Advanced algorithms for learning Q-functions

Author: Gaspard Lambrechts (gaspard.lambrechts@uliege.be)
Presenter: Arthur Louette (arthur.louette@uliege.be)

March 3, 2026

Outline

Markov decision processes, policies, Q-functions and Q-learning

Approximate Q-learning

Issues with approximate Q-learning

Improvements to approximate Q-learning

Conclusions

Notations

In this course, we use the classic reinforcement learning notations:

- $s \in \mathcal{S}$ for the states, $a \in \mathcal{A}$ for the actions,
- $V(s)$ for the state value function,
- $Q(s, a)$ for the state-action value function,
- $\eta(a|h)$ for the history-dependent policy (instead of $\pi(a|h)$[1]),
- $\pi(a|s)$ for the (stationary) Markov policy,
- $\mu(s)$ for the deterministic (stationary) Markov policy,
- $\arg\max$ gives a subset or a single value depending on the context.

In addition, we use the following abbreviations:

- DP: Dynamic programming,
- SGD: Stochastic gradient descent,
- IID: Independent and identically distributed.

[1] In lecture 1 to 3.

Markov decision processes, policies, Q-functions and Q-learning

Markov decision process

An MDP is represented by its model $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, p_0, \gamma)$:

- States $s_t \in \mathcal{S}$,
- Actions $a_t \in \mathcal{A}$,
- Transition distribution $T(s_{t+1}|s_t, a_t)$,
- Reward function $r_t = R(s_t, a_t)$,
- Initial distribution $p_0(s_0)$,
- Discount factor $\gamma \in [0, 1)$.

Note: usually, we consider $0 \leq r_t \leq R_{\max}$.

In MDPs, states satisfy the Markov property:

$$\begin{aligned} p(s_{t+1}|s_0, a_0, \dots, s_t, a_t) &= p(s_{t+1}|s_t, a_t) \\ &= T(s_{t+1}|s_t, a_t). \end{aligned}$$

MDP execution

Let $h_t = (s_0, a_0, \ldots, a_{t-1}, s_t) \in \mathcal{H}$ denote the history at time step t.

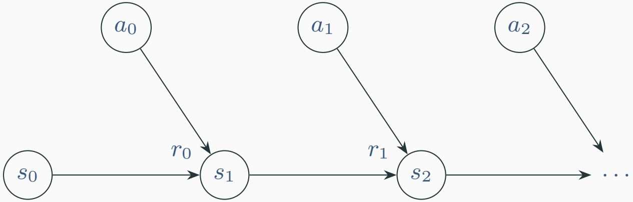


Figure 1: Bayesian graph of an MDP execution.

How to select actions?

MDP execution

Let $h_t = (s_0, a_0, \ldots, a_{t-1}, s_t) \in \mathcal{H}$ denote the history at time step t.

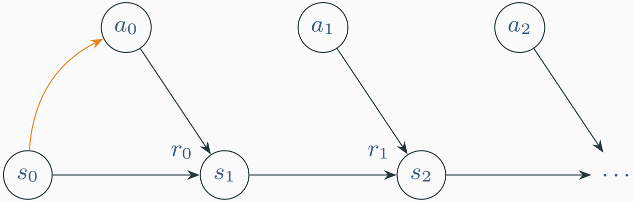


Figure 1: Bayesian graph of an MDP execution.

How to select actions?

MDP execution

Let $h_t = (s_0, a_0, \ldots, a_{t-1}, s_t) \in \mathcal{H}$ denote the history at time step t.

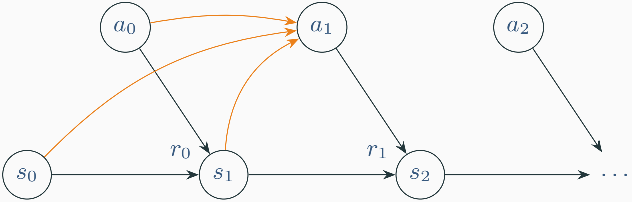


Figure 1: Bayesian graph of an MDP execution.

How to select actions?

MDP execution

Let $h_t = (s_0, a_0, \dots, a_{t-1}, s_t) \in \mathcal{H}$ denote the history at time step t.

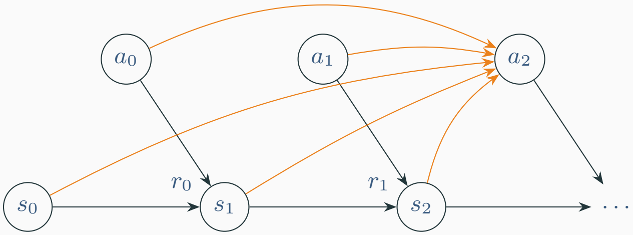


Figure 1: Bayesian graph of an MDP execution.

How to select actions?

MDP execution

Let $h_t = (s_0, a_0, \dots, a_{t-1}, s_t) \in \mathcal{H}$ denote the history at time step t.

