Deep Reinforcement Learning

Deep Q-Networks

Jordi Casas Roma

jordi.casas.roma@uab.cat

October 17, 2025

Universitat Autònoma de Barcelona

Table of Contents

Introduction
Preliminaries

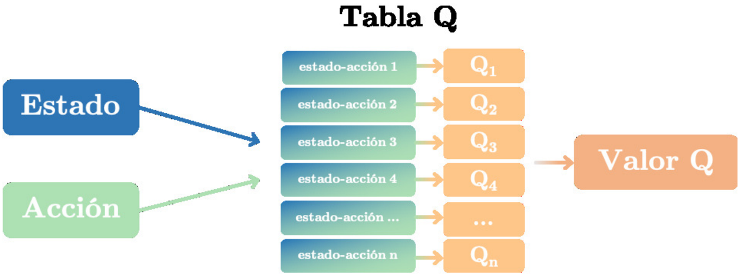
Previously... **Q-learning**:

$$\mathbf{Q} = \begin{array}{c} \begin{array}{ccccc} a_0 & a_1 & a_2 & a_3 & a_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 17 & 0 & 74 \\ 12 & 0 & 80 & 100 & 0 \\ 21 & 0 & 0 & 74 & 62 \\ 5 & 0 & 70 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array}$$

- The goal is to create a Q-table with $q(s, a)$ values for all possible pairs of states (s) and actions (a).

Introduction
Preliminaries

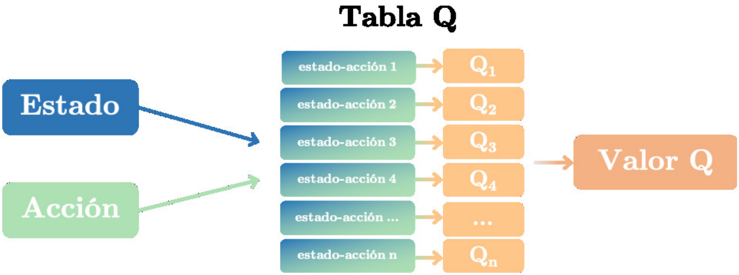
Previously... **Q-learning**:



- Then, we choose the action that gives us a maximum reward.

Introduction
Preliminaries

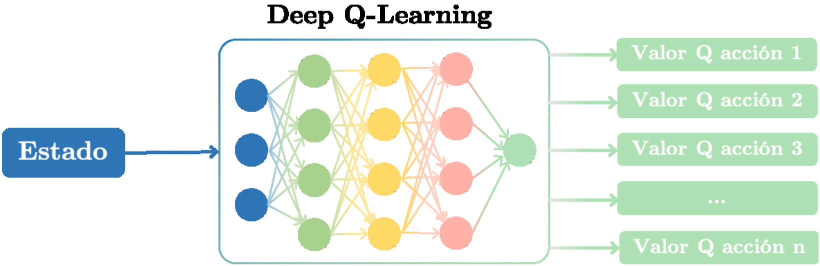
Previously... **Q-learning**:



- Then, we choose the action that gives us a maximum reward.
- When there are a very large number of action-state pairs, the *Q-learning* method does not work because it is technically impossible to store all the possible values in a Q-table.

Introduction
Preliminaries

Deep Q-learning:



- However, we can map state-action pairs to a value through non-linear functions.
- The most common option is using a neural network, that is, using deep learning techniques to represent the Q table.

Introduction

Preliminaries

Deep Q-learning:

- This combination of *Q-learning* with deep learning is what is known as *Deep Q-Network* (DQN).
- And the learning algorithm to approximate the function $Q(s, a)$ with a DQN is called, analogously, *Deep Q-Learning* (DQL).
- This is one of the most powerful value-based methods used within DRL.
- The DeepMind team was the first to propose combining convolutional neural networks with reinforcement learning (Mnih et al., 2013)[1], introducing DQNs for the first time.

[1] V. Mnih, K. Kavukcuoglu, D. Silver et al. (2013). *Playing Atari with Deep Reinforcement Learning*. NIPS Deep Learning Workshop

Table of Contents

DQN Architecture
Basic DQN

Algorithm 1 Basic DQN

Initialize network Q
while not converged **do**
    Set state s
    Choose action $a = \max_{a' \in \mathcal{A}} Q(s, a')$
    Agent takes action a, observe reward r and next state s'
    if episode ended **then**
        $y = r$
    else
        $y = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$
    end if
    Compute Loss function: $\mathcal{L} = [Q(s, a) - y]^2$
    Update $Q(s, a)$ with *backpropagation* and gradient descent
end while
return Q

DQN Architecture
Basic DQN

Drawbacks and problems: After reviewing the previous pseudo-code, what are the main issues or limitations in this algorithm?

