Deep Learning
Summer semester ‘24

8. Introduction to Reinforcement Learning

Slide credit: Slides in parts adapted from the lecture at the PRL @ FAU Erlangen-Nürnberg (K. Breininger, T. Würfl, A. Maier, V. Christlein).

Outline

1. Basics of Reinforcement Learning

2. Policies and value functions

3. Model-free control

What is Reinforcement Learning?

The fundamental challenge in artificial intelligence and machine learning is learning to make good decisions under uncertainty.

– E. Brunskill

Reinforcement learning is the idea of being able to assign credit or blame to all the actions you took along the way while you were getting that reward signal.

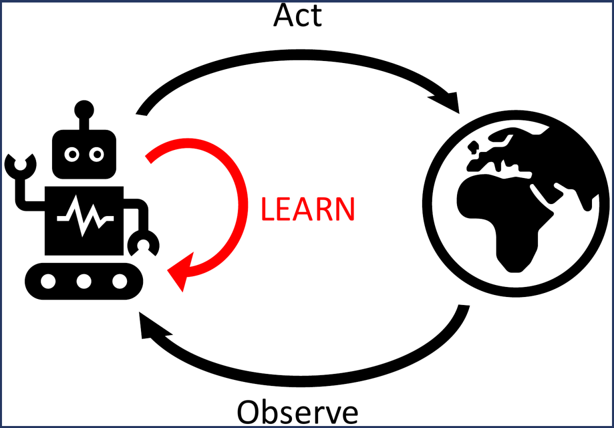
– J. Dean

Reinforcement learning is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal.

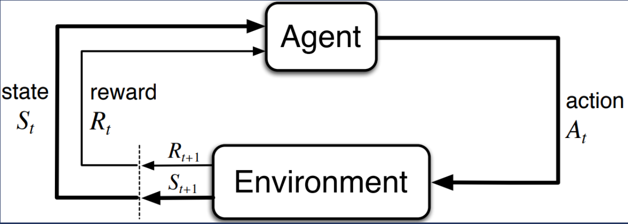
– R. Sutton

- Repeated **interactions** with the world;
- Do **not** know in advance how the world works.

Agent & world



Markov decision process - 1



- Sequence of **discrete** time steps $t = 0, 1, 2, \ldots;$
- At each time step:
  - Receive representation of **state** S_t;
  - Execute **action** A_t;
  - Obtain **reward** R_t and reach a **new state** S_{t+1}.
- Prolonged interaction between agent and environment generates a **trajectory**

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$$

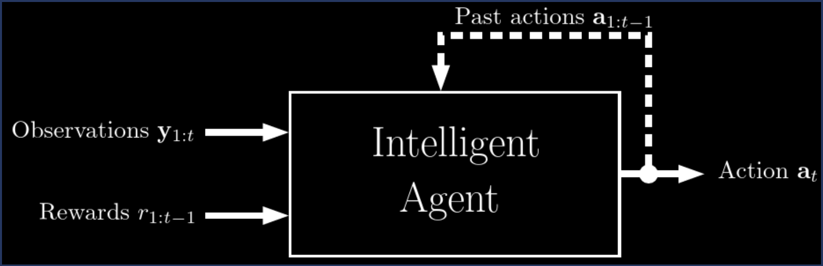
Markov decision processes - 2

Markov decision process

A **Markov decision process (MDP)** is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \iota, \gamma \rangle$, where

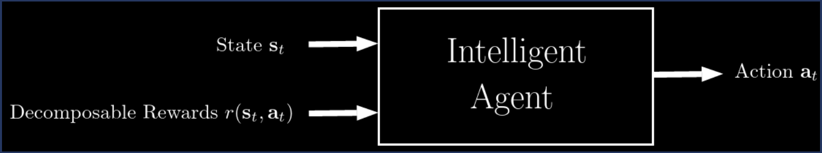
- $\mathcal{S}$ is the set containing all states;
- $\mathcal{A}$ is the set containing all actions;
- $\mathcal{R}$ is the reward function;
- $\mathcal{P}$ is the transition function;
- ι is the probability distribution over initial states;
- $\gamma \in [0, 1)$ is the discount factor.