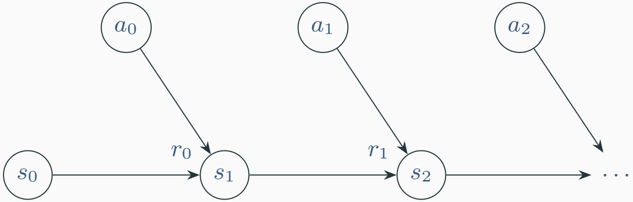


Figure 1: Bayesian graph of an MDP execution.

How to select actions? Naively, the action a_t should depend on the history h_t, that contains all the information available: $a_t \sim \eta(\cdot|h_t)$.

MDP execution

Let $h_t = (s_0, a_0, \dots, a_{t-1}, s_t) \in \mathcal{H}$ denote the history at time step t.

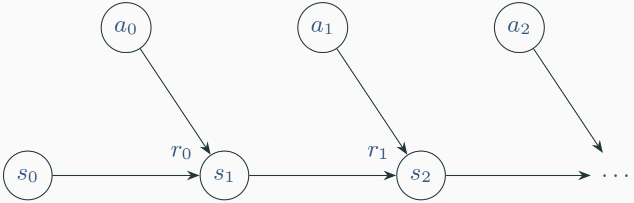


Figure 1: Bayesian graph of an MDP execution.

How to select actions? Naively, the action a_t should depend on the history h_t, that contains all the information available: $a_t \sim \eta(\cdot|h_t)$.

Can we relax this dependence by exploiting the Markov property of MDPs?

Stochastic policies in MDPs

Definition (History-dependent policy)

A history-dependent policy $\eta \in H = \mathcal{H} \to \Delta(\mathcal{A})$ is a mapping from a history to a distribution over the actions, whose density writes $\eta(a_t|h_t)$.

$$V^\eta(h) = \mathop{\mathbb{E}}_{\substack{a_t \sim \eta(\cdot|h_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| h_0 = h \right]$$

Definition (Markov policy)

A Markov policy $\pi \in \Pi = \mathcal{S} \to \Delta(\mathcal{A})$ is a mapping from a state to a distribution over the actions, whose density writes $\pi(a_t|s_t)$.

$$V^\pi(s) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right]$$

Optimality of Markov policies in MDPs

These theorems consider discrete state, action and observation spaces.

Theorem (Existence of the optimal policy)

There exists at least one history-dependent policy $\eta^* \in H$ such that:

$$V^{\eta^*}(h) = \max_{\eta \in H} V^{\eta}(h),\ \forall h \in \mathcal{H}.$$

Theorem (Optimality of Markov policies in MDPs)

There exists at least one Markov policy that performs as good as any history-dependent policy:

$$\max_{\pi \in \Pi} V^{\pi}(s) = \max_{\eta \in H} V^{\eta}(h),\ \forall h \in \mathcal{H},$$

where s is the last state in history h.

In MDPs, we will thus focus on (stationary) Markov policies $\pi \in \Pi = \mathcal{S} \to \Delta(\mathcal{A})$.

MDP execution with a Markov policy

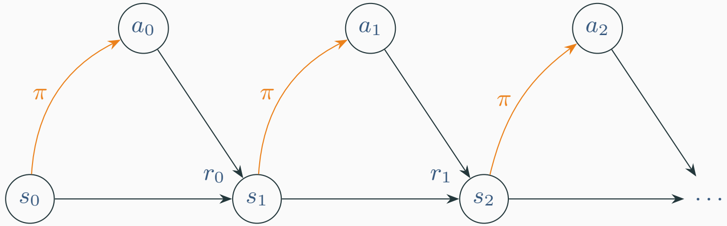


Figure 2: Bayesian graph of an MDP execution with a Markov policy.

Q-functions in MDPs

Definition (Q-function of a Markov policy)
The Q-function of a policy π gives the expected return starting from a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$, and following policy π afterwards:

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\substack{s_{t+1} \sim T(\cdot|s_t, a_t) \\ a_{t+1} \sim \pi(\cdot|s_{t+1})}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a \right].$$

It can be observed that $V^\pi(s) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [Q^\pi(s, a)]$.

Definition (Optimal Q-function)
The optimal Q-function gives the optimal expected return starting from a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$:

$$Q(s, a) = \max_{\pi \in \Pi} Q^\pi(s, a) = Q^{\pi^*}(s, a).$$

Deterministic Markov policy

Let us define the deterministic Markov policy.

Definition (Deterministic Markov policy)

A deterministic Markov policy $\mu \in M = \mathcal{S} \to \mathcal{A}$ is a mapping from a state to an action, written $a_t = \mu(s_t)$.

$$V^{\mu}(s) = \mathop{\mathbb{E}}_{s_{t+1} \sim T(\cdot|s_t, \mu(s_t))} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mu(s_t)) \middle| s_0 = s \right]$$

Optimality of deterministic Markov policies in MDPs

By definition of the Q-function ($Q = Q^{\pi^*}$), we can build an optimal policy π^*:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} Q(s,a) \\ 0 & \text{else} \end{cases}$$

The deterministic Markov policy $\mu^*(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$ has the same return as π^*, since it selects the same actions.

