Data Science

Reinforcement Learning

Lionel Fillatre
Polytech Nice Sophia
lionel.fillatre@univ-cotedazur.fr

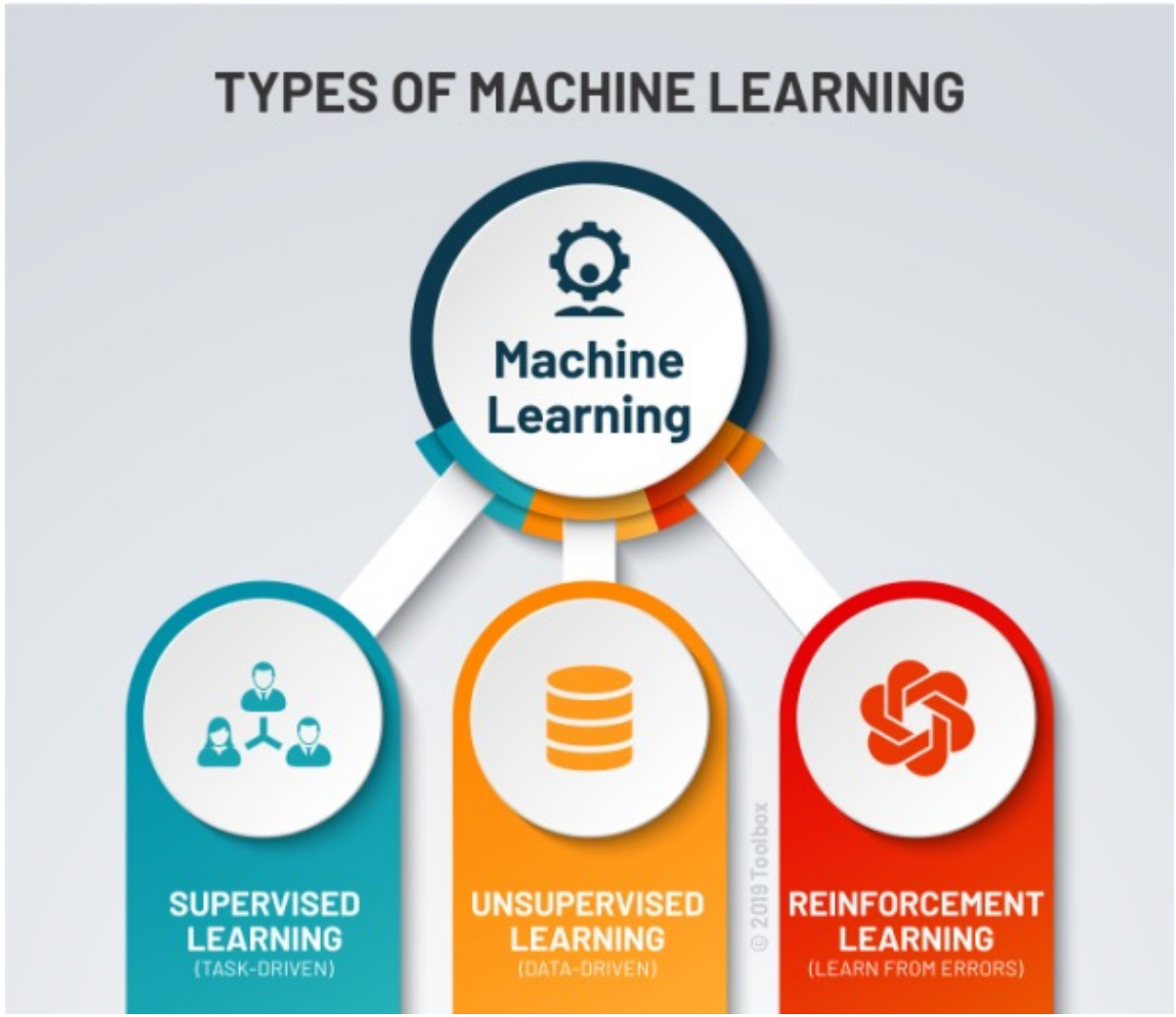
Outline of this Lecture

- Introduction
- State-Value Function
- State-Action Value Function
- Dynamic Programming
- Conclusion

