Deep Q-Networks

Sunmook Choi `felixchoi@korea.ac.kr`

September 8, 2023

1 Introduction

Deep reinforcement learning is a combination of reinforcement learning and deep learning. It involves using neural networks to approximate the value function or policy in a reinforcement learning problem. Deep Q-learning have started the realm of deep reinforcement learning, and it is realized by training Deep Q-Networks (DQN). DQN is trained in a way of approximating the optimal action-value function by a neural network.

In Section 2, we describe DQN and some ways to overcome possible challenges while training. The training algorithm is based on the original Q-learning with some modifications due to such challenges. In Section 3, we describe some variants of DQN to alleviate some possible problems from the original DQN.

2 Deep Q-Networks

Basically, the algorithm of training DQN is based on Q-learning in RL. Recall the update equation of Q-learning.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.1}$$

The update equation of Q-learning implies that the algorithm is aimed to learn the optimal action-value function motivated by the Bellman optimality equation:

$$q_*(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \max_a q_*(S_{t+1}, a) \mid S_t = s, A_t = a\right] \tag{2.2}$$

One way to find the optimal action-value function is to approximate the action-value function by a neural network, denoted by Q_θ. The network is referred to *Deep Q-Networks* (DQN). A natural way to learn the function Q_θ is first run an episode of a game, obtain a trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T)$, and then minimize the following loss $L(\theta)$ using gradient descent algorithms:

$$L(\theta) = \frac{1}{T} \sum_{t=0}^{T-1} \Big[r_{t+1} + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t) \Big]^2 \tag{2.3}$$

However, this optimization encounters some challenges from a deep learning perspective. Most deep learning algorithms assume the data samples to be independent. However, it is impossible in reinforcement learning because an environment is assumed to have Markov property. That is, the samples $(s_t, a_t, r_{t+1}, s_{t+1})$ in the trajectory are highly correlated, which makes the optimization of Q_θ difficult. Furthermore, the data distribution is assumed to be fixed in deep learning, while the data distribution changes as the algorithm learns new behaviors in reinforcement learning. This discrepancy can be problematic when the original deep learning algorithms are applied to reinforcement learning algorithms.

To alleviate the problems of correlated data and non-stationary distributions, the authors of the paper [2, 3] utilized the technique known as *experience replay*. Moreover, another non-trainable network, called the *target network* is used to improve the stability of optimization. We will discuss these methods in the following subsections.

2.1 Experience Replay

An experience replay mechanism is employed to train the function Q_θ mitigating high temporal correlation between samples and non-stationary data distribution problems. We first introduce a *replay memory* (also known as a *replay buffer*) $\mathcal{D}$ to store the agent's experiences $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ with a fixed capacity N. We notice that the experiences are sampled not only from one episode but also from many episodes that the agent experiences. Then, the network Q_θ is trained from the samples $e_j \sim \mathcal{D}$ that are uniformly sampled from the replay memory $\mathcal{D}$. Let $\mathcal{B}$ be the collected samples. Then the objective becomes the follows:

$$L(\theta) = \frac{1}{|\mathcal{B}|} \sum_{e_j \in \mathcal{B}} \Big[r_{j+1} + \gamma \max_a Q_\theta(s_{j+1}, a) - Q_\theta(s_j, a_j) \Big]^2 . \tag{2.4}$$

There are several advantages to apply the mechanism in a deep learning perspective. First, each step of experience $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ is potentially used for multiple times in weight updates. Moreover, this would let the network see rare experiences repeatedly, showing better data efficiency. Second, learning directly from consecutive samples in the same episode is inefficient. It is because there are strong temporal correlations between the samples, breaking the independency assumption on samples in deep learning. Hence, sampling randomized samples from the replay memory will break these correlations, making the weight update more efficient.

The authors [2, 3] mentions that the uniform sampling method from the replay memory might have some limitations. First, uniform sampling does not differentiate important transitions and gives equal importance to them. Also, samples in the replay memory are always overwrited with recent transitions in a finite capacity. The paper [6] introduces new type of experience replay mechanism that considers priorities on transition samples in the replay memory.