Theorem (Optimality of deterministic Markov policies)

In MDPs, there exists a deterministic Markov policy that performs at least as good as any history-dependent policy:

$$\max_{\mu \in M} V^{\mu}(s) = \max_{\pi \in \Pi} V^{\pi}(s) = \max_{\eta \in H} V^{\eta}(h),\ \forall h \in \mathcal{H},$$

where s is the last state in history h. The proof is in [Bertsekas, 2012].

$\Rightarrow$ An optimal (deterministic) policy can be derived from the Q-function.

Bellman optimality equation

Theorem (Bellman optimality equation)
The optimal Q-function satisfies the Bellman optimality equations:

$$\begin{aligned} Q(s,a) &= R(s,a) + \gamma \underset{s' \sim T(\cdot|s,a)}{\mathbb{E}} \left[\max_{a' \in \mathcal{A}} Q(s',a') \right], \; \forall s \in \mathcal{S}, \; \forall a \in \mathcal{A} \\ &= (BQ)(s,a), \; \forall s \in \mathcal{S}, \; \forall a \in \mathcal{A}, \end{aligned}$$

where B denotes the Bellman optimality operator.

Notes:

- The Bellman optimality operator B is a contraction mapping.
- The Bellman optimality equations $q = Bq$ have a unique solution Q.

Learning the Q-function

A1. When the MDP model $\mathcal{M}$ is known[1], we can use dynamic programming. It applies the Bellman update synchronously on all states and actions:

$$Q_N(s,a) = R(s,a) + \gamma \underset{s' \sim T(\cdot|s,a)}{\mathbb{E}} \left[\max_{a' \in \mathcal{A}} Q_{N-1}(s',a') \right], \ \forall s \in \mathcal{S}, \ \forall a \in \mathcal{A}.$$

Note: $Q_0(s,a) = 0$. This converges to the optimal Q-function: $\lim_{N \to \infty} Q_N = Q$.

A2. When the MDP model $\mathcal{M}$ is unknown, we can use Q-learning on sample transitions (s,a,r,s'). It applies "a small amount of sample Bellman update" asynchronously on (s,a):

$$Q_k(s,a) = (1-\alpha_k) Q_{k-1}(s,a) + \alpha_k \left(r + \gamma \max_{a' \in \mathcal{A}} Q_{k-1}(s',a') \right).$$

Note: $Q_0(s,a) = 0$. This converges to the optimal Q-function, under mild conditions on the learning rates α_k and the samples (s,a,r,s').

[1] Or learned from a dataset of transitions $\mathcal{T} = \{s_i, a_i, r_i, s_i'\}_{i=1}^{|\mathcal{T}|}$.

Approximate Q-learning

MDPs with large state spaces

Consider an MDP with a large or continuous state space. For example,

- Game of Go: $|\mathcal{S}| \approx 1.74 \times 10^{182}$, $|\mathcal{A}| = 441$,
- Smart thermostat: $\mathcal{S} = \mathbb{R}^2$, $|\mathcal{A}| = 2$.

So far, we have focused on lookup tables (one entry for every pair (s, a)):

- In DP, can we compute $Q_N(s, a)$ for all (s, a)?
- In Q-learning, what is the probability of updating $Q_k(s, a)$?
- In both cases, is it possible to store the current estimate for all (s, a)?

MDPs with large state spaces

Consider an MDP with a large or continuous state space. For example,

- Game of Go: $|\mathcal{S}| \approx 1.74 \times 10^{182}$, $|\mathcal{A}| = 441$,
- Smart thermostat: $\mathcal{S} = \mathbb{R}^2$, $|\mathcal{A}| = 2$.

So far, we have focused on lookup tables (one entry for every pair (s, a)):

- In DP, can we compute $Q_N(s, a)$ for all (s, a)? No.
- In Q-learning, what is the probability of updating $Q_k(s, a)$? Low or zero.
- In both cases, is it possible to store the current estimate for all (s, a)? No.

One alternative is to use function approximators.

- Allows a compact representation.
- Allows to generalize to unseen inputs.

Function approximators

There exists many classes of function approximators:

- Linear approximator,
- Decision trees,
- Random forests,
- Nearest neighbors,
- Neural networks.

Function approximators

There exists many classes of function approximators:

- Linear approximator,
- Decision trees,
- Random forests,
- Nearest neighbors,
- Neural networks.

In this lecture, we focus on neural networks.

Function approximators

There exists many classes of function approximators:

- Linear approximator,
- Decision trees,
- Random forests,
- Nearest neighbors,
- Neural networks.

In this lecture, we focus on neural networks.

Some notations:

- When denoting a neural network evaluated at a given input x and parameter value θ_k, we use $f_{\theta_k}(x) \coloneqq f(x; \theta_k)$.
- When denoting the gradient of a neural network with respect to its parameters, evaluated at a given input value x and parameter value θ_k, we use $\nabla_\theta f_{\theta_k}(x) \coloneqq (\nabla_\theta f(x; \theta))(\theta_k)$.

Approximate algorithms for learning Q-functions

A1. When transitions can be sampled from $\mathcal{M}$ or a set $\mathcal{T} = \{s_i, a_i, r_i, s'_i\}_{i=1}^{|\mathcal{T}|}$ is available, we can use approximate dynamic programming. See last lecture.