- Appendix: value iteration algorithm convergence

Introduction

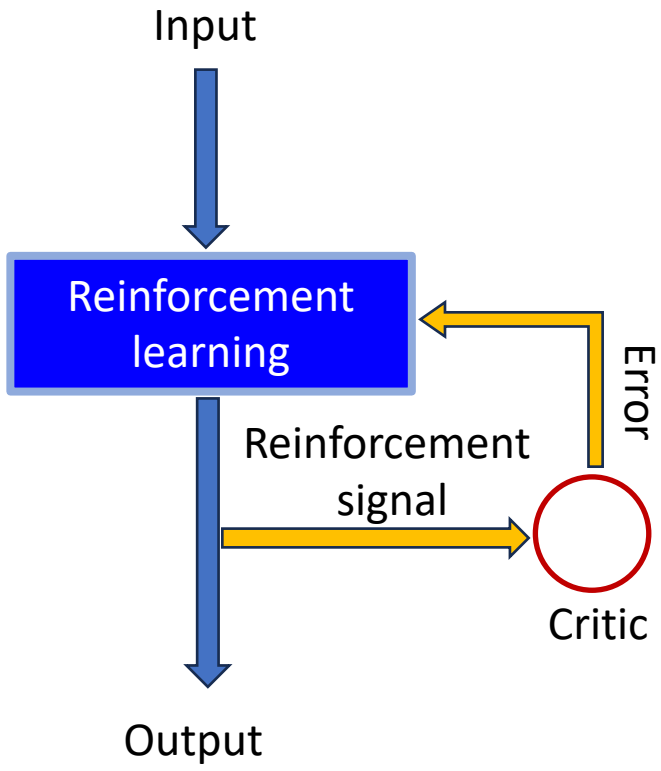
Branches of Machine Learning



https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/

Characteristics of Reinforcement Learning

- What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d data)
  - Agent's actions affect the subsequent data it receives



Some examples

- Playing games: Atari

https://www.youtube.com/watch?v=V1eYniJ0Rnk

- Playing games: Super Mario

https://www.youtube.com/watch?v=wfL4L_l4U9A

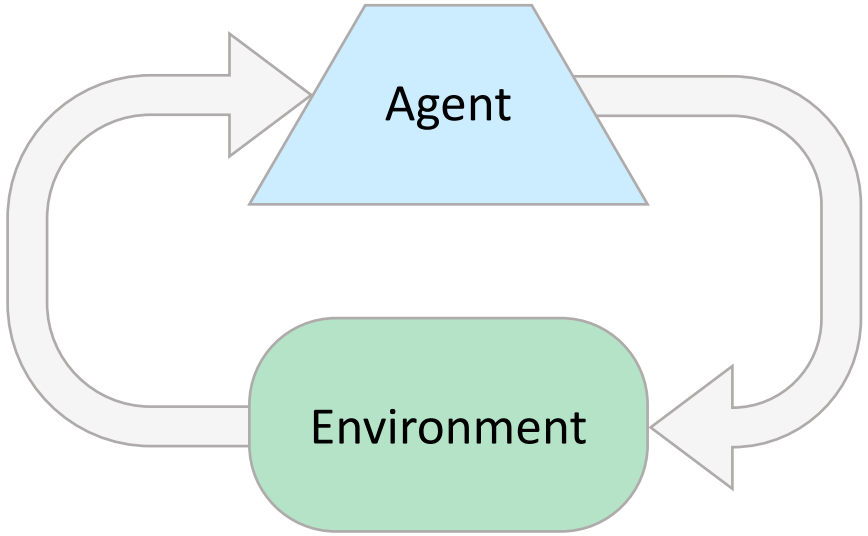
- Making pancakes with a robot

https://www.youtube.com/watch?v=W_gxLKSsSIE

Reinforcement Learning

State: S_t
Reward: R_t

$S_{t+1} \sim \mathcal{P}(\quad | S_t, A_t)$
$R_{t+1} \sim \mathcal{R}(\quad | S_t, A_t, S_{t+1})$



Actions: A_t
$A_t \sim \pi(\quad | S_t)$

Trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

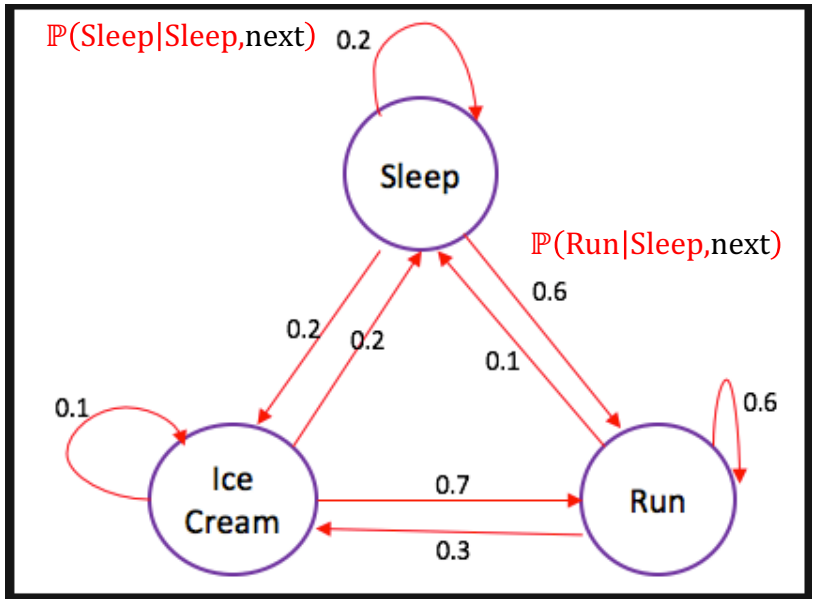
Agent-Environment Interface

- The agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$
- At each time step t, the agent receives some representation of the environment's state, S_t, and on that basis selects an action, A_t
- One time step later, in part as a consequence of its action, the agent receives a numerical reward, R_{t+1}, and finds itself in a new state, S_{t+1}
- The agent and environment together thereby give rise to a sequence or trajectory that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

