Reinforcement learning

Linus Falk

April 24, 2023

Contents

1 Introduction **2**

2 Markov Decision process **5**

3 Dynamic programming **9**

4 Model-free prediction **13**

5 Model-free control **19**

6 Model-based Reinforcement learning **23**

7 Model-based RL in continuous action and state spaces **26**

8 Function approximation **27**

1 Introduction

Machine learning can typically be separated into three branches:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Reinforcement learning is different from the two other branches in the way that it is not supervised with labeled data, but it doesn't find patterns in data completely on its own as in unsupervised learning. Instead there is a reward signal, and the feedback can be delayed, so time come in as a factor as well. The actions of the so called **agent** affect the subsequent data it receives.

The agent-environment interface

In our reinforcement learning setup we have the **agent** which is the decision maker and the **environment**, which can be seen as everything else in the setup. The goal is to maximize the cumulative reward. The agents task is to learn this from experience.

The agent observes the environments state and takes some sort of action, the agent then receives a rewards from the action and the environments state changes. The agent must now observe the new state and take a new action, and so on. Note: in reinforcement learning the environment, agent and/or reward may be stochastic.

Probability theory - review

Lets consider two random variables $X \in \chi$ and $Y \in \gamma$

- Probability: the probability that we will observe $x \in \chi$ is written as:

$$\Pr\{X = x\} \tag{1}$$

- Conditional probability: If we have already observed $y \in \gamma$ then the probability that we will observe $x \in \chi$ is written:

$$\Pr\{X = x|Y = y\} \tag{2}$$

- Probability functions:

$$\mathrm{p(x)} = \Pr\{X = x\}, \mathrm{p}(x|y) = \Pr\{X = x|Y = y\} \tag{3}$$

- Probabilities sum to 1:

$$\sum_{x\in\chi} p(x) = 1 \text{ and } \sum_{x\in\chi} p(x|y) = 1 \tag{4}$$

- The expected value:

$$\mathrm{E}[X] = \sum_{x\in\chi} xp(x), E[X|Y = y] = \sum_{x\in\chi} xp(x|y) \tag{5}$$

What is a state?

The state S_t is a representation of the environment at time t. The state space $\mathcal{S}$ is a set of all possible states. S_t will be dependent on what happened in the past, before time t. S_t contains all information that is relevant for the prediction of the future at time t.

The Markov property

$$p(S_{t+1}|S_0, A_0, S_1, A_1, \ldots, S_t, A_t) = p(S_{t+1}|S_t, A_t) \tag{6}$$

Example: The taxi environment

- Taxi: 25 different positions possible
- Passenger: 5 positions including picked up
- Destinations: 4 options
- In total 25 x 5 x 4 = 500 configurations

We can enumerate all possible configurations in some way, let $\mathcal{S} = \{0, 1, \ldots, 499\}$. This is a finite state space.

Example: Inverted pendulum

When balancing a stick, the control signal or action is the torque at the bottom. The state?

- $S_t = \theta_t$

No! we don't have any information about the direction the stick is moving in.

- $S_t = \begin{bmatrix} \theta_t \\ \frac{d\theta_t}{dt} \end{bmatrix} \in \mathcal{S} \subset \mathbb{R}^2$

This is an example of a continuous state space, Infinitely many possible states.

What is the reward

The reward R_t is a scalar signal that tells how it is doing at that time step t. The goal is the maximize the cumulative reward over **long-time**.

In the taxi example could we for example see a reward of -10 if a illegal pick-up or drop-off occurred, a successful drop-off would score +20 and all other actions would score -1 in reward. So in other words, to maximize the total reward we should deliver passenger in as few steps as possible.

In the inverted pendulum example could the goal be to keep the angle as close to zero as possible, perhaps: $R_{t+1} = -\theta_t^2$ and if we also want to use a low torque we could try: $R_{t+1} = -c_1\theta_t^2 - c_2 a_t^2$. With $\theta_t = 0$ and $a_t = 0$ we would get the maximum reward $R_{t+1} = 0$

Sequential decision making

The goal is to select actions that maximize the future reward. So actions may have long term consequences and reward may be delayed. It can also be the case that sacrificing immediate reward can gain more long-term reward. Examples:

- A financial investment,
- Fueling a car, prevent fuel stop later

Designing the reward functions

In this course we will often receive the reward functions. However it is important to note that the design of the reward function is important since it will determine what the agent tries to achieve. In some cases the agent might also find unintended ways to increase the reward and the reward influences **how** the agent learns.

an example would be the mountain car with the goal of reaching the top/the flag with as few steps as possible. The reward is -1 for each step until the flag is reached. This is a sparse rewards, all action looks equally bad until the flag is reached the first time. It could be possible to use a more informative reward function, but it is important to make sure that the agent still optimize the correct thing we want.

What is an action

An action A_t is the way an agent can affect the state a time t and $\mathcal{A}$ is the set of all possible actions. In the taxi example would an action be for example: go north, go west and pick-up and in the pendulum example could it be: torque.

Stochastic environments

A deterministic environment is one which the outcome of an action is determined by the current state and the actions taken. While in the stochastic environment there is some randomness and the outcome of an action is not completely determined by the current state. This means that the same action in the same state might have different outcomes at different times. This means that it is harder for the agent to learn an optimal policy.

The power of feedback

If no feedback is given we can pre-compute all actions $A_0, A_1, ...$ if the first state S_0 is given. While with feedback we decide the action A_t after we have observed the state at time t, S_t. Example could be to walk with or without blindfold. So instead of trying to find good actions we want to try to find a good policy.

What is a policy?

Policy is a distribution over actions given states:

$$\pi(a|s) = \Pr\{A_t = a|S_t = s\} \tag{7}$$

A policy defines how the agent will behave in different states. If we have a deterministic policy we can sometimes write:

$$a = \pi(s) \tag{8}$$

Example: The linear quadratic regulator

$$S_t = FS_t + GA_t + W_t \tag{9}$$

If we want to maximize the expected value

$$E\left[\sum_{t=0}^{\infty}(-c_1 S_t^T S_t - c_2 A_t^T A_t)\right] \tag{10}$$

Then the **Optimal policy:** when we observe s_t, choose the action:

$$a_t = \pi(s_t) = -Ls_t \tag{11}$$

for some fixed L, that we can find if F and G are know.

Terminology

Reinforcement learning	**Optimal control**
Environment	System / plant
State, S_t	State, x(t)
Action, A_t	Input, u(t)
Reward	Cost, c(t)
Policy	Controller
Maximize reward	Minimize cost
Learn policy from experience	Use model to find controller (LQC or MPC)

Table 1: example

Exploration vs exploitation

The agent should be able to learn a good policy without losing to much reward. **Exploration** is used to learn more about the environment and **exploitation** is the use of information to maximize the reward. We usually have both explore and exploit.

Model vs model-free

In model based reinforcement learning we learn a model from experience and uses the model to find a good policy and/or predictions. While in model-free learning we learn a policy and/or predictions without first learning the model. In this course the main focus will be on model-free reinforcement learning.

2 Markov Decision process

This lecture will explain the concepts of Markov decision process, MDP's, the Bellman equation and optimal policy

Recap

For a state to have **Markov property** means that the state contains all the information that is useful to predict the future. In other words, we don't need the previous states to predict the future, the useful information is stored in the current state. A **Policy**: π(a|s) choosing an action a when we are in state s, can be deterministic or stochastic. The **prediction** gives the future cumulative reward following a policy. We want to find the policy that maximize the cumulative future reward by **control**.

Markov decision process

If we begin with assuming that $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ have finite numbers of elements then the **translation probabilities** are given by:

$$p(', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \tag{12}$$

The Markov property determines the **dynamics** of the environment, the probability of transitioning to a new state depends only on the current state and action. The **state transitions**:

$$p(s'|s, a) = \sum_{r \in \mathcal{R}} p(s', r|s, a) = \tag{13}$$

With the **expected reward**

$$r(s, a) = \mathbb{E}[R_{r+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} r p(s', r|s, a) \tag{14}$$

Episodic vs continuing tasks

- **Episodic tasks**
  - Has terminating states and the task end in finite time.
  - When reaching the terminating state the episode stops.
  - If you reach the terminating state you will stay there forever, receiving no future rewards.
- **Continuing tasks**
  - Often not a clear way to divide the task into independent episodes.
  - No state were the task is done
  - Must take into account infinitely many future rewards.