Algorithm 1: Fitted Q-Iteration

```
1 Let $\hat{Q}_0(s,a) = 0,\ \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}$.
2 for $n \leftarrow 1, \dots, N$ do
3     Let $x_i^n = (s_i, a_i),\ i = 1, \dots, |\mathcal{T}|$.
4     Let $y_i^n = r_i + \gamma \max_{a \in \mathcal{A}} \hat{Q}_{n-1}(s'_i, a),\ i = 1, \dots, |\mathcal{T}|$.
5     Use a regression algorithm to induce $\hat{Q}_n$ from $\{(x_i^n, y_i^n)\}_{i=1}^{|\mathcal{T}|}$.
```

A2. When successive transitions (s, a, r, s') can be sampled online, we can use approximate Q-learning. This is the focus of this course. Why online learning?

- Random samples does not always visit the whole state-action space $\mathcal{S} \times \mathcal{A}$.
- Transitions can be sampled by a policy derived from the Q approximation.
- Particularly useful when interaction with the MDP is costly.

Q-function approximator

We use an approximation $Q_\theta(s,a) \approx Q(s,a)$, where $\theta \in \mathbb{R}^d$ is a parameter vector. The input of the function approximator is typically a vector $\mathbf{x} = x(s,a)$.[2]

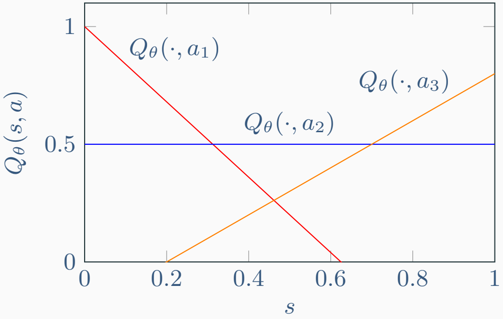


Figure 3: Example of linear Q_θ with $\mathcal{S} = [0,1] \subset \mathbb{R}$, $|\mathcal{A}| = 3$.

$Q_\theta(s,a) = x(s,a) \cdot \theta$ with $x(s,a) = \begin{pmatrix} s\delta_{a,a_1} & \delta_{a,a_1} & s\delta_{a,a_2} & \delta_{a,a_2} & s\delta_{a,a_3} & \delta_{a,a_3} \end{pmatrix}^T$ and $\theta = \begin{pmatrix} -1.6 & 1 & 0 & 0.5 & 1 & -0.2 \end{pmatrix}^T$.

[2]Note that this feature vector is fixed, it does not depend on θ.

Approximate Q-learning

Consider the standard Q-learning update for a transition (s, a, r, s').

$$\begin{aligned} Q_k(s,a) &= (1-\alpha_k)Q_{k-1}(s,a) + \alpha_k \left(r + \gamma \max_{a' \in \mathcal{A}} Q_{k-1}(s',a') \right) \\ &= Q_{k-1}(s,a) + \alpha_k \underbrace{\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{k-1}(s',a') - Q_{k-1}(s,a) \right)}_{\delta_k} \end{aligned}$$

In approximate Q-learning, we want to find a θ_k such that:

$$Q_{\theta_k}(s,a) = Q_{\theta_{k-1}}(s,a) + \alpha_k \underbrace{\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s',a') - Q_{\theta_{k-1}}(s,a) \right)}_{\delta_k}.$$

How to adapt θ_k such as to increase $Q_{\theta_{k-1}}(s,a)$ proportionally to $\alpha_k \delta_k$?

Approximate Q-learning

Consider the standard Q-learning update for a transition (s, a, r, s').

$$\begin{aligned} Q_k(s,a) &= (1-\alpha_k) Q_{k-1}(s,a) + \alpha_k \left(r + \gamma \max_{a' \in \mathcal{A}} Q_{k-1}(s',a') \right) \\ &= Q_{k-1}(s,a) + \alpha_k \underbrace{\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{k-1}(s',a') - Q_{k-1}(s,a) \right)}_{\delta_k} \end{aligned}$$

In approximate Q-learning, we want to find a θ_k such that:

$$Q_{\theta_k}(s,a) = Q_{\theta_{k-1}}(s,a) + \alpha_k \underbrace{\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s',a') - Q_{\theta_{k-1}}(s,a) \right)}_{\delta_k}.$$

How to adapt θ_k such as to increase $Q_{\theta_{k-1}}(s,a)$ proportionally to $\alpha_k \delta_k$?

$$\theta_k = \theta_{k-1} + \quad \alpha_k \delta_k \quad \nabla_\theta Q_{\theta_{k-1}}(s,a)$$

Approximate Q-learning

Consider the standard Q-learning update for a transition (s, a, r, s').

$$\begin{aligned} Q_k(s,a) &= (1-\alpha_k)Q_{k-1}(s,a) + \alpha_k \left(r + \gamma \max_{a' \in \mathcal{A}} Q_{k-1}(s', a') \right) \\ &= Q_{k-1}(s,a) + \alpha_k \underbrace{\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{k-1}(s', a') - Q_{k-1}(s,a) \right)}_{\delta_k} \end{aligned}$$