DQN Architecture
Basic DQN

Drawbacks and problems: After reviewing the previous pseudo-code, what are the main issues or limitations in this algorithm?

1. The policy used does not include exploration-exploitation, so learning is slower and can lead to suboptimal solutions.

DQN Architecture
Basic DQN

Drawbacks and problems: After reviewing the previous pseudo-code, what are the main issues or limitations in this algorithm?

1. The policy used does not include exploration-exploitation, so learning is slower and can lead to suboptimal solutions.
2. It does not take into account that in DRL the data are not independently and identically distributed (i.i.d.) as required by the SGD algorithm, which in turn implies a high correlation between states.

Table of Contents

Table of Contents

Improvements
ϵ-Greedy Method

Drawbacks and problems:

As previously stated...

1. The policy used does not include exploration-exploitation, so learning is slower and can lead to suboptimal solutions.

Improvements

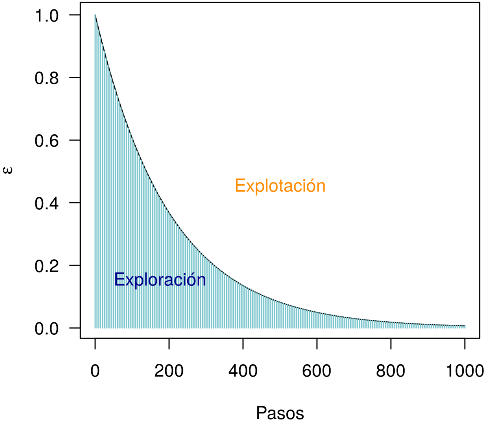
ϵ-Greedy Method

ϵ-**Greedy Method**:

1. The **method ϵ-greedy** allows you to consider these two needs of the agent:
   - explore randomly at the beginning, when you still don't have enough information (Q's approximation is bad);
   - and use the Q approximation (without randomness) to decide actions when learning is in a more advanced state.
2. The method introduces a probability parameter ϵ, which indicates when to go from a random policy to a Q policy.
   - When the value is 1, all actions taken are random.
   - This probability is reduced during the training, so the agent will take more actions in accordance with Q policy.

Improvements
ϵ-Greedy Method

Effect of varying ϵ on exploration-exploitation



Improvements
ϵ-Greedy Method

ϵ-**Greedy Method** parameters:

- There are several implementations...
  - Linear
  - Exponential
- But, usually, we need to define:
  - Initial ϵ value
  - Decay factor
  - Minimum ϵ value

Improvements
ϵ-Greedy Method

ϵ-**Greedy Method** example:

```
EPS_START = 1.0
EPS_DECAY = 0.999985
EPS_MIN = 0.2

def epsilon_decay(epsilon, decay, minimum):
    return max(epsilon * decay, minimum)
```

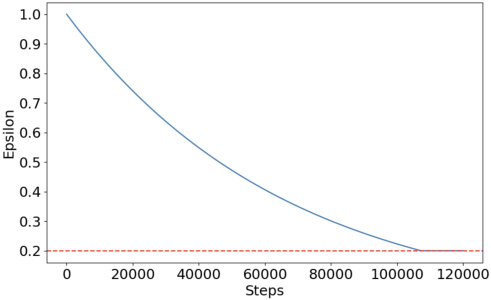


Table of Contents

Improvements

Experience Replay buffer

Drawbacks and problems:

- Data is not independent because we are entering it sequentially.
  - Even if we stored a quantity of data prior to the current state, they would be closely related to each other.
- Data does not have an identical distribution to the examples provided by the policy we hope to learn.
  - We collect data either by the current policy, or randomly, or both at the same time (ϵ-*greedy*), so it will have nothing to do with its distribution according to the final policy.

Improvements
Experience Replay buffer

Experience Replay buffer:

1. Store a certain amount of experiences while the agent is experimenting, so that it can learn from recent experiences.
2. Randomly select a subset of this stored data for the neural network in order to reduce the correlation between them.