Full reinforcement learning problem



- Agent can only test $p(\mathbf{y}_{t+1}|\mathbf{y}_{1:t}, \mathbf{a}_{1:t})$ to obtain rewards $r_t = r(\mathbf{y}_{1:t}, \mathbf{a}_{1:t})$.

Assumption: Markovian observable state



Markov decision process

- Observe the state $\mathbf{s}_t = \mathbf{b}_t$ directly and remain Markovian $p(\mathbf{s}_{t+1}|\mathbf{s}_{1:t}, \mathbf{a}_{1:t}) = p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$.

Reinforcement learning problems

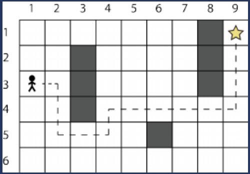
- There are different dichotomies for RL problems:
  - Discrete ($\mathcal{S}$ and $\mathcal{A}$ are finite sets);
  - Continuous ($\mathcal{S}$ and/or $\mathcal{A}$ are infinite sets).
  - Deterministic ($\mathcal{R}$ and $\mathcal{P}$ are functions);
  - Stochastic ($\mathcal{R}$ and/or $\mathcal{P}$ are probability distributions).
  - Discrete time;
  - Continuous time.
  - Fully observable;
  - Partially observable.
  - …

The three approaches of RL

- **Value-based** methods
  - Obtain an estimate of the (action-)value function and **use it to derive the policy**;
  - Commonly used in problems with **discrete actions**.
- **Policy-based** methods
  - Obtain the policy **explicitly** by maximizing a performance measure;
  - Commonly used in **continuous problems** with **low-dimension state and action spaces**.
- **Actor-critic** methods
  - Obtain **both** an estimate of the (action-)value function and an explicit approximation of the policy;
  - These two functions are optimized **jointly**;
  - Commonly used in **continuous problems** with **large state and action spaces**, e.g., deep RL.

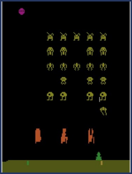
Value-based methods

- Commonly used for problems with **discrete actions**;



(a) Grid-world

(b) Stock trading

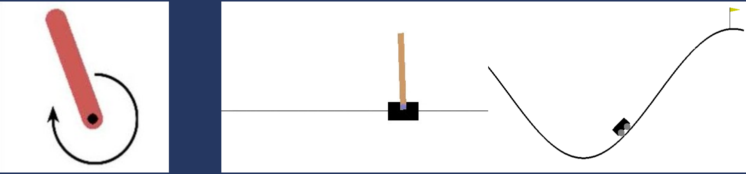


(c) Videogames

- **Classic** value-based approaches: Q-Learning, SARSA, Fitted Q-Iteration (FQI), Least Squares Policy Iteration (LSPI), ...;
- **Deep** value-based approaches: Deep Q-Network (DQN), Double DQN, Distributional DQN, Rainbow, ...

Policy-based methods

- Commonly used for **low-dimensional problems** with **continuous actions**;



(a) Pendulum (b) Cart-pole (c) Mountain car

- **Examples:** REINFORCE, GPOMDP, Reward Weighted Regression (RWR), Relative Entropy Policy Search (REPS), ….

Actor-critic methods

- Used for **high-dimensional problems** with **continuous actions**;

(a) MuJoCo (b) Anymal robot (c) Manipulation

- **Examples:** Deep Deterministic Policy Gradient (DDPG), Trust-Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), ...

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Different types of MDPs

- **Discrete** (or **finite**) MDP:
  - **Discrete** state space $\mathcal{S}$
  - **Discrete** action space $\mathcal{A}$
- **Continuous** MDP:
  - **Continuous** state space $\mathcal{S}$
  - **Discrete** or **continuous** action space $\mathcal{A}$

- **Deterministic** MDP:
  - **Deterministic** transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$;
  - Reward function is $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- **Stochastic** MDP:
  - **Stochastic** transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to$ a probability;
  - Reward function is $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$

Dynamics of MDPs

- The **dynamics** of an MDP is defined by a probability

$$\mathcal{P}(s', r|s, a) \triangleq P\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{1}$$

  for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, $r \in \mathcal{R}$, and $s' \in \mathcal{S}$.
- Being a probability