In approximate Q-learning, we want to find a θ_k such that:

$$Q_{\theta_k}(s,a) = Q_{\theta_{k-1}}(s,a) + \alpha_k \underbrace{\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a') - Q_{\theta_{k-1}}(s,a) \right)}_{\delta_k}.$$

How to adapt θ_k such as to increase $Q_{\theta_{k-1}}(s,a)$ proportionally to $\alpha_k \delta_k$?

$$\theta_k = \theta_{k-1} + \underbrace{\alpha_k \delta_k}_{\substack{\text{desired} \\ \text{change in} \\ Q_{\theta_k}(s,a)}} \underbrace{\nabla_\theta Q_{\theta_{k-1}}(s,a)}_{\substack{\text{direction that} \\ \text{increases} \\ Q_\theta(s,a)}}$$

Loss in approximate Q-learning

Note that the approximate Q-learning update

$$\theta_k = \theta_{k-1} + \alpha_k \underbrace{\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a') - Q_{\theta_{k-1}}(s, a) \right)}_{\delta_k} \nabla_\theta Q_{\theta_{k-1}}(s, a)$$

is equivalent to

$$\theta_k = \theta_{k-1} - \frac{1}{2}\alpha_k \nabla_\theta \left(r + \gamma \max_{a' \in \mathcal{A}} \left(\mathrm{sg}_\theta \left[Q_{\theta_{k-1}}(s', a') \right] \right) - Q_{\theta_{k-1}}(s, a) \right)^2$$

where sg_θ denotes the *stop-gradient* operator for θ, **informally defined as**:

$$\mathrm{sg}_\theta\, f(\theta) = f(\theta),$$
$$\nabla_\theta\, \mathrm{sg}_\theta\, f(\theta) = 0.$$

In other words, this update is a gradient step in the direction that minimizes the L_2 distance to a target $r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a')$, considered independent of θ.

Approximate Q-learning algorithm

In summary, the approximate Q-learning algorithm writes as follows.

Algorithm 2: Approximate Q-learning

```
1 Initialize $\theta$ randomly.
2 Reset environment to $s_0$.
3 for $k \leftarrow 1, \dots, K$ do
4     Select $a_{k-1} \sim \mathcal{U}(\mathcal{A})$.
5     Take action $a_{k-1}$ and observe $r_{k-1}$ and $s_k$.
6     Let $(s, a, r, s') = (s_{k-1}, a_{k-1}, r_{k-1}, s_k)$.
7     Compute $\delta_k = r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a') - Q_{\theta_{k-1}}(s, a)$.
8     Update $\theta_k = \theta_{k-1} + \alpha_k \delta_k \nabla_\theta Q_{\theta_{k-1}}(s, a)$.
```

Issues with approximate Q-learning

Differences with supervised learning

Is this a valid SGD algorithm? What is the loss? What about the samples?

$$\theta_k = \theta_{k-1} - \frac{1}{2}\alpha_k \nabla_\theta \left(r + \gamma \max_{a' \in \mathcal{A}} \left(\mathrm{sg}_\theta \left[Q_{\theta_{k-1}}(s', a') \right] \right) - Q_{\theta_{k-1}}(s, a) \right)^2$$

Differences with supervised learning

Is this a valid SGD algorithm? What is the loss? What about the samples?

$$\theta_k = \theta_{k-1} - \frac{1}{2}\alpha_k \nabla_\theta \left(r + \gamma \max_{a' \in \mathcal{A}} \left(\mathrm{sg}_\theta \left[Q_{\theta_{k-1}}(s', a') \right] \right) - Q_{\theta_{k-1}}(s, a) \right)^2$$

It is not, for the following reasons:

- **I1.** Non stationary ("moving target"),
- **I2.** Non IID samples ("correlated samples").

Note that the problem of non stationarity comes from bootstrapping.

$\Rightarrow$ No convergence guarantees. Divergence can be observed even in simple cases.

Example: divergence with linear function approximator

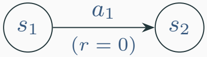


Figure 4: Transition in a simple MDP with $|\mathcal{S}| = 2$ and $|\mathcal{A}| = 1$.

Let $\mathcal{S} = \{s_1, s_2\}$ and $\mathcal{A} = \{a_1\}$. Let $\mathbf{x}_1 = x(s_1, a_1) = (1)$ and $\mathbf{x}_2 = x(s_2, a_1) = (2)$. Let $Q_\theta(s, a) = x(s, a) \cdot \theta$ a linear approximator with $\theta \in \mathbb{R}$.

The update writes as follows for $(s, a, r, s') = (s_1, a_1, 0, s_2)$.

$$\begin{aligned}
\theta_k &= \theta_{k-1} + \alpha_k \left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a') - Q_{\theta_{k-1}}(s, a) \right) \nabla_\theta Q_{\theta_{k-1}}(s, a) \\
&= \theta_{k-1} + \alpha_k \left(0 + \gamma(\theta_{k-1} \cdot \mathbf{x}_2) - (\theta_{k-1} \cdot \mathbf{x}_1) \right) \mathbf{x}_1 \\
&= \theta_{k-1} + \alpha_k \left(2\gamma\theta_{k-1} - \theta_{k-1} \right) \\
&= \theta_{k-1} + \alpha_k \theta_{k-1} \left(2\gamma - 1 \right)
\end{aligned}$$

Let us consider $\theta_0 > 0$ and $\gamma > 0.5$, repeating this update gives $\lim_{k \to \infty} \theta_k = \infty$.