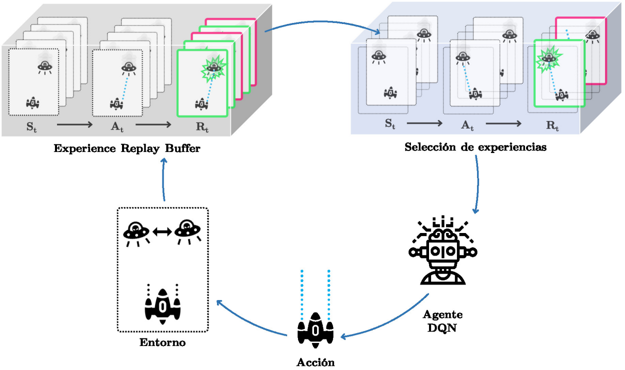
Improvements
Experience Replay buffer

Experience Replay buffer:

- This technique is known as **experience replay buffer** (also called *replay buffer* or *experience replay*).
- Past experiences are saved in a fixed-size *buffer*.
  - Because the buffer has a fixed size, older experiences will be removed from the buffer as new experiences arrive.
- For **training**, we extract a random subset of these experiences to feed the neural network.

Improvements
Experience Replay buffer

- Experiences (state, action, reward and new state) will be stored in the *replay buffer*.
- The NN will select a random subset of these experiences to train and improve learning.



Improvements
Experience Replay buffer

The **experience replay buffer** will allow us to:

- Have training data that is more independent of each other (a random selection is made of the data stored in the *buffer*, thus breaking the temporal correlation).
- Have sufficiently recent data so that they are almost identically distributed (the associated policy in the current state will be more similar to the final policy as the end of the process is reached).

Table of Contents

Improvements
Target Network

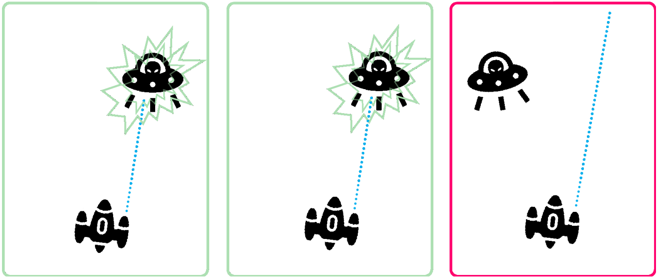
Drawbacks and problems:

- The correlation of the data means that each action directly affects the next state (because the data is not *i.i.d.*).
- The vectors (s, a, r, s') of one state and the next one will be very similar, almost indistinguishable for the neural network.
- This can lead to **very unstable training**.
  - We will be forcing our agent to take actions similar to those it did in the previous state, regardless of the new situation.

Improvements

Target Network

- The alien appears on the right and the agent fires with the right gun, hitting.
- The problem is that if this happens in successive states, the agent does not learn when to shoot to the left, since the value of Q for shooting to the right will always be much higher!



Improvements
Target Network

Target Network:

- To solve this problem, we can introduce a second neural network $\hat{Q}$ to improve learning.
- Instead of obtaining the target value $Q(s', a')$ and the predicted value $Q(s, a)$ with the same neural network, we will calculate the target value with this second neural network which we will call *target network*.
  - We use Q (**primary** network) to obtain the **predicted** value $Q(s, a)$.
  - We use $\hat{Q}$ (**target** network) to obtain the **target** value $Q(s', a')$.

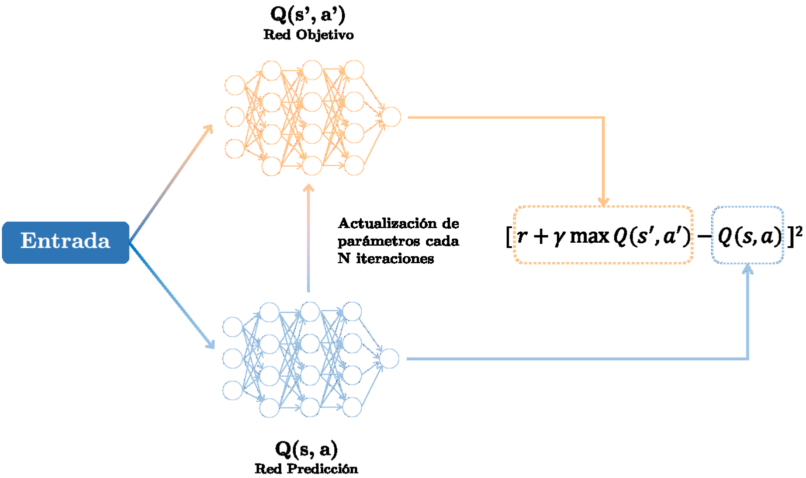
Improvements
Target Network

Target Network:

- This second network $\hat{Q}$ will be a copy of the main one, but with fixed weights.
- Every a certain number of iterations N (between 1,000 and 10,000, generally) the coefficients (weights and biases) of the main network (prediction network) are copied to the target network.
  - The target network explores the state space with those values for a while.
- Informally, what we will be doing is "stopping the game" in this second neural network to extract more information from the state space.
- The experiences learned in this target network can also be stored in the *replay buffer* to be able to use them in the learning process.