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} \mathcal{P}(s', r|s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2}$$

- Note that for **deterministic** MDPs, if an action $a \in \mathcal{A}$ executed in a state $s \in \mathcal{S}$, leads to a reward $r \in \mathcal{R}$ and next state $s' \in \mathcal{S}$

$$\sum_{s'' \in \mathcal{S} \setminus \{s'\}} \sum_{r' \in \mathcal{R} \setminus \{r\}} \mathcal{P}(s'', r'|s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$$

$$\mathcal{P}(s', r|s, a) = 1$$

Dynamics of MDPs

$$\mathcal{P}(s', r|s, a) \triangleq P\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{3}$$

- Dynamics only depend on current state $s \in \mathcal{S}$ and executed action $a \in \mathcal{A}$;
- This is known as **Markov property**;
- Equation 3 enables obtaining **all** information about the environment

$$\mathcal{P}(s'|s, a) \triangleq P\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} \mathcal{P}(s', r|s, a)$$

$$\mathcal{R}(s, a) \triangleq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} \mathcal{P}(s', r|s, a)$$

$$\mathcal{R}(s, a, s') \triangleq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \frac{\sum_{r \in \mathcal{R}} r \cdot \mathcal{P}(s', r|s, a)}{\mathcal{P}(s'|s, a)}.$$

Episodes

- Interaction between an agent and the environment is modeled through **episodes**;
- The agent starts an episode in one of the possible **initial states** sampled according to the initial state distribution ι;
- The episode **ends** when:
  - The agent has performed an arbitrary maximum number of steps, known as **horizon**;
  - …OR, the agent has reached an **absorbing state**, i.e., a state where all actions lead to itself, and the reward is always 0.
- Episodes are used to **train** agents and **evaluate** their behavior.

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Returns

Return

The sum of rewards collected after T steps is called **return**

$$J_t \triangleq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_{t+T}; \tag{4}$$

- Usually, the return is computed over a whole episode;
- The **discounted return** is

$$J_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_{t+T}; \tag{5}$$

- $\gamma = 0$: the agent is **myopic**, only cares about immediate reward;
- $\gamma = 1$: the agent cares equally about **all** rewards, even the ones collected after infinite steps;
- The case of $0 < \gamma < 1$ is the most common one;
- If $T = \infty$, $0 < \gamma < 1$, and the reward R is a positive constant number, the return is a geometric series converging to $\frac{R}{1-\gamma}$.

Why discounting?

Most MDPs are discounted. Why?

- **Mathematically** convenient to discount rewards;
- Avoids **infinite returns** in cyclic MDPs;
- **Uncertainty** about the future may not be fully represented;
- **Example:** if the reward is financial, **immediate** rewards may earn more interest than **delayed** rewards;
- **Animal/human behavior** shows preference for immediate reward;
- It is sometimes possible to use **undiscounted** MDPs (i.e., $\gamma = 1$), e.g. if all sequences terminate.

Rewards

- The return has a **recursive** nature:

$$\begin{aligned} J_t &\triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \\ &= R_{t+1} + \gamma J_{t+1}. \end{aligned}$$

- The **goal** of Reinforcement Learning is obtaining a behavior that maximizes the discounted return J_t starting from any time step t;
- Thus, the reward function expresses **desired** (or **undesired**) behavior that should be **reinforced** (or **avoided**);
- The reward function is designed by a **human expert** according to what he/she wants to obtain
  - Reward functions where reward often changes are called **dense**, e.g., distance from a goal state;
  - Reward functions where reward is almost constant are called **sparse**, e.g., always 0 except for 1 upon reaching a goal state.

Goals and rewards

- Is a **scalar** reward an adequate notion of a **purpose**?
  - **Sutton hypothesis**: All of what we mean by **goals** and **purposes** can be well thought of as the **maximization** of the cumulative sum of a received scalar signal (reward);
  - Still an open problem, but its current solution is so **simple** and **flexible** we have to disprove it before considering anything different.
- A goal should specify **what** we want to achieve, **not how** we want to achieve it;
- The same goal can be specified by (infinite!) **different reward functions**;
- A goal must be **outside** the agent's direct control – thus outside the agent;
- The agent must be able to **measure** success:
  - **explicitly**;
  - **frequently** during its lifespan.

