DEEP Q-NETWORK

Sara Silvestrelli
Dipartimento di Economia, Metodi Quantitativi e Strategie di Impresa
Università degli studi di Milano-Bicocca

ABSTRACT

One goal of AI is to produce autonomous agents able to interact with the environment and learn optimal behaviours through a trial and error mechanism. RL is one of the well-founded frameworks for experience-driven learning. Deep Learning accelerated progress in RL defining the field of *deep reinforcement learning* (DRL) that has already been applied to many different problems. DRL approaches in particular have been very successful in high dimensional problems. The algorithm, known as *Deep Q-Network*, was developed by enhancing a classic Reinforcement Learning algorithm called Q-Learning with Deep Neural Networks and a technique called *replay experience*.

***K*eywords** Artificial Intelligence · Deep Reinforcement Learning · DQN

1 Introduction

Deep Reinforcement Learning is a ML technique that combines Deep Learning and Reinforcement Learning. In general, a deep neural network is trained to approximate the optimal policy and/or optimal value function. Most DRL approaches work on discrete actions, but there are also some solutions for continuous control problems [1].

In a single-agent environment a valid solution for complex learning problems with large state/actions spaces are deep Q-Learning algorithms. It is one of the most popular approach that uses some form of function approximation for the action-value. The basic step for DQL is that the initial state is fed into the neural network and it outputs the Q-value of all possible actions. The DNN that approximates a Q-function is called a deep Q-Network, or DQN.

2 Framework

Deep Q-Network [2] represents an off-policy model-free algorithm which works with Q-value operator and that can handle both a discrete and a continuous action space.

Q Learning technique builds the Q table containing the Q values of all state-action pairs using Q value iterations. Q Learning by storing each state-action pair requires less operations and memory space. Thus, it is a method works properly in low-dimensional environments. It loses its effectiveness as the number of states or actions increases.

DQN compensates for the lack of generalization of Q because, when it is considered a large state-action space, the Q^* estimates are often implemented with approximation functions. Neural networks are known to be very efficient in approximating functions and can manage well continuous spaces. The value function is parameterized by a vector of parameters $\theta = (\theta_1, \theta_2, ..., \theta_n)^T$.

The function approximator can be thought of as a mapping from a vector $\theta \in \mathbb{R}^n$ to the space of the value function. The main difference between Q-learning and DQN is the agent's "brain": in Q-learning it is the Q-table, in DQN it is a DNN (Figure 1). DQN leads to a more stabilized training of action-value approximation and target increasing generalization power.

Furthermore, RL is unstable due to correlations in the sequence of observations and the small updates to Q values that can lead to huge change in the agent's policy. *Experience replay* and *iterative update* of Q values are two strategies

[1]Image source DOI:10.21608/mjeer.2020.22756.1003.

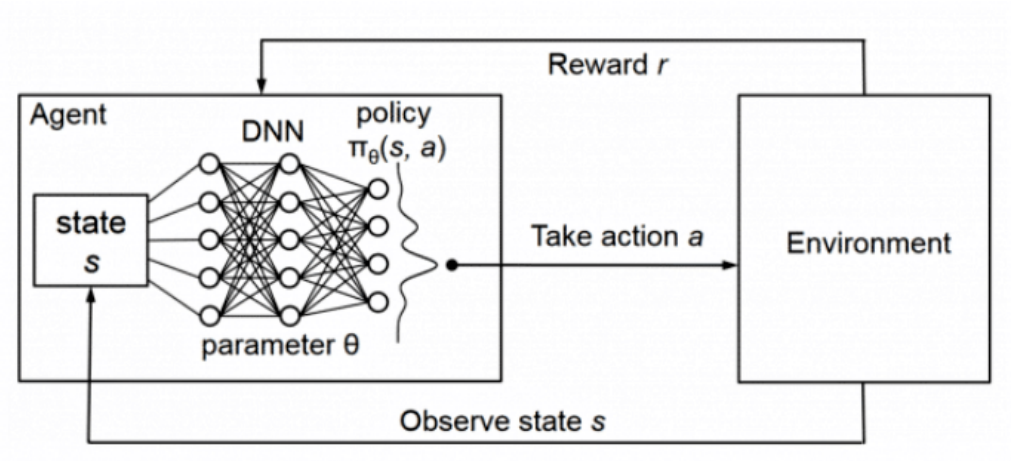

Figure 1: Deep Q-Network framework[1].

introduced to solve these problems [3]. DQN architecture has two neural networks: the Q network that helps selecting the best action and the Target network that compute the target Q value of taking that action at the next state.

During the training phase we want to minimize a loss function, the Temporal Difference error function (TD function), which is the difference between the Q value of a state-action pair and its Q target.