Target network

In order to solve the issue of non stationarity (**I1**) of the target, the regression problem is broken into successive regression problems with fixed targets.

Algorithm 3: Approximate Q-learning with target network

```
1  Initialize $\theta_0$ randomly.
2  Save $\theta' \leftarrow \theta_0$.
3  Reset environment to $s_0$.
4  for $k \leftarrow 1, \ldots, K$ do
5      if $\theta_k$ has converged then
6          Update $\theta' \leftarrow \theta_k$.
7      Select $a_{k-1} \sim \mathcal{U}(\mathcal{A})$.
8      Take action $a_{k-1}$ and observe $r_{k-1}$ and $s_k$.
9      Let $(s, a, r, s') = (s_{k-1}, a_{k-1}, r_{k-1}, s_k)$.
10     Compute $\delta_k = r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta'}(s', a') - Q_{\theta_{k-1}}(s, a)$.
11     Update $\theta_k = \theta_{k-1} + \alpha_k \delta_k \nabla_\theta Q_{\theta_{k-1}}(s, a)$.
```

Between two target network updates, the target for a given (s, a) is sampled according to a fixed distribution: $\mathbb{E}_{s' \sim T(\cdot|s,a)} [R(s, a) + \gamma \max_{a' \in \mathcal{A}} Q_{\theta'}(s', a')]$.

Target network

Some notes about approximate Q-learning with target network:

- Usually, the update is periodic: $k \bmod C = 0$ (not a convergence criterion).
- It is linked to fitted Q-iteration (FQI). At the n^{th} target update, the target network $Q_{\theta'}(s,a)$ has fitted $\mathbb{E}_{s' \sim T(\cdot|s,a)}\left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_{\theta'}(s',a')\right]$ using the $(n-1)^{\text{th}}$ target network in its target, like in FQI. A notable difference is the first target which is randomly initialized in Q-learning.
- After n target updates, the approximate Q-function is at best n-step greedy.

Experience replay

In order to solve the issue of non IID samples (**I2**), past transitions are stored in a buffer, and updates are performed on random transitions sampled from it.

Algorithm 4: Approximate Q-learning with replay buffer

```
1  Initialize $\theta_0$ randomly.
2  Initialize empty buffer $\mathcal{B}$.
3  Reset environment to $s_0 \sim p_0$.
4  for $k \leftarrow 1, \dots, K$ do
5      Select $a_{k-1} \sim \mathcal{U}(\mathcal{A})$.
6      Take action $a_{k-1}$ and observe $r_{k-1}$ and $s_k$.
7      Store transition $(s_{k-1}, a_{k-1}, r_{k-1}, s_k)$ in $\mathcal{B}$.
8      Sample transition $(s, a, r, s')$ from $\mathcal{B}$.
9      Compute $\delta = r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a') - Q_{\theta_{k-1}}(s, a)$.
10     Update $\theta_k = \theta_{k-1} + \alpha_k \delta \nabla_\theta Q_{\theta_{k-1}}(s, a)$.
```

As the replay buffer $\mathcal{B}$ is filled, the successive samples that are drawn from the buffer are more and more IID.

Experience replay

Some notes about approximate Q-learning with experience replay:

- Allows to reuse a sample several time (like in supervised learning).
- In practice, the buffer has a fixed capacity, we need a replacement strategy.
- Usually, when the buffer is full, the new transition replaces the oldest.

We can also use a sampling strategy. One example is prioritized experience replay: the probability of sampling a transition (s, a, r, s') is proportional to its absolute temporal difference $|\delta| = |r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a') - Q_{\theta_{k-1}}(s, a)|$.

- The idea is to select samples for which the error is big more often.
- When generating a transition, store $(s, a, r, s', |\delta|)$ using current Q_θ.
- Each time that a transition is sampled, update $|\delta|$ using current Q_θ.

Example: $p(\delta) \propto |\delta|^\alpha$, with α the temperature. Requires a clever data structure.

Overestimation of maximum sampling

There is a third issue in approximate Q-learning.

Theorem (Jensen inequality)

For any convex function ϕ and random variable X, we have:

$$\mathbb{E}\left[\phi\left(X\right)\right] \geq \phi\left(\mathbb{E}\left[X\right]\right).$$

Theorem (Convexity of the maximum function)

For any $n \geq 1$, the function ϕ_n: $(x_1, \ldots, x_n) \mapsto \max_{1 \leq i \leq n} \left(x_i\right)$ is convex:

$$\begin{aligned}&\lambda\phi_n(x_1, \ldots, x_n) + (1-\lambda)\phi_n(y_1, \ldots, y_n)\\ &\quad \geq \phi_n(\lambda x_1 + (1-\lambda)y_1, \ldots, \lambda x_n + (1-\lambda)y_n),\ \forall\lambda \in [0, 1].\end{aligned}$$

Theorem (Overestimation of maximum sampling)

From the Jensen inequality and the convexity of the maximum function, we can write:

$$\mathbb{E}\left[\max_i \left(X_i\right)\right] \geq \max_i \left(\mathbb{E}\left[X_i\right]\right).$$

Overestimation in approximate Q-learning

Reminder: the sample target is $r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s', a')$, with $Q_\theta(s', a') \approx Q(s', a')$.