2.2 Target Network

From Eq. 2.4, notice that the target of $Q_\theta(s_i, a_i)$ is $r_{i+1} + \gamma \max_a Q_\theta(s_{i+1}, a)$. However, since the target of our training object contains the same parameter θ, the target also changes while training, which is referred to *non-stationary target problem*. This problem would make the training unstable, and we need to freeze the network in the target. Hence, the authors [3] introduce another network called the *target network*. This network, denoted by $\hat{Q}$, is a separate network which is used only for generating the targets. The target network $\hat{Q}$ has the same architecture as the agent's network Q_θ. To freeze the target network, the parameters, say $\hat{\theta}$ in $\hat{Q}$ is fixed during training. Instead, every C updates, we copy the parameters θ from Q_θ and change the target networks parameter $\hat{\theta}$ into θ. With the target network $\hat{Q}$, the final objective will be the following:

$$L(\theta) = \frac{1}{|\mathcal{B}|} \sum_{e_j \in \mathcal{B}} \left[r_{j+1} + \gamma \max_a \hat{Q}_{\hat{\theta}}(s_{j+1}, a) - Q_\theta(s_j, a_j) \right]^2 . \tag{2.5}$$

2.3 Deep Q-learning Algorithm

With the utilization of experience replay and the target network, we train the network Q_θ based on Q-learning and gradient descent algorithm. We call this algorithm *Deep Q-learning*, presented in Algorithm 2.1.

3 DQN variants

3.1 Double DQN

3.2 Dueling DQN

3.3 Prioritized Experience Replay

References

[1] Andrew Barto and Richard S. Sutton, *Reinforcement Learning: An Introduction* (2nd ed.), The MIT Press, 2018.

[2] V.Mnih, et. al, *Playing Atari with Deep Reinforcement Learning*, arXiv:1312.5602, 2013

Algorithm 2.1 Deep Q-learning

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target network $\hat{Q}$ with $\hat{\theta} = \theta$
Initialize step size $\eta$
 1: for episode= $1, \ldots, M$ do
 2:     Start an episode at a state $s_0$
 3:     for $t = 0, \ldots, T-1$ do
 4:         With probability $\epsilon$, select a random action $a_t$, otherwise select
                $a_t = \max_a Q_\theta(s_{t+1}, a)$
 5:         Execute an action $a_t$ and observe reward $r_{t+1}$ and the next state $s_{t+1}$
 6:         Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
 7:         Sample random minibatch $\mathcal{B}$ of transitions $e_j = (s_j, a_j, r_{j+1}, s_{j+1})$ from $\mathcal{D}$
 8:         Set $y_j = \begin{cases} r_{j+1} & \text{for terminal } s_{j+1} \\ r_{j+1} + \gamma \max_a \hat{Q}_{\hat{\theta}}(s_{j+1}, a) & \text{for non-terminal } s_{j+1} \end{cases}$
 9:         Compute the loss: $L(\theta) = \frac{1}{2|\mathcal{B}|} \sum_{e_j \in \mathcal{B}} [y_j - Q_\theta(s_j, a_j)]^2$
10:         Compute the gradient: $\nabla_\theta L(\theta) = -\frac{1}{|\mathcal{B}|} \sum_{e_j \in \mathcal{B}} [y_j - Q_\theta(s_j, a_j)] \nabla_\theta Q_\theta(s_j, a_j)$
11:         Perform a gradient descent step: $\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$
12:         Every $C$ steps, update the target network: $\hat{\theta} = \theta$
13:     end for
14: end for
```

[3] V. Mnih, et. al, *Human-level control through deep reinforcement learning*, Nature 518, 529-533, 2015.

[4] Hado van Hasselt, Arthur Guez, and David Silver, *Deep Reinforcement Learning with Double Q-learning*, AAAI 2016

[5] Z. Wang, et. al, *Dueling Network Architectures for Deep Reinforcement Learning*, ICML 2016

[6] T. Schaul, et. al, *Prioritized Experience Replay*, ICLR 2016