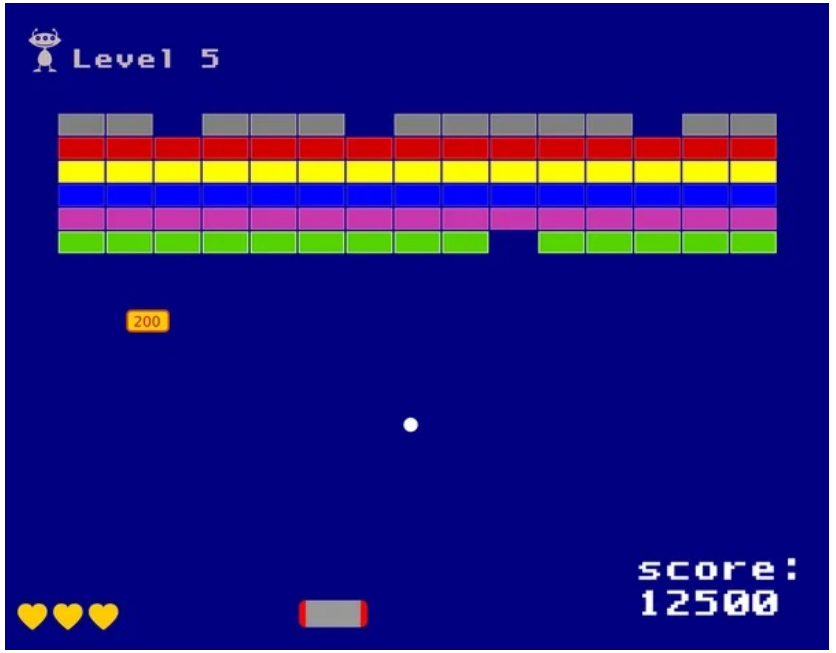
Markov Decision Process (MDP): States and Actions

- Let us describe the dynamics of the environment.
- The set of all possible actions is $\mathcal{A}$
- The set of all possible states is $\mathcal{S}$
- For a given state s, $\mathcal{A}(s)$ denotes the actions possible from this state s (some actions in $\mathcal{A}$ may be impossible)
- The state is a description of the environment in sufficient detail to determine its evolution.
  - Think of Newtonian physics.
  - Markov assumption: the state at time $t+1$ depends directly on the state and action at time t, but not on past states and actions.
- To describe the dynamics, we need to specify the state-transition probabilities $\mathbb{P}(S_{t+1}|S_t, A_t)$.
- In this lecture, we assume the state S_t is fully observable at any time, a highly nontrivial assumption.



$\mathcal{S} = \{\text{Sleep, Run, Ice Cream}\}$
$\mathcal{A} = \{\text{next}\}$
(only one action)

MDP: States and Actions



- Suppose you're playing Breakout. What should be the set of states and actions?
  - states = locations of the blocks, actions = paddle command (Right or Left)?
  - states = set of pixels, actions = trajectory of the paddle?

- In general, the right granularity of states and actions depends on what you're trying to achieve.

Full Dynamics Model

- **Full transition probabilities:**
  - It is given by the full model

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

  - This conditional probability gives the probability to achieve a state s' with a reward r from the state s when taking the action a.
  - Technically speaking, the reward $R_{t+1} = r$ is a random value that is function of s', s and a, i.e.it is defined by the four-argument probability density function $\mathbb{P}(R_{t+1} = r|S_{t+1} = s', S_t = s, A_t = a)$
  - All the presented results can be extended to this very general setting.

- **State-transition probabilities:**
  - The state-transition probabilities $p(s'|s, a) = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a)$ are obtained from

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

where $\mathcal{R}$ is the set of all possible rewards

MDP: Expected Reward

- We can also compute the expected rewards for state–action pairs $r(s, a)$:

$$r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \; p(r|s, a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

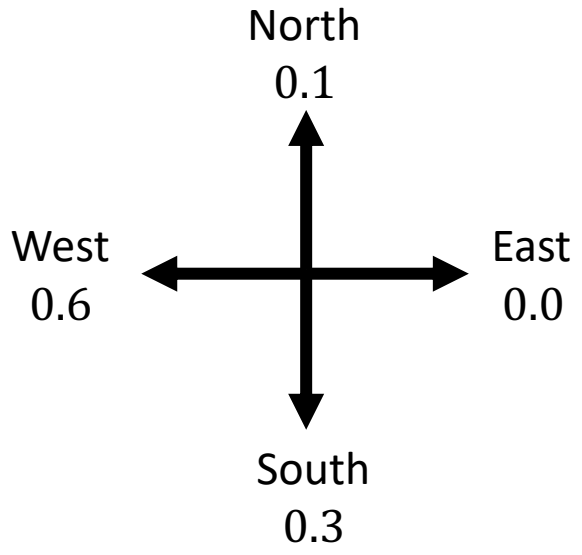
  - Two-argument function
  - Appropriate to model a reward just depending on state and action

