Reinforcement learning

Linus Falk

April 10, 2023

Contents

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.