The return

In a given state we want to maximize the future reward we can receive: $R_{t+1}, R_{t+2,\ldots}$. To make it possible to have non finite number of rewards we introduce the **discounted reward**

The discounted reward

$$G_t = R_{t+1}R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \text{where } 0 < \gamma \leq 1 \tag{15}$$

since we put $\gamma \leq 1$ we put less value on future rewards, $\gamma = 0.5 \Rightarrow \gamma^{10} = 0.001, \gamma = 0.9 \Rightarrow \gamma^{10} = 0.35$.

And if $\gamma < 1$ we make sure that $R_t < \overline{R}$ for all t, and then G_t is bounded:

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \overline{R} \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \overline{R} \tag{16}$$

If we know that the task ends after a finite number of steps we can get away with using non-discounted returns where $\gamma = 1$

The state value function

The state-value function estimates the expected long term rewards the agent will recieve starting from a specific state and following a given policy. Since S_t and R_t are random variables, the return is therefore also a random variable:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots$$

We must therefore consider the expected return:

The state-value function

The state-value function $v_\pi(s)$ of a MDP is the expected return starting from the state s and then following the policy π:

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s] \tag{17}$$

The prediction of cumulative reward is computed with $v_\pi(s)$

The action-value function

Another important value function is the **action-value function** that estimates the expected long-term reward an agent will receive starting from a specific state, taking a specific action and following a given policy.

The action-value function

The action-value function $q_\pi(s, a)$ is the exptected return starting from s, taking action a, and **then** following a policy π

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a] \tag{18}$$

This function is also often called the Q-function.

Bellman equations

Looking at the reward we can note that:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = R_{t+1} + \gamma G_{t+1} \tag{19}$$

Hence, the value function satisfies to following equation:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \end{aligned} \tag{20}$$

The value of s is the expected immediate reward plus the discounted expected vlue of the next state. In the same way is action-value function:

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}|S_t = s, A_t = a)] \tag{21}$$

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\ q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}|S_t = s, A_t = a)] \end{aligned} \tag{22}$$

The expectation

The state-value of a state s, is the expected action-value:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{23}$$

So for a deterministic policy a = $\pi(s)$ we get $v_\pi(s, \pi(s))$. And given s and a, the immediate reward r and the next state s' has probability $p(s', r|s, a)$ so that:

$$q_\pi(s,a) = \sum_{r,s'} p(s', r|s, a)(r + \gamma v_\pi(s')) \tag{24}$$

The Bellman equation for v_π and q_π

$$\begin{aligned} v_\pi = \sum_a \pi(a|s) q_\pi(s,a) = \sum_a \pi(a,s) \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')] \\ q_\pi(s,a) = \sum_{r,s'} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')] \end{aligned} \tag{25}$$

Solving the Bellman equation

Solving the Bellman equation is done by solving a system of linear equation, this is because the dependencies between the different states. Each state $s \in \mathcal{S}$ gives one equation. There is a unique solution that can be expressed analytically. If there is a large number of states, large $\mathcal{S}$, then there are more efficient ways to solve it with iterative solution. If $p(s', r|s, a)$ is not know, we need to **learn** $v_\pi(s)$ from **experience**. If the number of states $\mathcal{S}$ is infinite, we can't compute the value for each state individually, and instead we need to find some function $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$.

Optimal value function

This represent the maximum expected cumulative reward that can be achieved from a given state, following the best possible policy. It quantifies the highest long-term reward that the agent can expect to obtain when makin **optimal decision** from each state. The optimal state-value function is given by:

$$v_*(s) = \max v_\pi(s), \text{for all } s \in \mathcal{S} \tag{26}$$

The optimal action-value function is given by:

$$q_*(s,a) = \max q_\pi(s,a), \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \tag{27}$$

The optimal $v_*(s)$ should be the maximum of $q_*(s,a)$, so we have:

$$v_*(s) = \max_a q_*(s,a) \tag{28}$$

The equation for $q_*(s,a)$:

$$\begin{aligned} q_*(s,a) = \sum_{r,s'} p(s',r|s,a)(r + \gamma v_*(s')) = \max_a \mathbb{E}[R_{t+1} + \gamma(S_{t+1})|S_t = s, A_t = a] \\ = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \end{aligned} \tag{29}$$

Solving the Bellman optimality function

$$v_*(s) = \max_a \sum_{r,s'} p(s',s|s,a)[r + \gamma v_*(s')] \tag{30}$$

This is a system of non-linear equation (non linear because of the max function). There is one equation for each state, s. In general there is no closed form solution. But there are iterative methods to solve it.

Optimal policy

Partial ordering over policies: This is the relationship between different policies, where some of them can be ranked as better or worse than others. This is based on their performance or expected returns. For some policy pairs there is no such definitive ranking. This means that the ordering captures the hierarchy of policies in terms of effectiveness but not necessarily a linear ranking of them.

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq t_{\pi'}(s), \text{ for all } s \tag{31}$$

Theorem

- There exists at least one optimal policy π_* such that $\pi_* \geq \pi$ for all policies π.
- All optimal policies achieve the optimal state-value function $v_{\pi*}(s) = v_*(s)$
- All optimal policies achieve the action-value function $q_{\pi*}(s,a) = q_*(s,a)$

So how do make a decision in state s?

1. Choose an action, a that maximizes the optimal action-value $q_*(s,a)$
2. Then use an optimal policy form s'

The **control** part is to find this optimal policy. Were the policy is described with:

$$\pi_*(s) = \arg\max_a \sum_{r,s'} p(s',r|s,a)[r + \gamma v_*(s')] \tag{32}$$

which is optimal. But **note** if we already know $q_*(s,a)$ then we don't need the dynamics to find an optimal policy.

Repetition

Here follow some important concepts that were covered in previous lectures:

- **States, actions and rewards**, $s \in \mathcal{S}, a \in \mathcal{A}, r \in \mathcal{R}$
- **Dynamic/model:** $p(s', r|s, a)$
- **Policy:** $\pi(a|s)$ (For deterministic policy also $a = \pi(s)$)
- **The return:**
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{33}$$
- **State-value function:** Expected return when starting in s and following policy π:
$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{34}$$
- **Action-value function:** Expected return when starting in s, taking action a and **then** follow policy π
$$q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] \tag{35}$$
- **Relations:**
$$\begin{aligned} v_\pi &= \sum_a \pi(a|s) q_\pi(s|a) \\ q_\pi &= \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{36}$$
for a deterministic policy a $a = \pi(s)$ we have $v_\pi(s) = q_\pi(s, \pi(s))$
- **Bellman equation for state-values:**
$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \end{aligned} \tag{37}$$
- **Optimal value function:**
$$\begin{aligned} v_* &= \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S} \\ q_*(s, a) &= \max_\pi q_\pi(s, a), \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \end{aligned} \tag{38}$$
- **Bellman optimality equation:**
$$v_*(s) = \max_a q_*(s, a) = \max_a \sum_{r,s'} p(s', r|s, a)[r + \gamma v_*(s')] \tag{39}$$

3 Dynamic programming

Dynamic programming is a class of algorithms that solves a problem by breaking it down to smaller, overlapping sub-problems. Solving these sub-problems and combine them to one solution. Methods like value and policy iteration are useful for solving Markov Decision Processes by iteratively updating value functions or policies to find the optimal strategy. For this section we will assume that we know the dynamics $p(s', r|s, a)$.