- We can also compute the expected rewards for state–action–next-state triples $r(s, a, s')$

$$r(s, a, s') = \mathbb{E}[R_t | S_t = s, A_t = a, S_{t+1} = s'] = \sum_{r \in \mathcal{R}} r \; p(r|s, a, s') = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)}$$

  - Three-argument function
  - Appropriate to model a reward just depending on state, action and next state

MDP: Policies



- The way the agent chooses the action in each step is called a policy.
- **Two main types:**
  - Deterministic policy: $A_t \sim \pi(S_t)$ for some function $\pi: \mathcal{S} \to \mathcal{A}$
  - Stochastic policy: $A_t \sim \pi(\quad|S_t)$ for some function $\pi: \mathcal{S} \to \mathcal{P}(\mathcal{A})$. (Here, $\mathcal{P}(\mathcal{A})$ is the set of distributions over actions.)

- With stochastic policies, the distribution over rollouts, or trajectories, factorizes:

$$p(s_1, a_1, \ldots, s_T, a_T) = p(s_1)\pi(a_1|s_1)p(s_2|s_1, a_1)\pi(a_2|s_2) \cdots p(s_T|s_{T-1}, a_{T-1})\pi(a_T|s_T)$$

- Note:
  - The fact that policies need consider only the current state is a powerful consequence of the Markov assumption and full observability.
  - If the environment is partially observable, then the policy needs to depend on the history of observations.

MDP: Reward

- **Episode**: A sequence of interactions based on which the reward will be judged at the end

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots, A_{T-1}, R_T$$

- The **return G_t** is the total discounted reward from time-step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

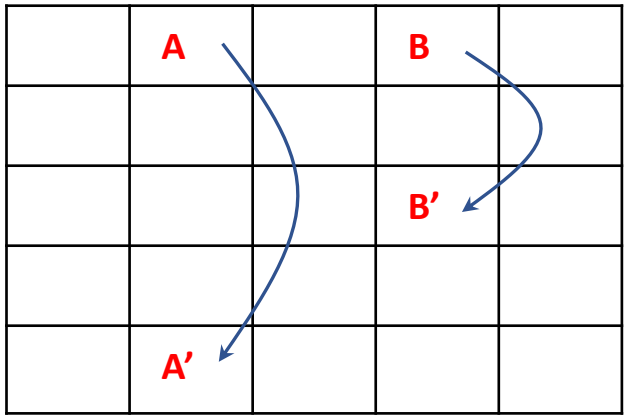
- The discount $\gamma \in [0,1]$ tells how much we care about reward now versus reward later.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation (the sum blows up for infinite length)
  - $\gamma = 1$ leads to undiscounted return

Definition of the reward matters

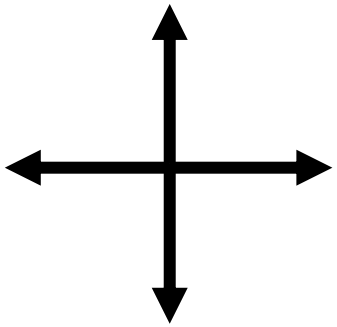
- How might you define a reward function for an agent learning to play a video game?
  - **Score**: Change in score (why not current score?)
  - **Novelty**: Some measure of novelty (this is sufficient for most Atari games!)
  - **Demonstration**: Learning from demonstrations allows us to avoid specifying a reward directly and instead just learn to imitate how a human would complete the task.
  - **Feedback**: We can also incorporate human feedback by evaluating the quality of episodes or even sharing control with the agent in an interactive manner.
  - **Transfer learning**: It may be possible to use transfer learning to train on many similar games, and infer a “common sense” reward function for this game.

https://openai.com/blog/faulty-reward-functions/

Example: Gridworld



Actions (move on the grid): north, south, east, west



- The cells of the grid correspond to the states of the environment.
- Reward:
  - Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1.
  - Other actions result in a reward of 0, except :
    - From state A, all four actions yield a reward of $+10$ and take the agent to A'
    - From state B, all actions yield a reward of $+5$ and take the agent to B'.
- What's the strategy to achieve max reward?