Improvements

Target Network



Improvements

Target Network

Target Network:

- The target value $Q(s', a')$ (fixed for a certain time, N iterations) needed in the Bellman equation will now be provided by the target network $\hat{Q}$, and we will calculate the loss or error for each action with this value.
- After N iterations, it will synchronize with the main network and will receive new coefficients (weights and biases) with which to explore the state space again.
- With this mechanism, it is possible to stabilize the training and better generalize the patterns in the data, avoiding focusing too much on a region of the state space, which causes, as we have seen, a stagnation of the agent that ends up always repeating the same action.

Table of Contents

DQN Final Architecture I
Final DQN

Algorithm 2: DQN with ϵ-greedy, experience replay and target network

Parameter $\mathcal{C}$ (usually in range 1.000-10.000)
Initialize network Q
Initialize network $\hat{Q}$
Initialize experience replay memory D
Initialize the *agent* to interact with the *environment*

while not converged **do**
    /* SAMPLING PHASE */
    $\epsilon \leftarrow$ Setting new epsilon with ϵ-decay
    Choose action a from state s using policy ϵ-greedy(Q)
    Agent takes action a in state s, and get reward r and next state s'
    Store transition $(s, a, r, s', done)$ in experience replay memory D
    if enough experiences in D **then**
        /* LEARNING PHASE */
        Sample a random *batch* of N transitions from D
        for all transition $(s_i, a_i, r_i, s'_i, done_i)$ in *batch* **do**
            Compute $Q(s_i, a_i)$
            Compute $\max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$

DQN Final Architecture II
Final DQN

if $done_i$ **then**
$y_i = r_i$
else
$y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
end if
Compute Loss function: $\mathcal{L} = \frac{1}{N} \sum_{i=0}^{N-1} [Q(s_i, a_i) - y_i]^2$
Update Q using the SGD algorithm by minimizing the loss $\mathcal{L}$
Every $\mathcal{C}$ steps, copy parameters from Q to $\hat{Q}$
end for
end if
end while
return Q

DQN Final

Implementation details and parameters:

1. V. Mnih, K. Kavukcuoglu, D. Silver, et al. (2013). "*Playing Atari with Deep Reinforcement Learning*". NIPS Deep Learning Workshop, preprint `https://doi.org/10.48550/arXiv.1312.5602`.
   - First DQN + ϵ-greedy method + experience replay

2. V. Mnih, K. Kavukcuoglu, D. Silver, et al. (2015) "*Human-level control through deep reinforcement learning*". Nature 518, 529-533. `https://doi.org/10.1038/nature14236`
   - Extensions: multistep learning, Prioritized Experience Replay, Double Q-Network, Duelling Q-Network, etc.

Table of Contents

Bibliography
References

Some relevant references: (1/2)

1. **R. S. Sutton, A. G. Barto**. (2018). *Reinforcement Learning: An Introduction (Second edition)*. MIT Press, Cambridge, MA.
2. **M. Lapan**. (2024). *Deep Reinforcement Learning Hands-On (Third edition)*. Packt Publishing.
3. **V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller** (2013). *Playing Atari with Deep Reinforcement Learning*. https://doi.org/10.48550/arXiv.1312.5602
4. **V. Mnih, K. Kavukcuoglu, D. Silver, et al.** (2015) *Human-level control through deep reinforcement learning*. Nature 518, 529-533.
5. **J. F. Hernández-García, and R. S. Sutton** (2019). *Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target*. University of Alberta.
6. **H. Hasselt, A. Guez, D. Silver** (2015). *Deep Reinforcement Learning with Double Q-learning. Google DeepMind*.

Bibliography
References

Some relevant references: (2/2)

1. **T. Schaul, J. Quan, I. Antonoglou, D. Silver** (2016). *Prioritized Experience Replay*. Google DeepMind.
2. **Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas** (2016). *Dueling Network Architectures for Deep Reinforcement Learning*. Google DeepMind.
3. **M. G. Bellemare, W. Dabney, R. Munos** (2017). *A Distributional Perspective on Reinforcement Learning*. Google DeepMind.
4. **M. Fortunato, M. G. Azar, B. Pio, et al.** (2017). *Noisy Networks for Exploration*. Google DeepMind.
5. **M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, D. Silver** (2017). *Rainbow: Combining Improvements in Deep Reinforcement Learning*. Google DeepMind.