- **Prediction**
  - Given a policy π, predict the expected future return from each state.
  - That is, find the state-value function $v_\pi(s)$
- **Control:**
  - Given a MDP, find an optimal policy π_*

– If we first compute v_*, then we can use:

$$\begin{aligned} q_*(s,a) &= \sum_{r,s'} p(s',r|s,a)[r+\gamma v_*(s')] \\ \pi_* &= \arg\max_a q_*(s,a) \end{aligned} \tag{40}$$

Policy evaluation

For the problem we are given: π, then we compute $v_\pi(s)$ for all $s \in \mathcal{S}$. Using the bellman equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s',r|s,a)[r+\gamma v_\pi(s')] \tag{41}$$

This gives us a system of linear equation that can be solved analytically or with iterative process. For large state and or action space, it's more effective to use the iterative alternative.

Iterative policy evaluation

We start of with making some sort of initial guess. v_0, for example: $v_0(s) = 0$ for all s. In each of the iterations we use the RHS of the bellman equation:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s',r|s,a)[r+\gamma v_k(s')], \text{ for all } s \in \mathcal{S} \tag{42}$$

If we get to a point were $v_k(s) = v_{k+1}(s)$ then we have reached a v_k that solves the Bellman equation. For convergence it can be shown that $v_k(s) \to v_\pi(s)$ as $k \to \infty$. This is done using the method of contraction mapping: Let u_k and v_k be two different estimates, then:

$$||y_{k+1} - v_{k+1}||_\infty \leq \gamma ||u_k - v_k||_\infty \tag{43}$$

with $u_k = v_\pi$

$$||v_\pi - v_{k+1}||_\infty \leq ||v_\pi - v_k||_\infty \tag{44}$$

Bootstrapping is the process of using the old estimate v_k to improve our new estimate

Implementation

For a finite state space $\mathcal{S}$ we can represent the state value function $v(s)$ as an array with one element for each state in $\mathcal{S}$. $v_k \to v_\pi$ as $k \to \infty$, but in practice we stop when the difference between the new and old step is small enough. The algorithm is then:

The algorithm

Synchronous updates:

1. Initialize v_{old} (e.g. $v_{\text{old}} = 0$ for all s)
2. For all $s \in \mathcal{S}$:

$$v_{new}(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a)[r + \gamma v_{old}(s')] \tag{45}$$

3. if $|v_{old}(s) - v_{new}(s)| <$ tol for all s, output v_{new} and stop.
4. Otherwise let : $v_{old} \leftarrow v_{new}$ and go back to step 2.
5. Asynchronous updates also converge to $v_\pi(s)$ as long as we keep updating all states.

Asynchronous updates: (in-place updates)

1. Start with initial v(s) (e.g. v(s) = 0)
2. For all $s \in \mathcal{S}$

$$v(s) \leftarrow \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a)[r + \gamma v(s')] \tag{46}$$

3. If changes in v are small enough we are done, otherwise back to step 2.

This is easier to implement and only need one array v(s). Notice that now the updates depends on what order we sweep through the states. It also converges to $v_\pi(s)$, often faster.

Policy improvement

Given a policy π we now need to see how to evaluate $v_\pi(s)$, is it possible to find a better policy? That is the policy π' such that:

$$v_{\pi'}(s) \geq v_\pi(s), \text{ for all } s \in \mathcal{S} \tag{47}$$

The value of taking the action a in state s ant then following the policy π afterwards is given by:

$$q_\pi(s, a) = \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{48}$$

The **greedily** action with the respect to the action values i.e.:

$$\pi'(s) = \arg\max_a q_\pi(s, a) \tag{49}$$

The policy improvement theorem

Lets consider the deterministic policy $a = \pi(s)$ (the result holds for stochastic $\pi(a|s)$ also). Then the greedy policy with respect to $v_\pi(s)$ is:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) = \max_a q_\pi(s, a) \tag{50}$$

and hence

$$q_\pi(s, \pi'(s)) = \max_a q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s) \tag{51}$$

The policy improvement Theorem

If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ fir all $s \in \mathcal{S}$, then

$$v_{\pi'}(s) \geq v_\pi(s), \text{ for all } s \in \mathcal{S} \tag{52}$$

This means that $\pi'(s) = \arg\max_a q_\pi(s, a)$ is as good as, or better than $\pi(s)$

Is this an improvement?

The policy improvement is given by

$$\pi' = \arg\max_a q_\pi(s,a) = \arg\max_a \sum_{r,s'} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{53}$$

this will be **at least** as good as π in all states. But what if there is no improvement? What if $v_\pi(s) = v_{\pi'}$ for all s? then $q_\pi(s,a) = q_{\pi'}(s,a)$ and:

$$v_\pi(s) = v_{\pi'} = q_{\pi'}(s,a) = \max_a \sum_{r,s'} p(s',r|s,a)[r + \gamma v_\pi(s')] \text{ for all } s \tag{54}$$

This is the Bellman optminality equation, so π and π' are optimal policies! So the **conclusion** is that π' will be strictly better than π, unless π is already optimal.

Policy iteration

If we start with an initial policy π:

1. **Polict evaluation (E):** Compute $v_\pi(s)$ for all s. The iterative policy evaluation
2. **Policy improvement (I):** Let $\pi'(s) = \arg\max_a q_\pi(s,a)$ for all s
3. If we have a improvement go to 1. Otherwise we have found the optimal policy π

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_2} \xrightarrow{I} \pi_2 \ldots \xrightarrow{I} pi_* \xrightarrow{E} v_* \tag{55}$$

In the case of a finite MDP this will converge within a finite number of iterations. Some details regarding the implementation: In E: we can start from the previous policy to speed up the computations and in I: if there are several a that maximize $q_\pi(s,a)$, choose one arbitrary or a stochastic policy that picks between them with uniform probability.

Value iteration

In policy iteration we do the evaluation complete before we improve the policy, this is called **generalized policy iteration**. We can also stop the evaluation after just one sweep over all states, this is called **value iteration**

If we take a look on what happens if we do one iteration of evaluation before we improve, that is for all s:

$$\begin{aligned} q_{k+1}(s,a) &= \sum_{r,s'} p(s',r|s,a)[r + \gamma v_k(s')] \\ \pi_{k+1}(s) &= \arg\max_a q_{k+1}(s,a) \\ v_{k+1}(s) &= q_{k+1}(s, pi_{k+1}(s)) \end{aligned} \tag{56}$$

Or in one equation:

$$v_{k+1}(s) = \sum_{r,s'} p(s',r|s,a)[r + \gamma v_k(s')] \tag{57}$$

With this iteration we will converge to the optimal $v_*(s)$. We can for example use in place updates instead of synchronous updates.

Value iteration and the Bellman optimality equation

- **Value iteration:**

$$v_{k+1}(s) = \max_a \sum_{r,s'} p(s', r|s, a)[r + \gamma v_k(s')] \tag{58}$$

- **Fixed point:** if $v_k(s) = v_{k+1}(s)$ for all s, then

$$v_k(s) = \max_a \sum_{r,s'} p(s', r|s, a)[r + \gamma v_k(s')] \tag{59}$$

  This is the Bellman optimality function, so $v_k(s)$ is the optimal value function

- **Optimal policy:** When converged to v_* we can find an optimal policy;

$$\pi_*(s) = \arg\max_a g_*(s, a) \tag{60}$$

$$q_*(s, a) = \sum_{r,s'} p(s', r|s, a)[r + \gamma v_*(s')] \tag{61}$$

Summary

For all methods the idea is to apply the right hand side RHS of the corresponding Bellman equation repeatedly until it convergence. All methods can also be applied to q(s,a)

Problem	**Based on**	**Algorithm**
Prediction	Bellman equation for v_π	Iterative policy evaluation
Control	Bellman equation for v_π + Greedy policy improvement	Policy iteration
Control	Bellman optimality equation	Value iteration

Table 2: example

4 Model-free prediction

Model-free prediction is the process estimating the value functions directly from observed experiences explicitly modeling the environment's dynamics. Methods like Q-learning and SARSA use trial and error interactions with the environment to learn the optimal policies.

