Reinforcement Learning: Q-learning

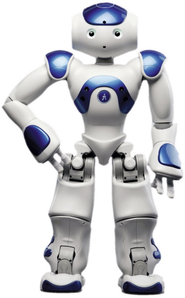
Thomas Bonald

MS Big Data
2018 – 2019

TELECOM ParisTech

Reinforcement Learning

- Learning by **trial and error**
- Inspired by the behavior of animals (including humans!)
- The **exploration-exploitation** trade-off
- Many applications: robotics, games, advertising, content recommendation, medicine, etc.



Outline

1. Markov Decision Process
2. Dynamic Programming
3. Q-learning
4. Deep Q-Network
5. Deep memory

Markov Decision Process

At time $t = 0, 1, 2, \ldots$, the agent in **state** s_t takes **action** a_t and:

- moves to some **state** s_{t+1}
- receives some **reward** r_{t+1}

The new state and the reward are **stochastic** in general.

Definition

We refer to the **transition probabilities** as:

$$p(s', r|\ s, a) = P(s_{t+1} = s', r_{t+1} = r|\ s_t = s, a_t = a)$$

These characterize the MDP.

The objective is to find some **policy** $\pi : s \mapsto a$ that maximizes the rewards $r_1, r_2, \ldots$.

Objective function

Given the rewards $r_1, r_2, \ldots$

Definition

We refer to the **discounted reward** as:

$$R = \sum_{t=1}^{T} r_t \gamma^{t-1}$$

Here T is the time horizon and $\gamma \in [0, 1]$ is the **discount factor**:

- $\gamma = 1 \longrightarrow$ cumulative reward
- $\gamma = 0 \longrightarrow$ immediate reward

Note: The time horizon can be **infinite** only if $\gamma < 1$,
or in the presence of **terminal** states

Outline

1. Markov Decision Process
2. **Dynamic Programming**
3. Q-learning
4. Deep Q-Network
5. Deep memory

Dynamic Programming

A set of techniques introduced by Richard Bellman in the 50's to solve various optimization problems, including MDP.

The **value function** is the expected reward from each state:

$$\forall s, \quad V(s) = E_\pi(R|s_0 = s)$$

This depends on the policy π.

Bellman's fixed-point equation

$$\forall s, \quad V(s) = E_\pi(r + \gamma V(s')|\ s)$$

The best value function is the solution to the fixed-point equation:

$$\forall s, \quad V^\star(s) = \max_a E(r + \gamma V^\star(s')|\ s, a)$$

Value Iteration

Algorithm

Starting from some arbitrary **value** function V,
iterate until convergence (preferably **asynchronously**):

$$\forall s, \quad V(s) \leftarrow \max_a E(r + \gamma V(s') | \ s, a)$$

- The **limit** is the unique solution to the fixed-point equation.
- Convergence is **guaranteed** whenever $\gamma < 1$.
- An **optimal policy** is then:

$$\forall s, \quad \pi(s) \leftarrow \arg\max_a E(r + \gamma V(s') | \ s, a)$$

Policy iteration

Algorithm

Starting from some arbitrary **policy** π_0, iterate until convergence:

1. **Evaluate** the policy (preferably **asynchronously**):

$$\forall s, \quad V(s) \leftarrow E_\pi(r + \gamma V(s') |\ s)$$

2. **Improve** the policy:

$$\forall s, \quad \pi(s) \leftarrow \arg\max_a E(r + \gamma V(s') |\ s, a)$$

- The **limit** of the sequence $\pi_0, \pi_1, \pi_2, \ldots$ is an optimal policy.
- Convergence is **guaranteed** whenever $\gamma < 1$.

May be much **slower** than Value Iteration!
No need for a precise evaluation of each policy...

Outline

1. Markov Decision Process
2. Dynamic Programming
3. **Q-learning**
4. Deep Q-Network
5. Deep memory

Learning

Two strategies when the model is **unknown**:

- Learn the model, then the optimal policy (e.g., using DP)
  $\longrightarrow$ **offline** learning
- Learn directly the optimal policy
  $\longrightarrow$ **online** learning

We focus on **online** learning (more popular).