Outline

1. Basics of Reinforcement Learning

2. Policies and value functions

3. Model-free control

Policies

- What do we mean by **behavior** of the agent?
- Agents interact on the environment by executing **actions** in each state;
- A **policy** is a probability distribution that, given a state $s \in \mathcal{S}$, computes the probability of executing any action $a \in \mathcal{A}$;
- Policies are commonly denoted π, e.g., $\pi(a|s)$ is the probability of executing action a in state s.
- Reinforcement Learning aims at obtaining the policy π that **maximize** the return J_t obtained from any state $s \in \mathcal{S}$;
- The policy maximizing the return from every state is called **optimal** policy and often denoted π^*.

Value functions

Value function

The value function $V^\pi(s)$ of a state s under a policy π is the **expected** discounted return when starting in s and following π thereafter:

$$V^\pi(s) \triangleq \mathbb{E}_\pi \left[J_t | S_t = s\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \tag{6}$$

Action-value function

The action-value function $Q^\pi(s, a)$ of taking an action a in a state s under a policy π is the **expected** discounted return when starting in s, executing action a, and following π thereafter:

$$Q^\pi(s, a) \triangleq \mathbb{E}_\pi \left[J_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \tag{7}$$

Optimal value functions

- The value functions induced by an optimal policy π^* are called **optimal** value functions;
- Optimal value function: $V^*(s) \triangleq \max_\pi V^\pi(s)$;
- Optimal action-value function: $Q^*(s,a) \triangleq \max_\pi Q^\pi(s,a)$;
- We have the following relation: $V^*(s) = \max_{a \in \mathcal{A}} Q^{\pi^*}(s,a)$;
- Optimal value functions can be used to compute optimal policies, by selecting $\operatorname{argmax}_{a \in \mathcal{A}} Q^*(s,a), \forall s \in \mathcal{S}$.

Optimal policy

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Properties of optimal policies

- Value functions define a partial ordering over policies $\pi \geq \pi'$ if $V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in \mathcal{S}$;
- There exists an **optimal policy** π^* that is better than or equal than all other policies $\pi^* \geq \pi, \forall \pi$;
- **All** optimal policies induce the **optimal value function**, $V^{\pi^*}(s) = V^*(s)$;
- **All** optimal policies induce the **optimal action-value function**, $Q^{\pi^*}(s,a) = Q^*(s,a)$;
- There is always a **deterministic** optimal policy for any MDP.

The **deterministic optimal** policy can be found by **maximizing** over $Q^*(s,a)$:

$$\pi^*(a|s) = \begin{cases} 1 \text{ if } a = \text{argmax}_{a \in \mathcal{A}} Q^*(s,a) \\ 0 \text{ otherwise} \end{cases}$$

Bellman equation - 1

Julius-Maximilians-UNIVERSITÄT WÜRZBURG

Bellman equation for V^π

Value functions have the recursive relationship shown for rewards:

$$\begin{aligned} V^\pi(s) &\triangleq \mathbb{E}_\pi\left[J_t | S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma J_{t+1} | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} \mathcal{P}(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi[J_{t+1}|S_{t+1} = s']\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} \mathcal{P}(s', r|s, a)\left[r + \gamma V^\pi(s')\right], \forall s \in \mathcal{S}; \end{aligned} \tag{8}$$

- Equation 8 is known as **Bellman equation**;
- Relation between the value of a state and the values of successor states.

Bellman equation - 2

Bellman equation for Q^π

$$
\begin{aligned}
Q^\pi(s,a) &\triangleq \mathbb{E}_\pi\left[J_t | S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma J_{t+1} | S_t = s, A_t = a\right] \\
&= \sum_{s',r} \mathcal{P}(s', r | s, a)\left[r + \gamma \mathbb{E}_\pi[J_{t+1} | S_{t+1} = s']\right] \\
&= \sum_{s',r} \mathcal{P}(s', r | s, a)\left[r + \gamma V^\pi(s')\right] && (9) \\
&= \sum_{s',r} \mathcal{P}(s', r | s, a)\left[r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s', a')\right] && (10)
\end{aligned}
$$