Repetition

- **States, action and rewards:** $s \in \mathcal{S}, a \in \mathcal{A}, r \in \mathcal{R}$
- **Dynamic model :** $p(s', r|s, a)$
- **Policy:** $\pi(a|s)$ (for deterministic also $)a = \pi(s)$
- **The return:**
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{62}$$
- **State-value function:** Expected return when starting in s and following policy π
$$v_\pi(s) = \mathbb{E}\left[G_t | S_t = s\right] \tag{63}$$
- **Action-value function:** Expected return when starting in s, taking action a and then follow π
$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a] \tag{64}$$
- **Bellman equation:**
$$v_\pi = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \text{ for all } s \in \mathcal{S} \tag{65}$$
- **Policy evaluation:**
$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \text{ then } v_k(s) \to v_\pi(s) \tag{66}$$
- **Policy improvement:**
$$\begin{aligned} q_\pi(s, a) &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \\ \pi'(s) &= \arg\max_a q_\pi(s, a) \end{aligned} \tag{67}$$
- **Policy iteration:**
$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_2} \xrightarrow{I} \pi_2 \dots \xrightarrow{I} pi_* \xrightarrow{E} v_* \tag{68}$$
- **Bellman optimality equation:**
$$v_*(s) = \max_a q_*(s, a) = \max \sum_{r,s'} p(s', r|s, a)[r + \gamma v_*(s')] \tag{69}$$
- **Value iteration:** (based on Bellman optimality equation)
$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')], \text{then } v_k(s) \to v_*(s) \tag{70}$$
- **Optimal policy:**
$$\begin{aligned} q_*(s, a) &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \\ \pi_*(s) &= \arg\max_a q_*(s, a) \end{aligned} \tag{71}$$

Monte-Carlo methods

Given the problem that we throw two dice and call their sum G, what is $V = \mathbb{E}[G]$? We can calculate this using different methods:

- **By hand:**
  - Each dice has 6 sides, so we can get 36 combinations
  - There is no way to get $G = 1$, so $p(1) = 0$
  - There is one way to get $G = 2$, so $p(2) = 1/36$
  - ... and so on.
  - We finally get $\mathbb{E}[G] = \sum_{g=1}^{12} gp(g) = 7$
- **Monte-Carlo**
  - Make many throws and get **independent** observations: $G_1, G_2, \ldots, G_n$
  - Use the empirical mean to estimate $V = \mathbb{E}[G]$:

$$\hat{V}_n = \frac{1}{n} \sum_{k=1}^{n} G_k \tag{72}$$

  - This methods don't require the knowledge of how the dice works!
- **Law of large numbers:** $\hat{V}_n \to \mathbb{E}[G]$ as $n \to \text{inf}$

Bias and variance

Let $\hat{\theta}_n$ be an estimate of θ using n random samples. Since the samples are random, we got that $\hat{\theta}_n$ is a stochastic variable. This makes it possible to talk about the expected value and variance of $\hat{\theta}_n$.

- **Bias:** (unbiased if bias = 0)

$$\text{Bias } (\hat{\theta}_n) = \mathbb{E}[\hat{\theta}_n] - \theta \tag{73}$$

- **Variance:**

$$\text{Var } (\hat{\theta}_n) = \mathbb{E}[(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])^2] \tag{74}$$

- **The MSE:**

$$\text{MSE } (\hat{\theta}_n) = \mathbb{E}[(\hat{\theta}_n - \theta)^2] = \text{Var } (\hat{\theta}_n) + \text{Bias } (\hat{\theta}_n)^2 \tag{75}$$

- **Consistent** if $\hat{\theta}_n \to \theta$ as $n \to \text{inf}$

We want to estimate the expected value $V = \mathbb{E}[G]$ by using the observations: $G_1, G_2, \ldots, G_n$

$$\hat{V}_n = \frac{1}{n} \sum_{k=1}^{n} G_k \tag{76}$$

Where the bias and variance are:

$$\begin{aligned} \text{Bias}(\hat{V}_n) = \mathbb{E}[\hat{V}_n] - V = \mathbb{E}\left[\tfrac{1}{n}\textstyle\sum_{k=1}^{n} G_k\right] - V = \frac{1}{n} \sum_{k=1}^{n} \mathbb{E}[G] - V = \mathbb{E}[G] - V = 0 \\ \text{Var } (\hat{\theta}_n) = \mathbb{E}[(\hat{V}_n - \mathbb{E}[\hat{V}_n])^2] = \frac{35}{6n} \approx \frac{5.83}{n} \end{aligned} \tag{77}$$

So, as n approaches infinity the variance goes to zero.

Incremental updates

We don't have to save the information of all previous observations, wasting memory and recalculate everything for each new observation. We can store the passed information with:

$$\hat{V}_{n-1} = \frac{1}{n-1}\sum_{j=1}^{n-1} G_j \tag{78}$$

And when getting one more observation G_n:

$$\hat{V}_n = \frac{1}{n}\sum_{j=1}^{n} G_j = \frac{1}{n}(G_n + \sum_{j=1}^{n-1} G_j) = \frac{1}{n}(G_n + (n-1)\hat{V}_{n-1}) = \hat{V}_{n-1} + \frac{1}{n}(G_n - \hat{V}_{n-1}) \tag{79}$$

With this we can start from that $\hat{V} = 0, n = 0$ and then for each new observation G do this:

$$\begin{aligned} n &\leftarrow n + 1 \\ \hat{V} &\leftarrow \hat{V} + \frac{1}{n}(G - \hat{V}) \end{aligned} \tag{80}$$

We summarize it more generally like this:

$$\underset{\text{New estimate}}{\hat{V}} \leftarrow \underset{\text{Old estimate}}{\hat{V}} + \underset{\text{Step size}}{\alpha_n}\Big[\underset{\text{Target}}{G} - \underset{\text{Old estimate}}{\hat{V}}\Big] \tag{81}$$

With each step we move the estimate a bit closer to the observed "target". For the empirical mean the step size is $\alpha_n = \frac{1}{n}$. For independent and identically distributed (i.i.d) observation of G this will converge to $\mathbb{E}[G]$ if:

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty \tag{82}$$

In **non-stationary** problems with a constant $\alpha \in (0, 1)$ we "forget" the old observations. The variance will in this case not go to zero, but it can adjust to changing probabilities. The extreme cases when $\alpha = 0$ will give $\hat{V} \leftarrow \hat{V}$, meaning that we don't learn anything. In the case $\alpha = 1$ will give us that $\hat{V} \leftarrow G$ meaning that we forget all past observations.

Monte-Carlo prediction

Lets consider an episodic task (a task that terminates within a finite number of steps). Our goal is to learn $v_\pi(s)$ from experience under policy π:

$$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_T \sim \pi \tag{83}$$

The return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T \tag{84}$$

The value function:

$$v_\pi = \mathbb{E}[G_t | S_t = s] \tag{85}$$

Monte-Carlo: Estimate $v_\pi(s)$ by using the empirical mean return of many episodes instead of the expected return. With **Monte-Carlo** we do not need to know what the probability $p(s', r|a, s)$ is!

First-visit vs every-visit

- **First-visit**
  1. Sample an episode $S_0, A_0, R_1, \ldots, S_{T-1,A_{T-1},R_T}$ using policy π.
  2. **The first** time step t that state s is visited, add G_t to Returns(s).
  3. Let V(s) = average(Returns(s)).
  4. Go back to step 1.
- **Every-visit**
  1. Sample an episode $S_0, A_0, R_1, \ldots, S_{T-1,A_{T-1},R_T}$ using policy π.
  2. **Every** time-step t that state s is visited, add G_t to Returns(s).
  3. Let V(s) = average(Returns(s)).
  4. Go back to step 1.