Let us consider $Q_\theta(s', a')$ as random variables.[3] Let us assume them unbiased:

$$\mathop{\mathbb{E}}_{\theta} \left[Q_\theta(s', a') \right] = Q(s', a'),\ \forall s' \in \mathcal{S},\ \forall a' \in \mathcal{A}.$$

Even under this assumption, the theorem of overestimation of maximum sampling allows us to write:

$$\mathop{\mathbb{E}}_{\theta} \left[\max_{a' \in \mathcal{A}} Q_\theta(s', a') \right] \geq \max_{a' \in \mathcal{A}} \mathop{\mathbb{E}}_{\theta} \left[Q_\theta(s', a') \right] = \max_{a' \in \mathcal{A}} Q(s, a).$$

This is known has the problem of overestimation in Q-learning (**I3**).

As the updates repeat, this can grows unboundedly (even with a target network).

[3] Randomness comes from θ through the initialisation, the transitions sampled in the environment, the usage of bootstrapping, etc.

Double Q-learning

In order to fix the overestimation problem, we can use double Q-learning.

Instead of picking $\max_{a' \in \mathcal{A}} Q_\theta(s', a') = Q_\theta\left(s', \arg\max_{a' \in A} Q_\theta(s', a')\right)$, we use another estimation $Q_{\theta'}$ to select the action in Q_θ.[4]

$$Q_\theta\left(s', \arg\max_{a' \in A} Q_{\theta'}(s', a')\right)$$

This decouples action selection and action evaluation. If Q_θ and $Q_{\theta'}$ are not correlated, this is an unbiased estimator.

Intuitively, under the assumption of unbiasedness, if $Q_{\theta'}$ selects an action a' that it overestimates, $Q_\theta(s', a')$ still is an unbiased estimate of $Q(s', a')$ since it is uncorrelated to $Q_{\theta'}$.

[4] $Q_{\theta'}$ is not necessarily the target network.

Deep Double Q-learning

When using a target network $Q_{\theta'}$, it is common to use it as second network for double Q-learning. Despite violating the decorrelation assumption, this change mitigates the overestimation problem in practice.

Algorithm 5: Approximate Q-learning with target double Q-learning

```
1  Initialize $\theta_0$ randomly.
2  Save $\theta' \leftarrow \theta_0$.
3  Reset environment to $s_0$.
4  for $k \leftarrow 1, \ldots, K$ do
5      if $\theta_k$ has converged then
6          Update $\theta' \leftarrow \theta_k$.
7      Select $a_{k-1} \sim \mathcal{U}(\mathcal{A})$.
8      Take action $a_{k-1}$ and observe $r_{k-1}$ and $s_k$.
9      Let $(s, a, r, s') = (s_{k-1}, a_{k-1}, r_{k-1}, s_k)$.
10     Compute $\delta_k = r + \gamma Q_{\theta_{k-1}}\left(s', \arg\max_{a' \in \mathcal{A}} Q_{\theta'}(s', a')\right) - Q_{\theta_{k-1}}(s, a)$.
11     Update $\theta_k = \theta_{k-1} + \alpha_k \delta_k \nabla_\theta Q_{\theta_{k-1}}(s, a)$.
```

Improvements to approximate Q-learning

Actions out

Reminder: we consider discrete actions.

Naively, the approximation can be implemented by $Q_\theta(s, a) = f_\theta(x(s, a))$, with $f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}$ a neural network parametrized by θ (actions in).

Computing the target $r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s', a')$ requires $|\mathcal{A}|$ forward passes of f_θ[5].

Instead, the approximation can be implemented by $Q_\theta(s, a) = g_\theta(x(s))_a$, with $g_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{|\mathcal{A}|}$ a neural network parametrized by θ (actions out).

Computing the target only requires a single forward pass of g_θ. Moreover, the intermediate features (activations) of g_θ are shared for all actions, which is a good inductive bias if the Q-function of all actions are strongly correlated for a given s.

[5] It can nevertheless be batched and parallelized.

Mini-batches

Like in supervised learning, we can use mini-batches instead of pure SGD.

- Reduces the variance of the gradient estimates.
- Using a bigger mini-batch size B is more computationally expensive.
- There is a trade-off, which interacts with the learning rate α.