Two key algorithms:

1. SARSA $\longrightarrow$ **on-policy** (training = target)
2. Q-learning $\longrightarrow$ **off-policy** (training $\neq$ target)

Proposed in the late 80's - early 90's

Action-value function

Recall that the optimal policy is characterized by the best **value** function:

$$\forall s, \quad \pi^\star(s) = \arg\max_a E(r + \gamma V^\star(s')|\ s, a)$$

Since the model is **unknown**, we need the estimate the **action-value** function:

$$Q(s, a) = E(r + \gamma V(s')|s, a)$$

An optimal policy is then:

$$\forall s, \quad \pi^\star(s) = \arg\max_a Q^\star(s, a)$$

with

$$Q^\star(s, a) = E(r + \gamma V^\star(s')|s, a) \quad V^\star(s) = \max_a Q^\star(s, a)$$

On-policy vs Off-policy

We need to estimate the **action-value** function:

$$Q(s, a) = E(r + \gamma V(s')|s, a)$$

Two strategies:

1. SARSA (State-Action-Reward-State-Action)

$$V(s') \leftarrow Q(s', \pi(s'))$$

   $\longrightarrow$ **on-policy** (training = target)

2. Q-learning

$$V(s') \leftarrow \max_a Q(s', a)$$

   $\longrightarrow$ **off-policy** (training $\neq$ target)

Exploration vs Exploitation

Which action $\pi(s)$ to select in state s?

- Pure exploitation $\longrightarrow$ **greedy**

$$\pi(s) \leftarrow \arg\max_a Q(s,a)$$

- Pure exploration $\longrightarrow$ **random**

$$\pi(s) \leftarrow \text{random}$$

- Exploration-Exploitation trade-off $\longrightarrow$ ε-**greedy**

$$\pi(s) \leftarrow \begin{cases} \text{random} & \text{with probability } \varepsilon \\ \arg\max_a Q(s,a) & \text{with probability } 1-\varepsilon \end{cases}$$

Note: The parameter ε may depend on time or on the number of visits to state s $\longrightarrow$ **adaptive greedy**

Temporal Difference

Assume you want to estimate the **empirical mean** of some data stream $x_1, x_2, \ldots$

Two strategies:

- Store the **sum**:
  For each new sample x_t,

$$S \leftarrow S + x_t \quad X \leftarrow \frac{S}{t}$$

- Update with the **temporal difference**:
  For each new sample x_t,

$$X \leftarrow X + \alpha(x_t - X) \quad \alpha = \frac{1}{t}$$

  Note: Most often, the learning rate α is fixed!

SARSA (State-Action-Reward-State-Action)

Algorithm

Parameters: γ, α, ε
Starting from some arbitrary **action-value** function Q,
iterate until convergence:

- $s \leftarrow$ random
- $a \leftarrow \pi(s)$
- while s not terminal:

$$
\begin{aligned}
&r, s' \leftarrow \text{transition}(s, a)\\
&a' \leftarrow \pi(s')\\
&Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))\\
&s, a \leftarrow s', a'
\end{aligned}
$$

with $\pi(s) \leftarrow \begin{cases} \text{random} & \text{with probability } \varepsilon \\ \arg\max_a Q(s, a) & \text{with probability } 1 - \varepsilon \end{cases}$

Q-learning

Algorithm

Parameters: γ, α, ε
Starting from some arbitrary **action-value** function Q,
iterate until convergence:

- $s \leftarrow$ random
- while s not terminal:

$$\begin{aligned} & a \leftarrow \pi(s) \\ & r, s' \leftarrow \text{transition}(s, a) \\ & Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s,a)) \\ & s \leftarrow s' \end{aligned}$$

with $\pi(s) \leftarrow \begin{cases} \text{random} & \text{with probability } \varepsilon \\ \arg\max_a Q(s,a) & \text{with probability } 1 - \varepsilon \end{cases}$

Note: Actor-critic = efficient implementation of Q-learning

SARSA vs Q-learning

Q-learning is more **aggressive** than SARSA

- Advantage: Faster convergence to the optimal policy
- Disadvantage: May lead to dangerous states!

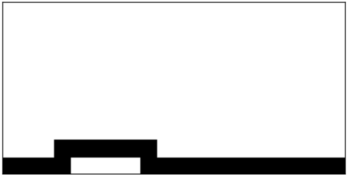


Figure: Low-altitude helicopter flight learned by Q-learning

Outline

1. Markov Decision Process
2. Dynamic Programming
3. Q-learning
4. **Deep Q-Network**
5. Deep memory

Deep Q-Network

In practice, the set of state-action pairs (s, a) may be **huge** (cf. Go, Atari), if not infinite or even continuous (robotics)
$\longrightarrow$ need for an **approximation** of the action-value function Q, applicable to unseen pairs (s, a)

Idea of DQN: Learn a **parametrized** action-value function $Q_\theta(s, a)$ with a DNN $\longrightarrow$ DeepMind paper in 2013

DQN

- Inputs: state-action pair (s, a)
- Output: action-value function $Q_\theta(s, a)$
- Loss: $(Q_\theta(s, a) - y)^2$ with $y = r + \gamma \max_{a'} Q_\theta(s', a')$

Challenge: The target y is moving!