Properties of Monte-Carlo

- **Positive:**
  - Consistent: $V(s) \to v_\pi(s)$ as $N(s) \to \infty$.
  - First-visit MC is unbiased (every-visit can be biased).
  - Does not make use of the Markov-property
- **Negative:**
  - Does not make use of the Markov-property
  - Generally high variance, reducing it may require a lot of experience.
  - Must wait until the end of episode to compute G_t and update V

Incremental updates

In the Monte-Carlo method we compute the average of all observed returns G_t which is seen in each state. With incremental updates

1. Collect a trajectory $S_0, R_1, S_1, R_2, \ldots, S_T$ following the policy π
2. For (the first/every visit) S_t compute G_t and let

$$\begin{aligned} N(S_t) &\leftarrow N(S_t) + 1 \\ V(S_T) &\leftarrow V(S_t) + \alpha_n(G_T - V(S_t)) \end{aligned} \tag{86}$$

Here the empirical mean is: $\alpha_n = \frac{1}{N(S_t)}$. For example non stationary environment we may instead is a constant α

Monte-Carlo vs Dynamic programming

Here we compare the two methods:

$$v_\pi = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s_{t+1})|S_t = s] \tag{87}$$

- **Dynamic programming:**

$$V(s) \leftarrow \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s] \tag{88}$$

  - **Bootstrapping**, each new estimate is based on a previous estimate
  - Computes the expectations exactly, but estimates since it is based on estimate $V(S_{t+1})$
  - In DP we need the model $p(s', r|s, a)$ to compute expectation.
- **Monte-Carlo**

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{89}$$

  - We do not use bootstrapping, this is because we use the full return G_t

- It is an estimate because we use the empirical mean of G_t and not $\mathbb{E}[G_t|S_t = s]$
- We don't need any model since the samples G_t can be computed from experience we collect.

Question... Can we combine bootstrapping and learning from experience?

Temporal-difference (TD) learning

We start again with the expected return for a given state:

$$v_\pi = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{90}$$

In Monte-Carlo we use the target G_t but in TD we use the TD-target $R_{t+1} + \gamma V(S_{t+1})$

$$\begin{aligned} \textbf{MC: } V(S_t) &\leftarrow V(S_t) + \alpha(G_t - V(S_t)) \\ \textbf{TD: } V(S_t) &\leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \end{aligned} \tag{91}$$

This is often called TD(0) since it is a special case of TD(λ) with $\lambda = 0$. TD methods bootstraps since the new estimate $V(S_t)$ is based on the estimate $V(S_{t+1})$. In TD we do not have a complete episode to base the update on.

TD-learning

- Initialize the estimate V (for example V(s) = for all s)
- Star in some state S
  1. Take action A according the policy $\pi(a|S)$
  2. Observe reward R and new state S'
  3. $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
  4. $S \leftarrow S'$
- (If the task is episodic, we would have to re-run the above loop for several episodes)

Note: We do not have to complete the episode before we start learning, and we can even learn while continuing the tasks.

The bias

Taking a look at the bias of the method and comparing it to the Monte-Carlo method.

- **The MC-target** G_t
  - **Unbiased** estimate of $v_\pi(S_t$
  - It is note based on previous estimates (in other words no bootstrapping)
- **The "true TD-target" :** $R_{t+1} + \gamma v_\pi(S_{t+1})$
  - **Unbiased** estimate of $v_\pi(S_t)$
  - This **cannot** be computed since we don't know $v_\pi(S_{t+1})$
- **The TD-target** $R_{t+1} + \gamma V(S_{t+1})$
  - Is a **biased** estimate of $v_\pi(S_t)$
  - Based on old estimate of V_{t+1}, bootstrapping

Comparing TD and MC

Monte-Carlo can only be used in episodic environments and have high variance but zero bias. This method converges to $v_\pi(s$ if α is decreased with a suitable rate. The Monte-Carlo method has good convergence properties even for function approximation. Another advantage is also that it is not sensitive to initial conditions. The MC method is usually more efficient in non-MDP environments. Temporal differences on the other hand can be used for both episodic and non-episodic environment and have a low variance but some bias. It also converges to $v_\pi(s)$ if α decreases with suitable rate. In comparison to MC it does not

always converge with function approximation. It is more sensitive to initial conditions but usually more efficient in MDP environments.

5 Model-free control

Model-free control is a class of algorithms that aim to learn an optimal policy directly from interacting with the environment without building an explicit model of the environments dynamics. They rely on a trial and error experience to update the estimate of the value-functions or action-value functions. Examples are Q-learning, learning the Q-function and SARSA learning the value-function.

Model free control

How do we improve the policy? One way is to take the greedy policy improvement with respect to the state value-function $v_\pi(s)$:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{92}$$

But this requires us to know the model: $p(s', r|s, a)$. If we instead take the greedy policy improvement with respect to the action value function we got:

$$\pi'(s) = \arg\max_a q_\pi(s, a) \tag{93}$$

In this case we don't need the model. So the first idea is to estimate the action-value function $q_\pi(s, a)$ instead of the state value-function $v_\pi(s)$

Example: Can we learn enough from greedy actions

- **Initial:** Q(left) = Q(right) = 0
- You open left and get the reward 2:

  Q(left) =2, Q(right)=0
- You open left and get reward 0:

  Q(left) =1, Q(right)=0
- You open left and get reward 4:

  Q(left) =3, Q(right)=0

With this approach we will never learn about what happens if we would open the right door

With that example in mind we present idea 2: make sure that we continue to explore different options!

ϵ-greedy exploration

There is a trade-off between exploiting current knowledge and exploring new options. A possible solution is to ensure that all actions have non-zero probability. We call this policy ϵ-greedy policy: If $\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}|}$ for all a and s.

- ϵ-greedy with respect to $q_\pi(s, a)$:
  - with probability 1 - ϵ choose a greedy action $\arg\max_a q_\pi(s, a)$
  - with probability ϵ choose an action at random.

Policy improvement theorem

For any ϵ-soft policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, i.e.

$$v_{\pi'}(s) \geq v_\pi(s), \quad \text{for all } s \in \mathcal{S} \tag{94}$$

The conclusion is that: policy improvement with ϵ policies will converge to the best ϵ-soft policy.

On-policy vs off-policy learning

With in-policy learning we can say that we are "learning on the job" by estimating the action value function $q_\pi(s, a)$ by running the policy π. In off-policy learning we "learn by looking over the shoulder", estimate the action value-function $q_\pi(s, a)$ while running a different policy μ. For example we learn about $q_*(s, a)$ (optimal q-function), while running a policy with more exploration.

Monte-Carlo control

We use the policy to collect trajectories: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_T$. Then estimating the state-value function by computing the average over all returns seen from each state. The incremental update is:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{95}$$

For estimating action-value function we compute the average over all returns seen from each state/action-pairs and the incremental update is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t)) \tag{96}$$

Exploration is needed!

The estimation of the state-values: $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$, converges to $v_\pi(s)$ as $N(s) \to \infty$ Estimation of action-values: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$, converges to $q_\pi(s, a)$ as $N(s, a) \to \infty$. However if the policy $\pi(s|a) = 0$ for some s and a then we will not learn this action-value! With a ϵ-soft policies we guarantee that $pi(a|s) > 0$ for all s and a.

As long as we use a ϵ-soft policy $Q(s, a)$ will converge to $q_\pi(s, a)$ as the number of sampled episodes goes to ∞. Now it is time to improve the policy!!

Monte-Carlo policy iteration

The policy evaluation: Monte-Carlo evaluation is used to get $Q = q_\pi$. The policy improvement: we let a new policy π be ϵ-greedy with respect to q_π. This will converge to the best ϵ-soft policy. For this we need infinitely many episodes to guarantee that $Q = q_\pi$ which is not possible in practice.

- At every episode:
  - Policy evaluation: Use MC to update Q.
  - Policy improvement: Let new π be ϵ-greedy with respect to Q.
  - On policy: we always update Q toward q_π for the the current policy.

Monte-Carlo control

1. Initialize Q (in other words Q(s,a) = 0 for all s and a) and let $\pi P\epsilon$-greedy(Q)
2. Sample episode using $\pi : S_0, A_0, R_1, \ldots, S_T$
3. For each state S_t and action A_t in the episode:

$$\begin{aligned} N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t)) \end{aligned} \tag{97}$$

4. Improve policy: $\pi \leftarrow \epsilon$-greedy(Q)
5. Go to step 2

Exploration

If we converge we get the best policy among the ϵ-soft policies. It is possible to gradually reduce the ϵ (but not to fast) toward zero, in order to converge to the optimal policy. Then after training we can remove the exploration by setting the $\epsilon = 0$ and thus using the greedy policy with respect to Q.