Bellman equation - 3

- Consider a finite MDP;
- Bellman equation can be expressed in **matrix** form

$$V = P^R R + \gamma P^V V \tag{11}$$

where V and R are column vectors with one entry per state, P^R is the rewards probability matrix, and P^V is the transition matrix

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} P^R_{11} & \dots & P^R_{1n} \\ \vdots & \ddots & \vdots \\ P^R_{n1} & \dots & P^R_{nn} \end{bmatrix} \begin{bmatrix} R(1) \\ \vdots \\ R(n) \end{bmatrix} + \gamma \begin{bmatrix} P^V_{11} & \dots & P^V_{1n} \\ \vdots & \ddots & \vdots \\ P^V_{n1} & \dots & P^V_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} \tag{12}$$

Solving the Bellman Equation

- The Bellman equation is a **linear** equation;
- It can be solved **directly**

$$
\begin{aligned}
V &= P^R R + \gamma P^V V \\
(I - \gamma P^V) V &= P^R R \\
V &= (I - \gamma P^V)^{-1} P^R R
\end{aligned}
$$

- The computational complexity is $O(n^3)$ for n states;
- Direct solution is only possible for **small** MDPs – e.g., direct inversion with Gaussian elimination, matrix decomposition (QR, Cholesky, etc.), iterative solutions (e.g., Krylov-subspaces);
- There are many iterative methods for large MDPs:
  - Dynamic Programming;
  - Monte-Carlo Evaluation;
  - Temporal Difference Learning.

Bellman optimality equation - 1

Theorem: Bellman's principle of optimality

"An **optimal** policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." (R.E. Bellman, Dynamic Programming, 1957)

Any policy (i.e., also the optimal one) must satisfy the self-consistency condition given by the Bellman equation.

Bellman optimality equation - 2

Bellman optimality equation for V^*

$$
\begin{aligned}
V^*(s) &\triangleq \max_{a \in \mathcal{A}} Q^*(s,a) \\
&= \max_a E_{\pi^*}\left[J_t | S_t = s, A_t = a\right] \\
&= \max_a E_{\pi^*}\left[R_{t+1} + \gamma J_{t+1} | S_t = s, A_t = a\right] \\
&= \max_a E_{\pi^*}\left[R_{t+1} + \gamma V^*(s') | S_t = s, A_t = a\right] \\
&= \max_a \sum_{s',r} \mathcal{P}(s', r | s, a)\left[r + \gamma V^*(s')\right], \forall s \in \mathcal{S};
\end{aligned} \tag{13}
$$

Bellman optimality equation - 3

Bellman optimality equation for Q^*

$$\begin{aligned} Q^*(s,a) &\triangleq \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a\right] \\ &= \sum_{s',r} \mathcal{P}(s', r|s,a)\left[r + \gamma \max_{a'} Q^*(s', a')\right], \forall s \in \mathcal{S}, a \in \mathcal{A}; \end{aligned} \quad (14)$$

Bellman operator - 1

Bellman operator for V^π

The **Bellman operator** for V^π is a mapping $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$

$$(T^\pi V^\pi)(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma V^\pi(s')\right] \tag{15}$$

- Using the Bellman operator, the Bellman equation can be **compactly** written as

$$T^\pi V^\pi = V^\pi; \tag{16}$$

- **Linear equation** in V^π and T^π;
- If $0 < \gamma < 1$, then T^π is a **contraction** w.r.t. the maximum norm;
- V^π is a **fixed point** of the Bellman operator T^π.

Bellman operator - 2

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Bellman operator for Q^π

The **Bellman operator** for Q^π is a mapping $T^\pi : \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$

$$(T^\pi Q^\pi)(s,a) = \sum_{s',r} \mathcal{P}(s',r|s,a)\left[r + \gamma \overbrace{\sum_{a'\in\mathcal{A}} \pi(a'|s')Q^\pi(s',a')}^{V^\pi(s')}\right] \tag{17}$$

- Using the Bellman operator, the Bellman equation can be **compactly** written as:

$$T^\pi Q^\pi = Q^\pi; \tag{18}$$

- **Linear equation** in Q^π and T^π;
- If $0 < \gamma < 1$, then T^π is a **contraction** w.r.t. the maximum norm;
- Q^π is a **fixed point** of the Bellman operator T^π.