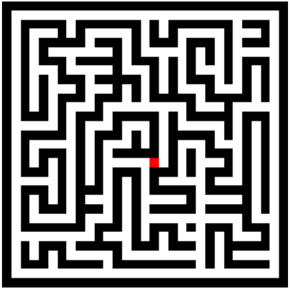
Outline

Deep memory

Assume the reward is known in the **terminal** state only:

$$s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T \rightarrow r$$

SARSA and Q-learning need a **long time** to learn the good actions in the initial states (no direct feedback)



Deep memory

Assume the reward is known in the **terminal** state only:

$$s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T \to r$$

SARSA and Q-learning need a **long time** to learn the good actions in the initial states (no direct feedback)

By **storing** the sequence and back-propagating the result,
the convergence is faster $\to$ Monte-Carlo methods
Known as the **tabular** case

Challenge: There is no way to identify good / bad actions in the sequence $a_0, a_1, \ldots, a_{T-1}$

TD(λ)-learning

Algorithm

Input: Policy π

Parameters: γ, α + trace factor $\lambda < 1$

Starting from some arbitrary **value** function V, iterate:

- $s \leftarrow$ random, $a \leftarrow \pi(s)$
- while s not terminal:

$$\begin{aligned} & r, s' \leftarrow \text{transition}(s, a) \\ & \delta \leftarrow r + \gamma V(s') - V(s) \\ & e(s) \leftarrow 1 \end{aligned}$$

  For $t = 0$ to current time:

$$V(s_t) \leftarrow V(s_t) + \alpha \delta e(s_t), \quad e(s_t) \leftarrow \lambda \gamma e(s_t)$$

  $s \leftarrow s'$

SARSA(λ)-learning

Algorithm

Parameters: γ, α, ε + trace factor $\lambda < 1$
Starting from some arbitrary Q, iterate until convergence:

- $s \leftarrow$ random, $a \leftarrow \pi(s)$
- while s not terminal:

$$
\begin{aligned}
& r, s' \leftarrow \text{transition}(s, a) \\
& a' \leftarrow \pi(s') \\
& \delta \leftarrow r + \gamma Q(s', a') - Q(s, a) \\
& e(s, a) \leftarrow 1
\end{aligned}
$$

  For $t = 0$ to current time:

$$
\begin{aligned}
& Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta e(s_t, a_t) \\
& e(s_t, a_t) \leftarrow \lambda \gamma e(s_t, a_t)
\end{aligned}
$$

  $s, a \leftarrow s', a'$

Q(λ)-learning

Algorithm

Parameters: γ, α, ε + trace factor $\lambda < 1$
Starting from some arbitrary Q, iterate until convergence:

- $s \leftarrow$ random, $a \leftarrow \pi(s)$
- while s not terminal:

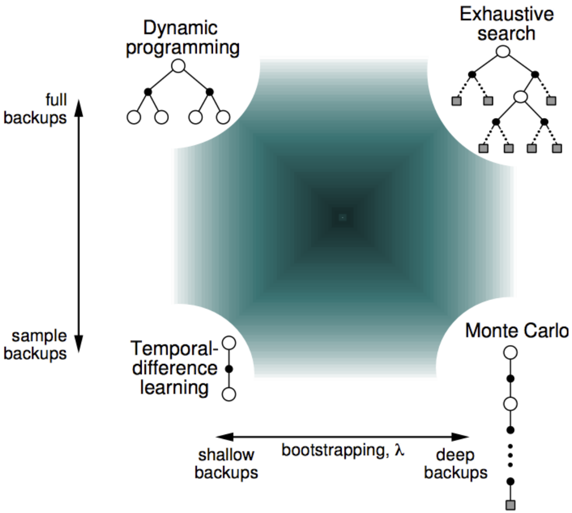
$$\begin{aligned} & r, s' \leftarrow \text{transition}(s, a) \\ & \delta \leftarrow r + \gamma \max_{a'} Q(s', a') - Q(s, a) \\ & a' \leftarrow \pi(s') \\ & e(s, a) \leftarrow 1 \end{aligned}$$

For $t = 0$ to current time:

$$\begin{aligned} & Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta e(s_t, a_t) \\ & e(s_t, a_t) \leftarrow \lambda \gamma e(s_t, a_t) \text{ if } a' \text{ maximizes } Q(s', \cdot) \text{ (0 otherwise)} \end{aligned}$$

$s, a \leftarrow s', a'$

Summary



From David Silver (DeepMind)

References

Olivier Sigaud (UPMC)
http://pages.isir.upmc.fr/~sigaud/

David Silver (DeepMind)
http://www0.cs.ucl.ac.uk/staff/d.silver/

Richard Sutton and Andrew Barto
Reinforcement Learning: An Introduction
Second edition, 2015