Planning and Policy

	Prediction	Control
Model-based Planning	• Dynamic programming: Policy Evaluation	• Dynamic programming: Policy Iteration • Dynamic programming: Value Iteration
Model-free reinforcement learning	• Monte Carlo prediction • Temporal Difference	• Monte Carlo control • Sarsa • Q learning

- **Planning**: a fully specified MDP is already known.
- **Reinforcement Learning**: agent interacts with an environment with unknown dynamics, i.e., the environment is a black box that takes in actions and outputs states and rewards.
- **Control in a model-free setting**: this is what most people mean when they use the term « Reinforcement learning ».

State-Value Function

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states and therefore to select between actions
- The **state-value function** $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \sum_{k=0}^{\infty} \gamma^k \mathbb{E}_\pi[R_{t+1+k} | S_t = s]$$

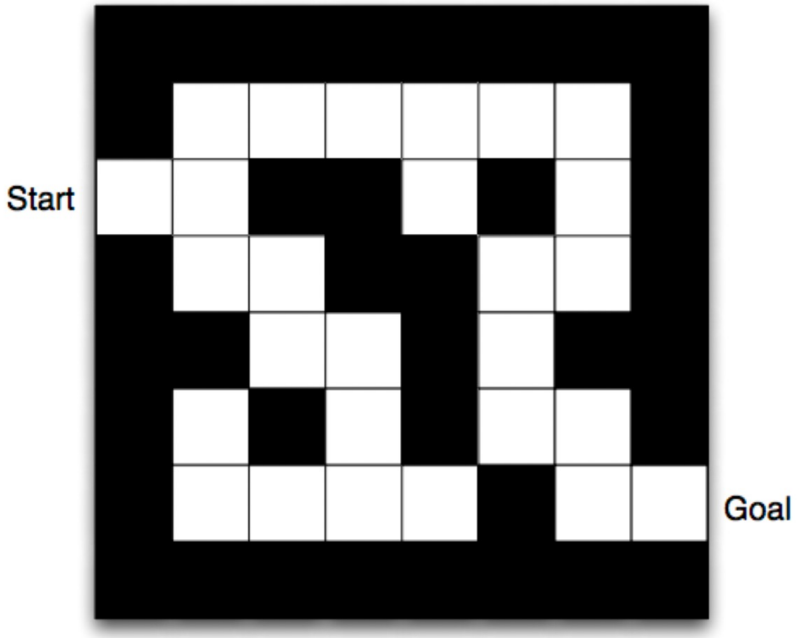
- $\mathbb{E}_\pi[X]$ denotes the expected value of a random variable X given that the agent follows policy π