Bellman optimality operator - 1

Bellman optimality operator for V^*

The **Bellman optimality operator** for V^* is a mapping $T^* : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$

$$(T^* V^*)(s) = \max_{a \in \mathcal{A}} \sum_{s',r} \mathcal{P}(s', r|s, a) \left[r + \gamma V^*(s')\right] \tag{19}$$

- Using the Bellman optimality operator, the Bellman optimality equation can be **compactly** written as:

$$T^* V^* = V^*; \tag{20}$$

- **Linear equation** in V^* and T^*;
- If $0 < \gamma < 1$, then T^* is a **contraction** w.r.t. the maximum norm;
- V^* is a **fixed point** of the Bellman operator T^*.

Bellman optimality operator - 2

Bellman optimality operator for Q^*

The **Bellman optimality operator** for Q^* is a mapping $T^* : \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$

$$(T^*Q^*)(s,a) = \sum_{s',r} \mathcal{P}(s',r|s,a)\left[r + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a')\right] \tag{21}$$

- Using the Bellman optimality operator, the Bellman optimality equation can be **compactly** written as:

$$T^*Q^* = Q^*; \tag{22}$$

- **Linear equation** in Q^* and T^*;
- If $0 < \gamma < 1$, then T^* is a **contraction** w.r.t. the maximum norm;
- Q^* is a **fixed point** of the Bellman operator T^*.

Properties of Bellman operators

- **Monotonicity**: if $f_1 \leq f_2$ component-wise

$$T^\pi f_1 \leq T^\pi f_2, \qquad T^* f_1 \leq T^* f_2; \tag{23}$$

- **Max-norm contraction**: for two vectors f_1 and f_2

$$\|T^\pi f_1 - T^\pi f_2\|_\infty \leq \gamma \|f_1 - f_2\|_\infty ; \tag{24}$$

$$\|T^* f_1 - T^* f_2\|_\infty \leq \gamma \|f_1 - f_2\|_\infty . \tag{25}$$

- V^π and Q^π are the **unique fixed points** of T^π;
- V^* and Q^* are the **unique fixed points** of T^*;
- For any vector $f \in \mathbb{R}^{|\mathcal{S}|}$ and any policy π, we have

$$\lim_{k\to\infty} (T^\pi)^k f = V^\pi, \qquad \lim_{k\to\infty} (T^*)^k f = V^*. \tag{26}$$

Outline

What is model-free control?

Model-free control: **optimize** the value function of an **unknown** MDP;

- **Input**: MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \iota, \gamma \rangle$;
- **No** access or knowledge to $\mathcal{P}$ and $\mathcal{R}$;
- Need to explore to gather knowledge of the MDP;
- **Output**: optimal value function V^* and optimal policy π^*.

On- and off-policy learning

- **On-policy** learning:
  - Learn about policy π from experience sampled from π;
  - Evaluate and improve the **same policy** that the agent is already using for action selection.
- **Off-policy** learning:
  - Learn about policy π from experience sampled from another policy b;
  - Evaluate and improve policy π that is **different** from the policy b that is used for action selection when exploring.

Exploration vs exploitation – A brief note

- **Online** decision-making involves a fundamental choice:
  - **Exploitation**: make the **best** decision given current information;
  - **Exploration**: gather **more** information.
- The best long–term strategy may involve **short–term sacrifices**;
- Gather **enough** information to make the **best** overall decisions.

Common exploration approaches in RL

- ε-greedy:

$$a_t = \begin{cases} a_t^* \text{ with probability } 1-\varepsilon \\ \text{random action with probability } \varepsilon \end{cases} \tag{27}$$

- Softmax:
  - Bias exploration towards **promising** actions;
  - Softmax action selection methods **grade** action probabilities by estimated values;
  - The most common softmax uses a **Gibbs (or Boltzmann)** distribution:

$$\pi(a|s) = \frac{\exp \frac{Q(s,a)}{\tau}}{\exp \sum_{a' \in \mathcal{A}} \frac{Q(s,a')}{\tau}}, \tag{28}$$

    where τ is a **temperature**:
    - $\tau \to \infty$: random action $P = \frac{1}{|\mathcal{A}|}$;
    - $\tau \to 0$: greedy action $a^* = \text{argmax}_a Q(s,a)$.