SARSA

If we return to MC and TD (aka SARSA) and compare these. TD has several advantages over MC-prediction: Lower variance, the capability to run online, without waiting to end of episode and it can be used with incomplete sequences. SARSA can apply TD to $q_\pi(s, a)$ and use ϵ-greedy policy improvements, and improve every time-step.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\ q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \end{aligned} \tag{98}$$

Estimating the state-values with given $\{S_t, R_{t+1}, S_{t+1}\} \sim \pi$ the update is:

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Target}} - V(S_t)) \tag{99}$$

Estimating the action-values with SARSA given $\{S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}\} \sim \pi$ the update will be:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\underbrace{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})}_{\text{Target}} - Q(S_t, A_t)) \tag{100}$$

SARSA control

- At every time-step:
  - **Policy evaluation:** Use SARSA to update Q
  - **Policy improvement:** Let new π be ϵ-greedy with respect to Q
  - **On-policy:** We always update Q towards q_π for current policy

SARSA-algorithm for control

- Initialize Q(s,a) in other words Q(s,a) = 0 for all s and a.
- For each episode
  1. Get initial state S
  2. Choose A from S that is ϵ-greedy with respect to Q
  3. For each step of episode:
     - Take action A and observe R, S'
     - Choose A' from S' that is ϵ-greedy with respect to Q
     - $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S', A') - Q(S,A))$
     - $S \leftarrow S', A \leftarrow A'$

Off-policy control - $\mathcal{Q}$-learning

Here the goal is the learn the action-value function $q_\pi(s,a)$ for a target policy π with experience from using the **behavior policy** μ. This is useful when we want to learn by observing how humans or other agents act. It is also useful when we want to re-use experience that is already collected from old policies. It is also useful when we want to learn the optimal $q_*(s,a)$ while following an exploratory policy.

$$q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, \underset{\sim\pi(a|S_{t+1})}{A_{t+1}}) | S_t = s, A_t = a \right] \tag{101}$$

If we consider the data collect using the behavior policy μ

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \sim \mu \tag{102}$$

Then update: let $A' \sim \pi(a|S_{t+1})$ and use the update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \gamma(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)) \tag{103}$$

We want both behavior and target policies to improve. The **target policy**, π: Greedy with respect to Q(s,a) and **behavior policy,** μ: ϵ-greedy with respect to Q(s,a). The Q-learning target is then:

$$R_{t+1} + \gamma Q(S_{t+1}, a) \tag{104}$$

Here is $A' = \arg\max_a Q(S_{t+1}, a)$. If we insert this we can rewrite the target as:

$$\begin{aligned} R_{t+1} = \gamma Q(S_{t+1}, \arg\max_a q_*(S_{t+1}, a)) = \\ R_{t+1} + \gamma \max_a q_*(S_{t+1}, a) \end{aligned} \tag{105}$$

We can compare this to the Bellman optimality equation:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_a q_*(S_{t+1}, a) | S_t = s, A_t = A] \tag{106}$$

The Q-learning theorem

Q-learning converges to the optimal action-value function $q_*(s,a)$ as $N(s,a) \to \infty$ if the step size α decreases toward 0 with a suitable rate.

With a good estimate of $q_*(s,a)$ we can find the optimal policy $\pi_*\text{greedy}(q_*)$. In practice it will often work with a constant α if it is small enough.

Q-learning for off-policy

- Initialize $Q(s, a$, in other words $Q(s, a) = 0$ for all s and a.
- For each episode:
  1. Get initial state S
  2. For each step of the episode:
     - Choose A from S that is ϵ-greedy with respect to Q
     - Take action A and observe R, S'.
     - $Q(s, a) \leftarrow Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A)$
     - $S \leftarrow S'$
  3. When the training is done. Get the target policy with respect to Q.

6 Model-based Reinforcement learning

This is a class of reinforcement learning algorithms that builds a model of the environments dynamics and reward structure to improve decision making, a policy. By approximating a model of the environment the model-based RL can plan better and faster. The model-based RL uses the model to simulate possible outcomes and optimize the actions accordingly. This approach can lead to more efficient learning, better exploration and improved generalization. Especially in environments with sparse rewards or limited data.

Repetition, if we do not have a model?

We can use experience to estimate value functions: $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ to make new estimates:

$$\text{New estimate} \leftarrow \text{Old estimate} + \text{Step size}[\text{Target} - \text{Old estimate}] \tag{107}$$

Repetition

- **TD-Learning** to estimate $v_\pi(s, a)$

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{108}$$

- **SARSA** (on-policy):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_t, A_t) - Q(S_t, A_t)] \tag{109}$$

- **Q-learning** (off-policy):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_t, A_t)] \tag{110}$$

Possible alternative: Use the experience to estimate a model of the environment.

What is a model

There are many ways to descrive what a model is, but here are two interesting quotes on the subject:

> Anything we can use to answer questions about the environment without interacting with it."
> – **Lennart Ljung and Torkel Glad**
>
> "Essentially, all models are wrong, but some are useful."
> – **George E. P. Box**

In reinforcement learning a **model** is something the **agent** can use to predict how the environment will respond to a given action. It can:

- Produce a prediction of next state and reward: $(s, a) \rightarrow (\hat{s}', \hat{r}'$

- Produce a prediction of next state $(s, a) \rightarrow \hat{s}'$

A **distribution model** is a model that provides probabilities for all possible possibilities: $p(s', r|s, a)$. This is in general more difficult than the other type of model. **Sample model** provides us with one random sample from the possible outcomes.

Why model-based?

The **advantages** with model-based RL is that the agent can learn a model with supervised learning that have more established methods. It has been used in classical control with success and for a wide range of different tasks. It is possible to use prior knowledge to reduce the number of unknown parameters. There is a similarity between model building and System identification in classical control. If feedback is used a perfect model is not typically not needed. With a model it is possible to reason about model uncertainty and offers more explainability. Transfer learning is an advantage were pre-existing knowledge and experience from one task is used to enhance performance in another related task. Examples of **disadvantages** with model-based RL are that estimated model might not capture the dynamics of the environment very well and the result may be a bad policy. By learning model and then construct the value function we have introduced two sources of approximation error. It is sometimes the case that the value function and optimal policy can be much simpler than an accurate model.

How to do model-based RL

This can be done in many different ways but the most common methods are:

1. Learn a model of the environment (Supervised learning / System identification)
2. Find the optimal policy for that model:
    - Classical control methods
    - Dynamic programming
    - Q-learning, SARSA etc with the samples produced with the model

There are some possible extensions, for example to use some uncertainty measure and find a that is robust to this uncertainty. Here follows a list of examples models:

- Table lookup
- Linear dynamic system
- Gaussian process model
- Linear regression model
- Deep neural network

Table lookup model

The model is $p(s', r|s, a)$ explicitly. Training data is collected $S_0, A_0, R_1, S_1, \ldots, R_T, A_T$ and used to estimate:

$$\hat{p}(s', r|s, a) = \frac{1}{N(s, a)} \text{ (Number of times } (s', r) \text{ came after } (s, a)) \tag{111}$$

For a good model at (s, a) we have to have seen this state-action pair many times. That is, exploration is still a very important part!

Model-based and sample-based planning

With a given estimated **model** $\hat{p}(s', r|s, a)$ we can solve the MDP and use some of our favorite methods to do so: value iteration or policy iteration. With **sample-based** planning we use the model to generate samples for us. This a very simple but powerful approach. We then use sample experience from the model:

$$S_{t+1}, R_{t+1} \sim p(s', r|S_t, A_t) \tag{112}$$

Then we apply *model-free* RL to these model generated samples, with for example:

- Monte-Carlo control
- SARSA
- Q-learning

This can be very useful when sampling form the real world is very expensive.

Learning and planning

In *model-free* learning we don't have a model and must learn from real experience to train/learn our value functions. With *Model-based* RL and *sample-based* planning we learn a model from **real** experience and then learn the value function (and or policy) from the simulated experience. *Dyna* or integrated learning and planning learns a model from experience and then learns the value function (and or policy) from **real** and **simulated** experience.