- Very important question: **How to calculate the value function?**

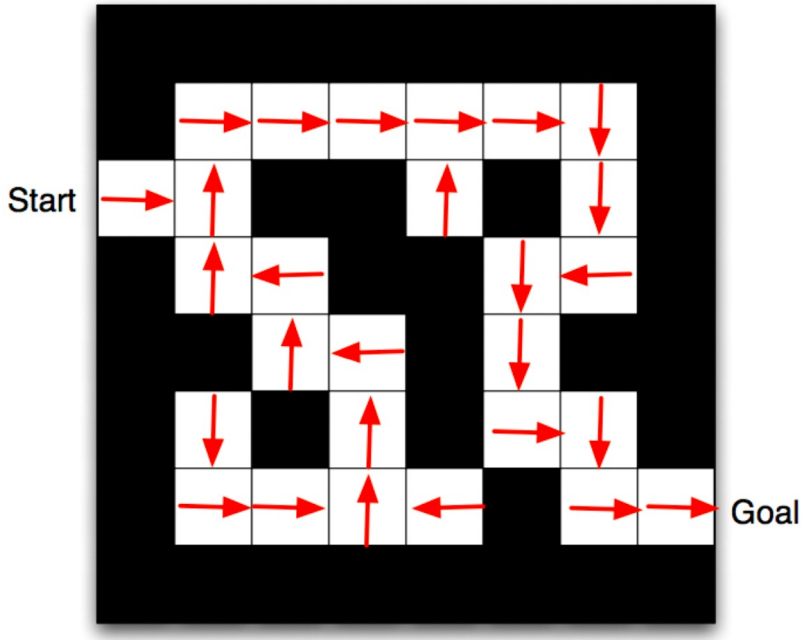
Example: Mazzle

- Rewards: −1 per time-step
- Actions: N, E, S, W
- States: Agent's location

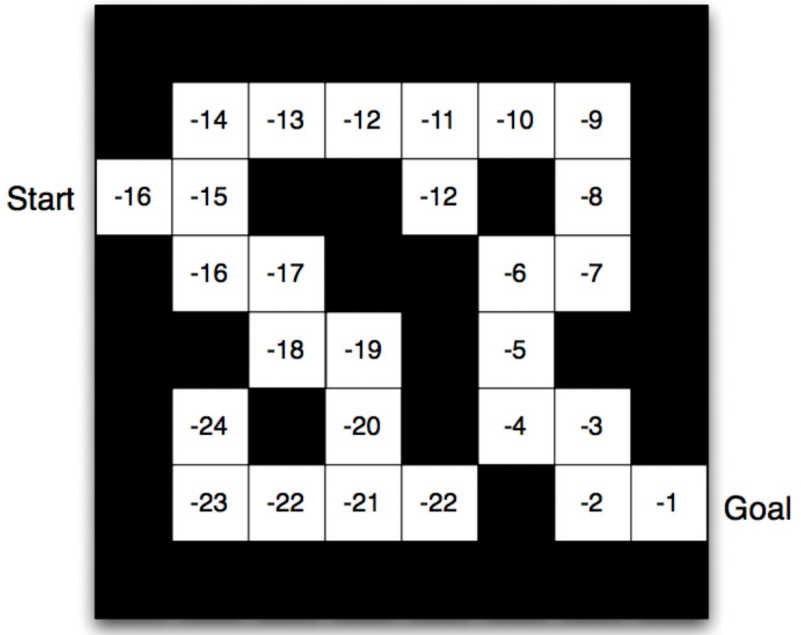
$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$



Mazzle



Proposed policy



Value function
(Easy interpretation here)

How to calculate the Value Function?

- Note that

$$v_\pi(S_{t+1} = s') = \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s'] = \mathbb{E}_\pi[G_t|S_t = s'] = v_\pi(s')$$

  - Hence, the value function does not depend on time t (because of the MDP model)

- Note also that

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) = R_{t+1} + \gamma G_{t+1}$$

  - Hence, we get

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

- Approach to calculate $v_\pi(s)$:
  - Compute $\mathbb{E}_\pi[R_{t+1}|S_t = s]$
  - Compute $\mathbb{E}_\pi[G_{t+1}|S_t = s]$
  - Combine both the results

How to calculate $\mathbb{E}_\pi[R_{t+1}|S_t = s]$?

- With the **full four-argument** model:

$$\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_{r\in\mathcal{R}} r \; p(r|s)$$

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

$$= \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}} p(s', r|s, a)\boldsymbol{p(a|s)}$$

$$= \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}} p(s', r|s, a)\boldsymbol{\pi(a|s)}$$

$$= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} r \, p(s', r|s, a)$$

because $p(s', r|s, a) = \frac{p(s', r, a|s)}{p(a|s)}$ and $\boldsymbol{p(a|s) = \pi(a|s)}$

How to calculate $\mathbb{E}_\pi[R_{t+1}|S_t = s]$?

- With the **two-argument reward function**

$$\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} r\, p(s', r|s, a)$$

$$= \sum_{a\in\mathcal{A}} \pi(a|s) r(s, a)$$

by definition of $r(s, a)$

- For simplicity, we use this form $\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_{a\in\mathcal{A}} \pi(a|s) r(s, a)$ in the rest of the lecture (so we do not exploit the full MDP dynamics).
- This expression is generally sufficient in practice since our policy is just a function $\pi(a|s)$ that does not depend on s'

Calculation of $\mathbb{E}_\pi[G_{t+1}|S_t = s]$ (slide 1/2)

- First let us prove that $\mathbb{E}_\pi[G_{t+1}|S_t = s] = \mathbb{E}_\pi[v_\pi(S_{t+1})|S_t = s]$
- The conditional expectation has the following property:
  - For random variables X, Y, Z, we have

$$\mathbb{E}[\mathbb{E}[X|Z,Y]|Y] = \mathbb{E}[X|Y]$$

- It follows that

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

since G_{t+1} does not depends on S_t when S_{t+1} is known.

https://en.wikipedia.org/wiki/Conditional_expectation

Calculation of $\mathbb{E}_\pi[G_{t+1}|S_t = s]$ (slide 2/2)

- Next, let us calculate that $\mathbb{E}_\pi[v_\pi(S_{t+1})|S_t = s]$

$$\mathbb{E}_\pi[v_\pi(S_{t+1})|S_t = s] = \sum_{s' \in \mathcal{S}} v_\pi(s')p(s'|s)$$

$$= \sum_{s' \in \mathcal{S}} v_\pi(s') \sum_{a \in \mathcal{A}} p(s', a|s)$$

$$= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} v_\pi(s')p(s'|s, a)p(a|s)$$

$$= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} v_\pi(s')p(s'|s, a)\pi(a|s)$$

$$\mathbb{E}_\pi[v_\pi(S_{t+1})|S_t = s] = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a)v_\pi(s')$$

Value Function: Bellman equation

- The foundation of many RL algorithms is the fact that value functions satisfy a recursive relationship, called the **Bellman equation**:

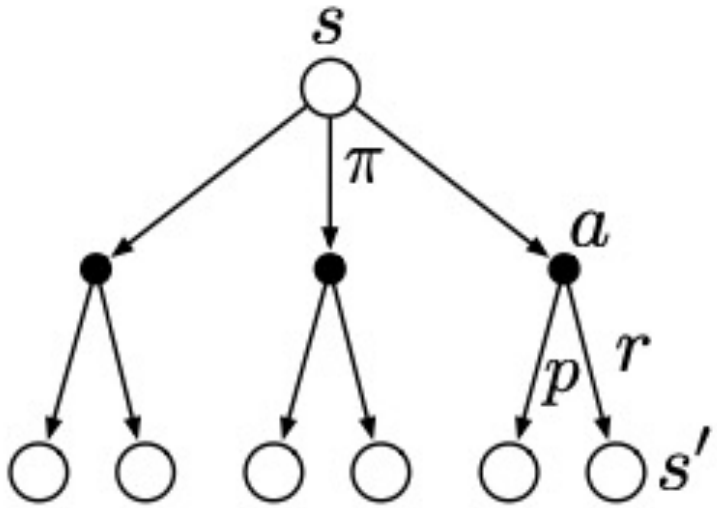
$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

$$= \sum_{a\in\mathcal{A}} \pi(a|s) r(s,a) + \gamma \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s\prime\in\mathcal{S}} p(s'|s,a) v_\pi(s')$$

$$= \sum_{a\in\mathcal{A}} \pi(a|s) \left[r(s,a) + \gamma \sum_{s\prime\in\mathcal{S}} p(s'|s,a) v_\pi(s') \right]$$

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left[r(s,a) + \gamma \sum_{s\prime\in\mathcal{S}} p(s'|s,a) v_\pi(s') \right]$$



Value Function: Bellman operator

- Viewing v_π as a vector V (where entries correspond to states), define the Bellman backup operator T_π:

$$T_\pi V(s) = \sum_{a \in A} \pi(a|s) \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \right]$$

- The Bellman equation can be seen as a fixed point of the Bellman operator

$$T_\pi v_\pi = v_\pi$$

Exact solution of the Bellman equations

- Let $\mathcal{S} = \{s_1, s_2, \dots, s_q\}$

$$v_\pi = \begin{pmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_q) \end{pmatrix}, \qquad \mathcal{R}_\pi = \begin{pmatrix} \sum_{a\in\mathcal{A}} \pi(a|s_1) r(s_1, a) \\ \\ \sum_{a\in\mathcal{A}} \pi(a|s_q) r(s_q, a) \end{pmatrix}, \qquad \mathcal{P}_\pi = \begin{pmatrix} \sum_{a\in\mathcal{A}} \pi(a|s_1) p(s_1|s_1, a) & \dots & \sum_{a\in\mathcal{A}} \pi(a|s_1) p(s_q|s_1, a) \\ \vdots & \ddots & \vdots \\ \sum_{a\in\mathcal{A}} \pi(a|s_q) p(s_1|s_q, a) & \dots & \sum_{a\in\mathcal{A}} \pi(a|s_q) p(s_q|s_q, a) \end{pmatrix}$$

- The set of Bellman equations

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left[r(s, a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s, a) v_\pi(s') \right]$$

becomes

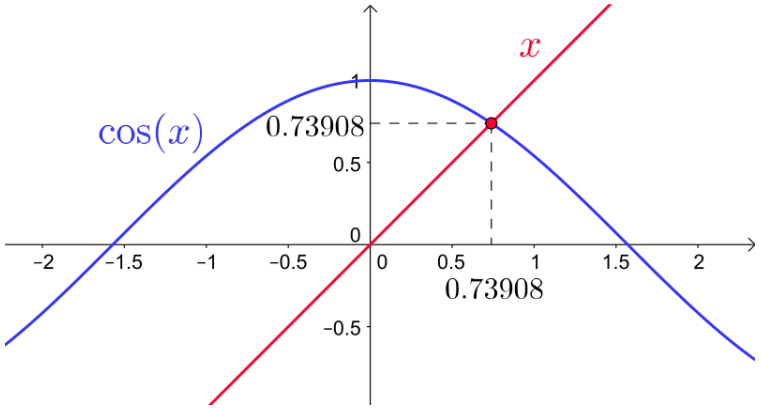
$$v_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi v_\pi$$

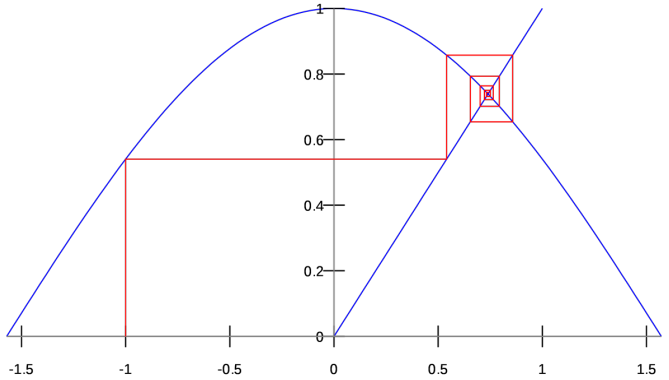
- If the **environnment model, the reward function and the policy are known** and if **the involved matrices are small enough**, we get

$$v_\pi = (I - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi \qquad \text{(if } I - \gamma \mathcal{P}_\pi \text{ is invertible)}$$