Sarsa algorithm for on-policy control

Initialize $Q(s, a)$ arbitrarily, except that $Q(\text{terminal}, \cdot) = 0$;
for each episode **do**
  Initialize s;
  Choose a from s using policy derived from Q (e.g., ε-greedy);
  for each step of episode **do**
    Take action a, observe r, s';
    Choose a' from s' using policy derived from Q (e.g., ε-greedy);
    $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$;
    $s \leftarrow s'$; $a \leftarrow a'$.
  end for
end for

Convergence of Sarsa

Theorem

Sarsa **converges** to the optimal action-value function, $Q(s,a) \rightarrow Q^*(s,a)$, under the following conditions:

- **GLIE** sequence of policies $\pi_t(s,a)$;
- **Robbins-Monro** sequence of step-sizes α_t:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad ; \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty. \tag{29}$$

Off-policy learning

- Learn about **target policy** $\pi(a|s)$;
- …while following **behavior policy** $b(a|s)$;
- Why is this important?
  - Learn from **observing** humans or other agents;
  - **Re-use** experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_t$;
  - Learn about **optimal** policy while following an **exploratory** policy;
  - Learn about **multiple** policies while following **one** policy.

Q-Learning

- We now consider **off-policy** learning of **action-values** $Q(s, a)$;
- Action to be executed at state s is chosen using **behavior** policy $a_t \sim b(\cdot|s)$;
- For the target, we consider a **successor** action from the **greedy** policy $a' = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$;
- Update $Q(s_t, a_t)$ as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)). \quad (30)$$

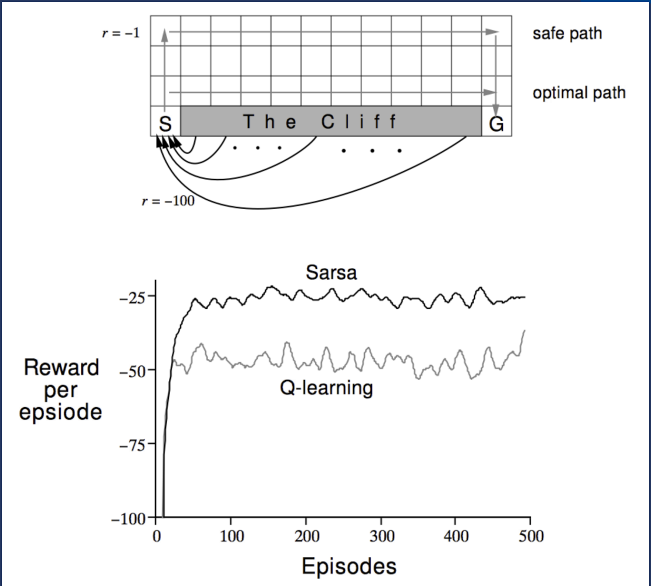
Theorem

Q-learning control **converges** to the optimal action-value function, $Q(s, a) = Q^*(s, a)$.

Q-Learning algorithm for off-policy control

Initialize $Q(s, a)$ arbitrarily, except that $Q(\text{terminal}, \cdot) = 0$
for each episode **do**
    Initialize s
    for each step of episode **do**
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \max_{a'} \gamma Q(s', a') - Q(s, a))$
        $s \leftarrow s'$
    end for
end for

Cliff-walking example



Q-Learning vs Sarsa

- **Sarsa:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)); \tag{31}$$

- **Q-learning:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \max_{a'} \gamma Q(s',a') - Q(s,a)); \tag{32}$$

- In the cliff−walking task:
  - Sarsa: learns a **safe non−optimal** policy away from edge;
  - Q−learning: learns **optimal** policy along edge.
- ε−greedy algorithm:
  - For $\varepsilon \neq 0$, Sarsa performs **better** online;
  - For $\varepsilon \to 0$ gradually, both **converge** to optimal.

Wrap-up

- What is Reinforcement Learning and Markov Decision Processes;
- Value function and policies;
- Bellman equations and operators;
- Model-free control algorithms.