3 Experience Replay

To make the agent learn and to train our network we use a technique that is called *Replay Memory*[2]. The idea is that the agent stores all his experiences in a buffer. At step t the experience is a tuple that contains the current state, the current action, the immediate reward and the next state of the environment:

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1}). \tag{1}$$

For each update i an experience tuple $(s, a, r, s\prime) \sim U(D)$ is sampled from replay memory. For each sample (or mini batch) the current θ parameters are updated according to gradient Descent. These batches containing either older and newer samples are inputted to both networks. The use of Replay Memory ensures that the agent sees each data multiple time before it is removed from memory. This is a very useful technique for environments where data samples are costly to collect, for example clinical data.

We use batches instead of considering consecutive sequences of experiences because otherwise they can be highly correlated with each other and this can lead to overfitting situations. Random sampling of experiences solves the problem. Experience replay also increases learning speed with mini-batches.

4 Target Network

The goal is minimizing the distance between the Q-target and $Q(s, a)$ by way of usual gradient descent algorithms. The Q target is unknown, so we use the Bellman Optimality equation to define it at each iteration i as follows [4]:

$$Q_{target} = r_{t+1} + \gamma \, max_{a\prime} \, Q(s\prime, a\prime; \theta_i^{\,-}) \tag{2}$$

where $\theta_i^{\,-}$ are only updated every $C \in \mathbb{N}$ time iterations with the Q network parameters assigning $\theta_i^{\,-} = \theta_{i-1}$ and maintaining fixed the target value with the original policy network's weights for C iterations.

The goal is optimizing the following square loss of the predicted Q-value and the target Q-value:

$$L_i(\theta_i) = E\big[\big(r_{t+1} + \gamma \, max_{a\prime} \, Q(s\prime, a\prime; \theta_i^{\,-}) - Q(s, a; \theta_i)\big)^2\big] \tag{3}$$

where $\theta_i^{\,-}$ are the parameters used to calculate the target network at iteration i.

[2]Also called 'Replay Buffer', 'Experience Replay'.

$Q(s, a; \theta)$ is cloned periodically to a separate target $\hat{Q}(s, a; \theta^-)$ employing a second network that doesn't get trained and ensures that the Target Q values remain stable for a short period.

5 Training algorithm

Once introduced the basic concepts behind DQN, the single steps of this approach are now explained.

The final procedure is summarized in Algorithm 1 adapted from Mnih et al. (2015) [5]. First it is initialized the replay memory, it can be done by executing random actions for a few time-steps. Are then created two instances of the DQN Network, the online and the target network with the weights of the target network set to be the same as that of the online network. At each episode (or patients in this case) is gathered a starting observation s and starts here a loop that tracks the total number of steps that need to be performed until terminal state is reached.

At each time-step an ϵ-greedy strategy is used to determine whether to perform random actions or act as the network suggests. The selected action is executed to obtain the immediate reward r_{t+1} and the next state s_{t+1}. The tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ is stored in the replay memory as a sample of training data and a mini-batch of random samples from the replay memory is drawn.

The online network takes as input the current state and action and predicts the Q value obtaining the *predicted Q value*. The Target network takes as input the next state and predicts the best Q value out of all actions that can be taken from that state obtaining the *target Q value*. The target Q-value is computed using the target network by taking into account if the current state is a terminal one, otherwise the outcome is set to be equal only to the reward.

The loss between the target and predicted Q values is used to train and update only the weights of the online network. At regular intervals, after $C \in \mathbb{N}$ steps the online network weights are copied to the target network.

Algorithm 1: DQN with experience replay

Initialization: replay memory $\mathcal{D}$ to capacity N , action-value function Q with random weights θ, target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

for episode=1,M **do**:
  Initialize state s_t
  for t=1,T **do**:
    With probability ϵ select a random action a_t
    otherwise select $a_t = max_a\, Q^*(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_{t+1} and state s_{t+1}
    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
    Set $s_{t+1} = s_t$

    Sample random minibatch of transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ from $\mathcal{D}$
    Set

$$y_j = \begin{cases} r_{j+1} & \text{for terminal state} \\ r_{j+1} + \gamma\, max_{a\prime}\, \hat{Q}(s_{j+1}, a\prime; \theta^-) & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$

    Every *C* steps reset $\hat{Q} = Q$ i.e., set $\theta^- = \theta$
  end for
end for

References

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, Nov 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[3] Yu Ding, Liang Ma, Jian Ma, Mingliang Suo, Laifa Tao, Yujie Cheng, and Chen Lu. Intelligent fault diagnosis for rotating machinery using deep q-network based health state classification: A deep reinforcement learning approach. *Advanced Engineering Informatics*, 42:100977, 10 2019.

[4] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4), 2018.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.