Dyna

Dyna combines the model-based learning and direct learning from real experience. This enables the agent to learn and update its knowledge about the environment while at the same time improving its policy. The key concept is called *experience replay* which allows the agent to learn from past experience and simulated experiences from the model which gives efficient exploration and decision making.

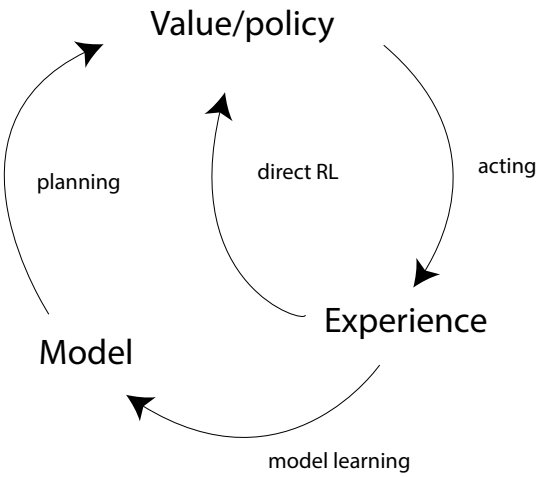

Figure 1: Dyna architecture

Tabular Dyna-Q for deterministic environments

With model learning, if we observe: $S_t, A_t, R_{t+1}, S_{t+1}$ then :

$$\text{Model}(S_t, A_t) \leftarrow R_{t+1}, S_{t+1} \tag{113}$$

And with Q-learning form both the simulated and real experience S, A, R, S' we can do:

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)] \tag{114}$$

Tabular Dyna-Q

Initialize $Q(s,a)$ and Model(s,a) for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(\int)$ Loop forever:

1. $S \leftarrow$ current (nonterminal) state
2. $A \leftarrow \epsilon$-greedy(S,Q)
3. Take action A; observe resultant reward R and state S'
4. $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
5. Model$(S_t, A_t) \leftarrow R_{t+1}, S_{t+1}$ (assuming that the environment is deterministic)
6. Loop repeat n times:
   - $S \leftarrow$ random previously observed state
   - $A \leftarrow$ random action previously taken in S
   - $R, S' \leftarrow$ Model(S,A)
   - $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

DynaQ vs DynaQ+

DynaQ is model biased, it (Naively) inherently assumes that the learned model is an accurate description of the real environment. So if the environment changes it takes time to adapt or in some cases it might take long time to discover a more efficient way to solve the task. DynaQ+ is a possible solution to this. The idea is to when simulating experience add a reward for exploring states not seen before: Specifically, if $R, S' \leftarrow$ Model(S,A) then use:

$$R + \kappa\sqrt{\tau} \tag{115}$$

in the Q-update. Here τ is the number of time steps since (S,A) was last visited in the real experience. We encourage the agent to visit states it has not seen for a long time. This extra exploration may cost some extra, but the curiosity can help if the real world changes for example.

Exploration vs Exploitation trade-offs

There is always a trade off in RL when it comes to exploration and exploitation. When we explore we gather more information and can find better policys but this cost us reward while we are doing this and there is always the possibility that we don't find a better policy either. With exploitation we make the best decisions from the knowledge we currently have and don't waste "energy" on trying to find better ways to do it. Some explorations ideas we seen so far are:

- ϵ-greedy
- DynaQ+
- Optimistic initialization

7 Model-based RL in continuous action and state spaces

Now we will start talking about how we can extend the ideas that have been presented so far to continuous state and action spaces. In this case we can't store the Q-values for all possible state/action-pairs in a table any more and we will therefore use function approximations instead:

$$Q(s,a;\mathbf{w}) \approx q_\pi(s,a) \tag{116}$$

There are many ways to do model-based RL in these situations.

A system identification approach

If we assume that the dynamics of the environment can be accurately described as:

$$s_{t+1} = f(s_t, a_t; \mathbf{w}) \tag{117}$$

where $\mathbf{w}$ is a vector of unknown parameter/weights. Here the $f(\cdot)$ can be a neural network, a basis expansion or a linear system. Data/experience is to be collected $(S_0, A_0, S_1, \ldots, A_{t-a}, S_T)$, learning the model using the prediction error method:

$$\hat{\mathbf{w}} = \arg\max_{w} \sum_{t=0}^{T-1} ||S_{t+1} - f(S_t, A_t; \mathbf{w})||_2^2 \tag{118}$$

Finding a policy can be done using *classic control* or *model-free* RL using the samples from the model.

8 Function approximation

Function approximation is a technique used for estimating the value function or policies for large or continuous state-action spaces. It uses various machine learning models, such as decision trees, neural networks or linear regression to generalize and predict values or optimal actions from a limited set of experience that have been observed. Using function approximation, the agent can scale to more complex environments and solve problems that are not suited for making tables of in tabular methods. When applying function approximation one of the challenges are to balance the trade-off between generalization and accuracy, making sure that the model accurately represent the underlying structure of the environment, but still being efficient to learn from limited amount of data.

Repetition

What are we trying to do? We have a *Prediction Problem*prediction problem, for a given policy π, we want to find the state-value function:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}|S_t = s)] \tag{119}$$

We also have a *Control Problem* to find the optimal policy: π_*. We solve this by using *Policy Iteration*: for a given policy π evaluate $v_\pi(s)$ and do policy improvement:

$$\pi'(s) = \arg\max_{a} q_\pi(s, a) \tag{120}$$

Where $q_\pi = \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')]$

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_2} \xrightarrow{I} \pi_2 \ldots \xrightarrow{I} pi_* \xrightarrow{E} v_* \tag{121}$$

So far we have only considered finite state and action spaces. In these cases we can represent $v_\pi(s)$ as an array with $|\mathcal{S}|$ elements and we can also represent $q_\pi(s, a)$ as a table with $|\mathcal{S}| \times |\mathcal{A}|$ elements. Example with the gridworld:

	UP	DOWN	LEFT	RIGHT
0	X	X	X	X
1	X	X	X	X
⋮	⋮	⋮	⋮	
15	X	X	X	X

Table 3: example

We find an estimate of $Q(s, a)$ by starting to fill in and then update the table with the estimates for each state/action pair that we see. But **How do we update the estimates of the estimated V or Q?** If we have observed:

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \tag{122}$$

We update the element that corresponds to (S_t, A_t) by using:

$$\text{New estimate} \leftarrow \text{Old estimate} + \text{Step size}[\text{Target} - \text{Old estimate}] \tag{123}$$

True vf	**Target**		
For v_pi	$v_\pi(s) = \mathbb{E}[G_t \vert S_t = s]$	MC-target	$G_t = R_{t+1} + \gamma R_{t+2+...}$
For v_pi	$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1)} \vert S_t = s]$	TD-target	$R_{t+1} + +\gamma(V(S_{t+1})$
For q_π	$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}.A_{t+1}) \vert S_t = s, A_t = a]$	SARSA-target	$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$
For optimal q_*	$q_*(s,a) = \mathcal{E}[R_{t+1} + \gamma \max_a q_*(S_{t+1}) \vert S_t = s, A_t = a]$	Q-learning target	$R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$

Table 4: example

Function approximation

If we now consider a very large or continuous state space: $\mathcal{S}$. The function approximation:

$$\begin{aligned} \hat{v}(s, \mathbf{w}) &\approx v_\pi(s) \\ \hat{q}(s, a, \mathbf{w}) &\approx q_\pi(s, a) \end{aligned} \tag{124}$$

Here is $\mathbf{w}$ the unknown parameters or weights of the what we need to estimate. The number of parameters in $\mathbf{w}$ is typically smaller than the number of states. Some generalization:

- A change in $\mathbf{w}$ may affect the value estimates of many or all states in $\mathcal{S}$
- We get an estimate value even for states that we haven't seen before.