Algorithm 6: Approximate Q-learning with replay buffer and mini-batches

```
1  Initialize $\theta_0$ randomly.
2  Initialize empty buffer $\mathcal{B}$.
3  Reset environment to $s_0 \sim p_0$.
4  for $k \leftarrow 1, \ldots, K$ do
5      Select $a_{k-1} \sim \mathcal{U}(\mathcal{A})$.
6      Take action $a_{k-1}$ and observe $r_{k-1}$ and $s_k$.
7      Store transition $(s_{k-1}, a_{k-1}, r_{k-1}, s_k)$ in $\mathcal{B}$.
8      Sample transitions $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^{B}$ from $\mathcal{B}$.
9      Compute $\delta_i = r_i + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s'_i, a') - Q_{\theta_{k-1}}(s_i, a_i)$, $i = 1, \ldots, B$.
10     Update $\theta_k = \theta_{k-1} + \alpha_k \frac{1}{B} \sum_{i=1}^{B} \delta_i \nabla_\theta Q_{\theta_{k-1}}(s_i, a_i)$.
```

Behavior policy

The advantage of online learning is to improve the samples that are generated as we refine the target policy (i.e., the policy that is being optimized).

However, in this course, actions were selected with a random policy: $a_k \sim \mathcal{U}(\mathcal{A})$, which does not exploit the current target policy that can be derived from Q_θ.

We define the ε-greedy policy of an approximation Q_θ as follows:

$$\pi_\varepsilon^{Q_\theta}(a|s) = \varepsilon \frac{1}{|\mathcal{A}|} + (1-\varepsilon)\, \underbrace{\delta_{\arg\max_{a \in \mathcal{A}} Q_\theta(s,a)}(a)}_{\text{greedy policy}},$$

where $\delta_a \in \Delta(\mathcal{A})$ denotes the Dirac distribution centered in a.

The approximate Q-learning algorithm can be improved by using the ε-greedy policy as behavior policy.

- With probability ε, we select an action at random.
- With probability $1-\varepsilon$, we select the greedy action (according to Q_θ).
- Typically, ε starts at 1 and decreases to a low value (not 0).

Behavior policy

Algorithm 7: Approximate Q-learning

```
1 Initialize $\theta$ randomly.
2 Reset environment to $s_0$.
3 for $k \leftarrow 1, \dots, K$ do
4     Select $a_{k-1} \sim \pi_{\varepsilon_{k-1}}^{Q_{\theta_{k-1}}}(\cdot|s_{k-1})$.
5     Take action $a_{k-1}$ and observe $r_{k-1}$ and $s_k$.
6     Let $(s, a, r, s') = (s_{k-1}, a_{k-1}, r_{k-1}, s_k)$.
7     Compute $\delta_k = r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{k-1}}(s', a') - Q_{\theta_{k-1}}(s, a)$.
8     Update $\theta_k = \theta_{k-1} + \alpha_k \delta_k \nabla_\theta Q_{\theta_{k-1}}(s, a)$.
```

- In standard Q-learning, we can use any behavior policy, as soon as every pair (s, a) is sampled infinitely many times.
- By selecting actions according to Q_θ, the distribution of the sample transitions changes during training.

Deep Q-Network

In DQN [Mnih et al., 2015], they use

- A target network,
- A replay buffer,
- Q-networks with actions-out,
- An ε-greedy behavior policy,
- Mini-batch gradient descent with an optimizer.

In Rainbow [Hessel et al., 2018], they also use

- Prioritized experience replay,
- Double DQN,
- Dueling DQN,
- Distributional DQN,
- Noisy DQN.

They achieve significantly better performance by combining all those components.

Demonstration

Conclusions

Conclusions

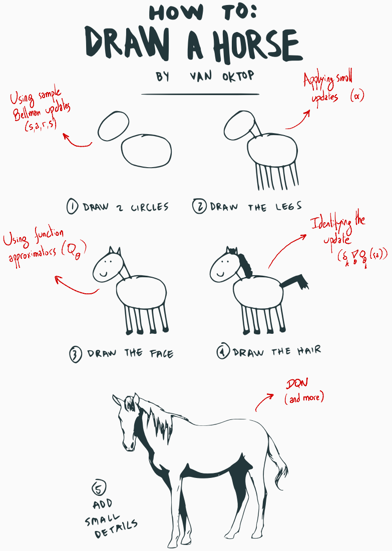
Conclusions:

- Q-learning allows online learning.
- It can be adapted to the approximate setting using suitable approximators.
- The Q-learning update needs to be adapted to this setting.
- Approximate Q-learning raises several problems that can be solved:
  - Non stationarity,
  - Non IID,
  - Overestimation bias.
- Many more improvements are required for achieving a decent performance.

Limitations:

- There exists no straightforward adaptations for continuous actions.
- Q-learning approximates $Q(s, a)$ instead of $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$.

Roadmap to human-level control on ATARI[6]


HOW TO:
DRAW A HORSE
BY VAN OKTOP
Using sample Bellman updates (s,a,r,s')
Applying small updates (α)
① DRAW 2 CIRCLES
② DRAW THE LEGS
Using function approximators (Q_θ)
Identifying the update $(\delta_k \nabla_\theta Q_{\theta_k}(s,a))$
③ DRAW THE FACE
④ DRAW THE HAIR
DQN (and more)
⑤ ADD SMALL DETAILS

[6] oktop.tumblr.com

References

Dimitri Bertsekas. *Dynamic Programming and Optimal Control: Volume II.* Athena Scientific, 2012.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-Level Control through Deep Reinforcement Learning. *Nature*, 518 (7540):529–533, 2015.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32 (01), 2018.