nabla_a Q(s, a \mid \theta^Q) \mid_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}(s \mid \theta^\mu) \mid_{s_i}$

        Update the target networks:
        $\theta^{Q'} = \gamma\theta^Q + (1-\gamma)\theta^{Q'}$
        $\theta^{\mu'} = \gamma\theta^\mu + (1-\gamma)\theta^{\mu'}$
```

Figure 4. DDPG Algorithm

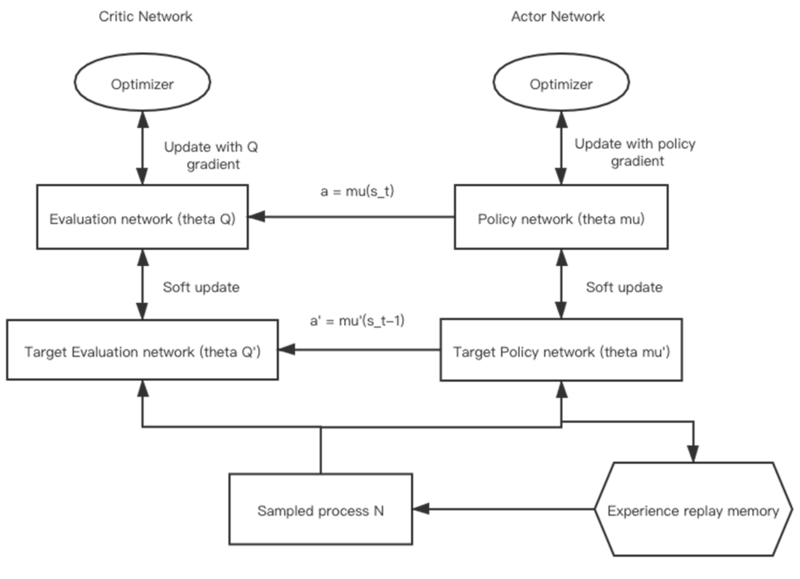

Figure 5. Deep Deterministic Policy Gradient

Lillicrap et al. [2] adapted the ideas from DQN to the continuous action domain. They presented an actor-critic model-free model algorithm Deep Deterministic Policy Gradient (DDPG), that can operate over the continuous action space. The algorithm is based on the deterministic policy gradient (DPG) algorithm [2]. DPG has some limitations that applying neural function approximators to this actor-critic method is unstable in many cases. DDPG adapts two ideas from the innovation of DQN that it can learn value functions using large, non-linear function approximators stably and robustly. First, the network is trained off-policy with samples using an experience replay mechanism to decorrelate the observations. Second, the network is trained with a target Q network to give consistent targets during temporal difference backups. One advantage of DDPG is that it requires only a straightforward actor-critic architecture and learning algorithm with very few moving parts, easy to implement and scale to large challenging problems.

Using the same action-value function and the Bellman equation as before, DDPG considers function approximators parameterized by θ^Q, and optimizes it by minimizing the loss:

$$L(\theta^Q) = E_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\left[\left(Q(s_t, a_t \mid \theta^Q) - y_t\right)^2\right]$$

where $y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) \mid \theta^Q)$. (4)

III. Discussion

In this section, I will review several interesting applications of DDPG.

A. DDPG for urban traffic light control

Casas [3] presented an application of DDPG for urban traffic light control. One of the key issues with traffic light timing optimization is the large scale of the input information available for the controlling agent, namely all the traffic data that is continually sampled by the traffic detectors that cover the urban network. In order to overcome the large scale of the available state information, deep reinforcement learning in the form of a DDPG algorithm is applied. Specifically, they make use of a multilayer perceptron type of architecture, both for the actor and the critic networks. The actor is designed so that the modifications to the traffic light timings keep the cycle duration. To improve convergence, they make use of a replay memory gradient norm clipping and a schedule for the discount rate γ. The input state used to feed the network consists of traffic detector information, namely vehicle counts and average speeds combined in a single speed score. The rewards used as reinforcement signals are the improvements over the measurements without any control action being performed. As a result, comparing DDPG with the classic Q-learning, both reach the same reward level. However, DDPG remains remarkably stable once it reached the peak performance.

B. DDPG for bipedal walking robot

Kumar et al. [4] presented a bipedal walking robot using DDPG. The robot demonstrates successful walking behavior by learning through several trials and errors, without any prior knowledge of itself or the world dynamics. After training the model in simulation, it was achieved faster walking or even rendered a running gait with an average speed of 0.83 m/s. The gait pattern of the bipedal walker was compared with the actual human walking pattern. The results show that the bipedal walking pattern had similar characteristics to a human walking pattern [5]. Deep reinforcement learning can be used as a convenient method to learn complex controls without prior knowledge of the dynamics of the agent or the environment. The same was demonstrated by the simulation of planar bipedal walker in real-world physics engine Gazebo.

IV. Conclusion

We have reviewed the major developments of Deep Deterministic Policy Gradient and some of the useful applications. We can see that DDPG has a lot of significant advantages. When the complexity of the network increases, the classic Q-learning can no longer scale, while DDPG can still improve the obtained rewards consistently. DDPG is able to better scale to large networks than the classic tabular approaches like Q-learning. There are also a few limitations that require consideration. One of them is that DDPG usually takes many training steps to find solutions like most of the model-free reinforcement learning approaches, which might lead to large time complexity. This requires further research. But in general, due to its capacity to handle more difficult problems with continuous action spaces, DDPG still has a promising future with potential applications in many areas.

References

[1] Minih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, NicolasHeess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015

[3] N. Casas, Deep deterministic policy gradient for urban traffic light control, *arXiv preprint arXiv:1703.09035*, March 2017.

[4] Arun Kumar, Navneet Paul, and SN Omkar. Bipedal walking robot using deep deterministic policy gradient. *arXiv:1807.05924*, 2018.

[5] Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning That Matters. *Proceedings of the AAAI Conference on Artificial Intelligence 32 (1),* 2018.

[6] Wawrzy´ nski, Pawe_. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks, 22(10):1484–1497,* 2009.

[7] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602,* 2013.

[8] Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. *Proceedings of the International Conference on Machine Learning (ICML),* 2014.

[9] Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. *Proceedings of the International Conference on Machine Learning (ICML),* 2016.

[10] Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 2000.