There are many ways to approximate a function: desicion trees, kNN, Gaussian process, linear regression and so on. We will here focus on differentiable function approximators and especially: Linear combination of features and neural networks. Good to know is that in Reinforcement learning is:

- The data is not independent and identically distributed (IID)
- Data distribution depends on the policy
- Every time the policy is improved, the distribution changes
- Hence, the data is *Non-stationary*

Linear function approximation

If we consider the approximation of $v_\pi(s)$. We let a *feature vector* $\mathbf{x}(s) \in \mathbb{R}^d$ associate with each $s \in \mathcal{S}$. The parameters of weights: $\mathbf{w} \in \mathbb{R}^d$

$$\mathbf{x}(s) = \begin{bmatrix} x_1(s) \\ \vdots \\ x_d(s) \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix} \tag{125}$$

Then we can estimate our $\hat{v}(s, \mathbf{w})$ by computing:

$$\hat{v}(s, \mathbf{w}) = \sum_{i=1}^{d} w_i x_i(s) = \mathbf{w}^T \mathbf{x}(s) = \mathbf{x}(s)\mathbf{w} \tag{126}$$

Here are some examples of features (sometimes called *basis functions*):

- Physical consideration
- Polynomials
- Fourier basis
- ...

Stochastic gradient descent (SGD)

In gradient descent we let $J(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}$ be a scalar valued function with a vector input. The aim is to find a set of weights $\mathbf{w}$ that minimize $J(\mathbf{w})$. We do this by using the gradient:

$$\nabla J(\mathbf{w}) := \frac{\partial J}{\partial \mathbf{w}} := \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_d} \end{bmatrix} \in \mathbb{R}^d \tag{127}$$

At a point $\overline{\mathbf{w}}$, $J(\mathbf{w})$ is decreasing fastest in the direction $-\nabla J(\overline{\mathbf{w}})$. To minimize $J(\mathbf{w})$ we start with an initial guess of our weights: $\mathbf{W}_0$ and then we start moving in the direction of maximum descent with a step size α:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \nabla J(\mathbf{w}_k) \tag{128}$$

This will push $\mathbf{w}_k$ into a local minimum.

Value function approximation with GD

We want to apply gradient descent to approximate $v_\pi(s)$ with $\hat{v}(s, \mathbf{w})$. The idea here is to minimize:

$$J(\mathbf{w}) = \frac{1}{2}\mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2] \tag{129}$$

Where the gradient is:

$$\nabla J(\mathbf{w}) = \frac{1}{2}\mathbb{E}_\pi[\nabla(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2] = -\mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}))\nabla\hat{v}(s, \mathbf{w})] \tag{130}$$

And we express the gradient descent with:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha\mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}))\nabla\hat{v}(s, \mathbf{w})] \tag{131}$$

(Here we find a problem: What if we cannot compute the expected value?). The full gradient descent with all data:

$$\mathbf{w} = \mathbf{w} - \alpha\mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}))\nabla\hat{v}(s, \mathbf{w})] \tag{132}$$

If we instead draw a sample S form the on-policy distibution and use:

$$\mathbf{w} = \mathbf{w} - \alpha\mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla\hat{v}(S, \mathbf{w})] \tag{133}$$

Then the expected update is equal to a full gradient update ("On average we update in the gradient descent direction")

Linear function approximation

In the case of linear functions the update looks like this

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s) \Rightarrow \nabla\hat{v}(s, \mathbf{w}) = \mathbf{x}(s) \tag{134}$$

So the update will be given by:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[v_\pi(S) - \hat{v}(s, \mathbf{w})]\mathbf{x}(S) \tag{135}$$

In the case with a linear function approximation $J(\mathbf{w})$ only has one optimum and thus any methods that finds a local optimum also finds the global optimum, convex problem.

Model free prediction

We want to estimate $v_\pi(s)$ without a known model. By following a policy π to get some data: $S_0, A_0, R_1, S_1, A_1, \ldots,$ and for each step we update or parameters or weights $\mathbf{w}$.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha[(v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}))\nabla\hat{v}(S_t, \mathbf{w})] \tag{136}$$

The problem is that: we don't know $v_\pi(S_t)$. One idea is to, as in the tabular case, replace $v_\pi(S_t)$ with a target U_t

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha[(U_t - \hat{v}(S_t, \mathbf{w}))\nabla\hat{v}(S_t, \mathbf{w})] \tag{137}$$

If U_t is unbiased estimate of $v_\pi(S_t)$ for all t, then $\mathbf{w}$ will converge to a local optimum, if assuming α decreases according to the usual assumptions. The targets we have discussed earlier are: Monte-Carlo and TD.

Monte-Carlo prediction with function approximation

The Monte-Carlo target is: $G_t = R_{t+1} + \gamma R_{t+1} + \ldots$, which is an unbiased estimate of $v_\pi(S_t) = \mathbb{E}[G_t|S_t]$. The updates will look like this:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(G_t - \hat{v}(s, \mathbf{w}))\nabla\hat{v}(S_t, \mathbf{w}) \tag{138}$$

That will converge to a local optimum. With linear function approximation it will converge to the *global* optimum. And as before, we have to wait until the end of the episode before G_t can be computed. G_t can have high variance since it is a very noisy estimate of $v_\pi(S_t)$, it also means that it can take very long to converge.

TD-prediction with function approximation

The TD-target is expressed with: $U_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})$ and is *biased* since it is based on the estimate $\hat{v}$. Hence the convergence cannot be guaranteed in general but it often works fine. In the case of linear function approximations i will converge to global optimum. As in the tabular case with TD we will often learn faster than with the MC approach and we also don't need to wait until we have finished an episode. This methods is some times called **semi-gradient**

$$\nabla(v_\pi(S_t) - \hat{v}(S_{t+1}, \mathbf{w}))^2 = -2(v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}))\nabla\hat{v}(S_t, \mathbf{w}) \tag{139}$$

Here we use the fact that $v_\pi(S_t)$ is independent of $\mathbf{w}$. But we replace $v_\pi(S_t)$ with the target $R_{t+1}+\gamma\hat{v}(S_{t+1}\mathbf{w})$, not taking into account that a change in $\mathbf{w}$ also changes the target.

Model-free control with function approximation

In this chapter we consider the case when the action space $\mathcal{A}$ is finite and small. The idea is to predict using function approximation $\hat{q}(s, a, \mathbf{w} =\approx q_\pi(s, a)$ and policy improvement which is ϵ -greedy with respect to $\hat{q}(s, a, \mathbf{w})$. Notice that if $\mathcal{A}$ is large the policy improvement can be hard to do. Here we will only consider *on-policy* methods like MC and SARSA since an *off-policy* methods like Q-learning have several issues when using function approximation.

Action-values with function approximation

We aim to approximate $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$ with the idea that minimizing $J(\mathbf{w}) = \mathbb{E}[(q_\pi(S_t, A_t)-\hat{q}(s, a, \mathbf{w}))^2]$ with stochastic gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(q_\pi(S_t, A_t) - \hat{q}(s, a, \mathbf{w}))\nabla\hat{q}(S_t, A_t, \mathbf{w}) \tag{140}$$

Replacing $q_\pi(s, a)$ with the estimated target U_t (e.g. MC or TD) and as in the tabular case we must continue to explore by using a policy that is ϵ-greedy with respect to $\hat{q}(s, a, \mathbf{w})$. Linear function approximations will

converge for both SARSA and MC close to the optimal approximation of q. In the case of nonlinear function approximations there is no guarantee of convergence, but will often work.

SARSA with function approximation

- Choose function approximation $\hat{q}(s, a, \mathbf{w})$ and an initial $\mathbf{w}$
- For each episode:
  1. Get initial state S
  2. Choose A from S in other words: ϵ-greedy with respect to $\hat{q}$
  3. For each step in the episode:
     - Take action A and observe R, S'
     - Choose action A' from S', ϵ-greedy with respect to $\hat{q}$
     - Let $U_t = \begin{cases} R, & \text{if } S' \text{ is terminal} \\ R + \gamma \hat{q}(S', A', \mathbf{w}), & \text{otherwise} \end{cases}$
     - $\mathbf{w} \leftarrow \mathbf{w} + \alpha[U_t - \hat{q}(S, A, \mathbf{w})]\nabla \hat{q}(S, A, \mathbf{w})$
     - $S \leftarrow S', A \leftarrow A'$