Fixed point algorithm: Principle

- The Dottie number d is the only real solution in $[-1,1]$ to the equation $\cos x = x$





- We get $\cos^n x = x$ for any integer n
- We can show that $\cos x$ is a contraction mapping: there exists $0 < q < 1$ such that

$$|\cos x - \cos y| < q|x - y| \text{ for all } (x, y) \in [-1,1]^2$$

- Hence, the Dottie number d is given by $\lim_{n\to\infty} x_n = d$ where

$$x_{n+1} = \cos x_n$$

$$x_0 \in \mathbb{R}$$

An Approximation: Iterative Policy Evaluation

- Let's use dynamic programming to find v_π.
  - This is an **approximation** since this is an iterative algorithm (v_k replaced v_π in the Bellman equations)
  - But it is **easier** and **faster**

- **Iterative Policy Evaluation:**
  - Start from an initial vector v_1 for v_π
  - For each $k = 1, 2, \ldots,$

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, v_k(s') \right]$$

- Observe:
  - A fixed point of this update is exactly a solution of the optimal Bellman equation (proof later).
  - We're treating v_π as a vector with $|S|$ entries.
  - This type of algorithm is an instance of dynamic programming.

Full Algorithm: Iterative Policy Evaluation

- This algorithm returns an estimate V of v_π

- Initialize:
  - A small threshold $\theta > 0$
  - $V(s)$ arbitrarily for all $s \in \mathcal{S}$ and $V(terminal)$ to 0

- Loop:
  - $\Delta \leftarrow 0$
  - Loop for each $s \in \mathcal{S}$:
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \sum_{a \in A} \pi(a|s)[r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s')]$
    - $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$

  until $\Delta < \theta$

State-Action Value Function

State-Action Value Function

- A closely related but usefully different function is the state-action value function
- The **action-value function** $q_\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \,|S_t = s, A_t = a\right]$$

- If you knew q_π, how would you obtain v_π?

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a)$$

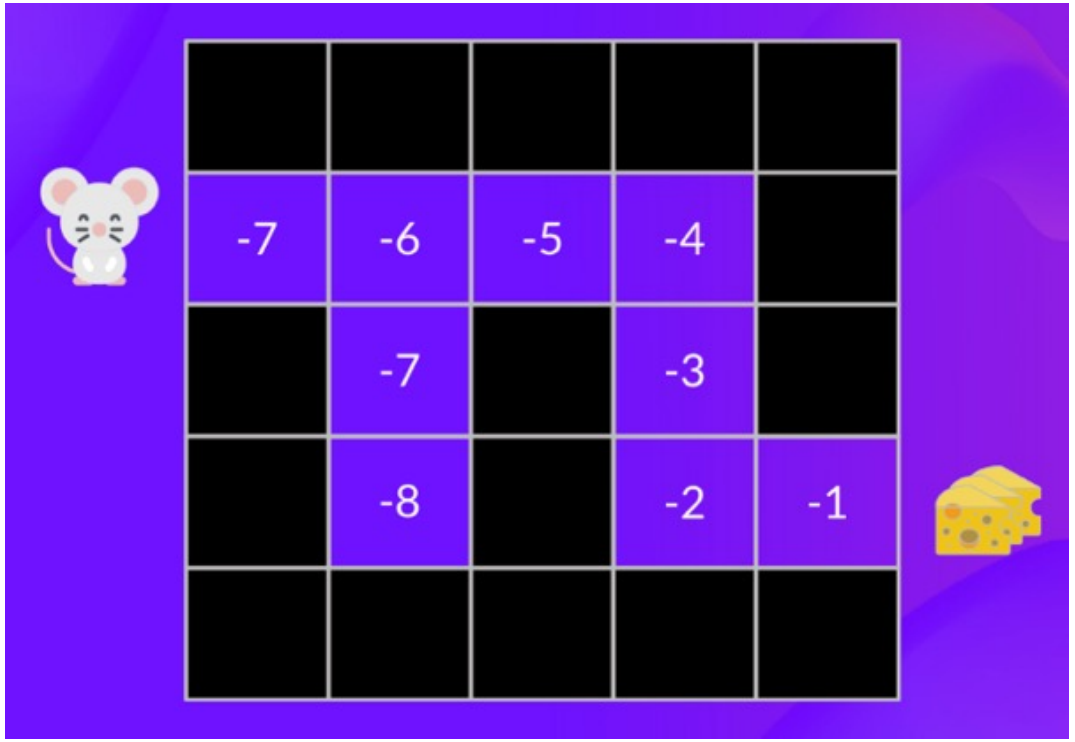
  - This requires knowing the policy, so in general it's rather easy to recover v_π from q_π.

- If you knew v_π, how would you obtain q_π?

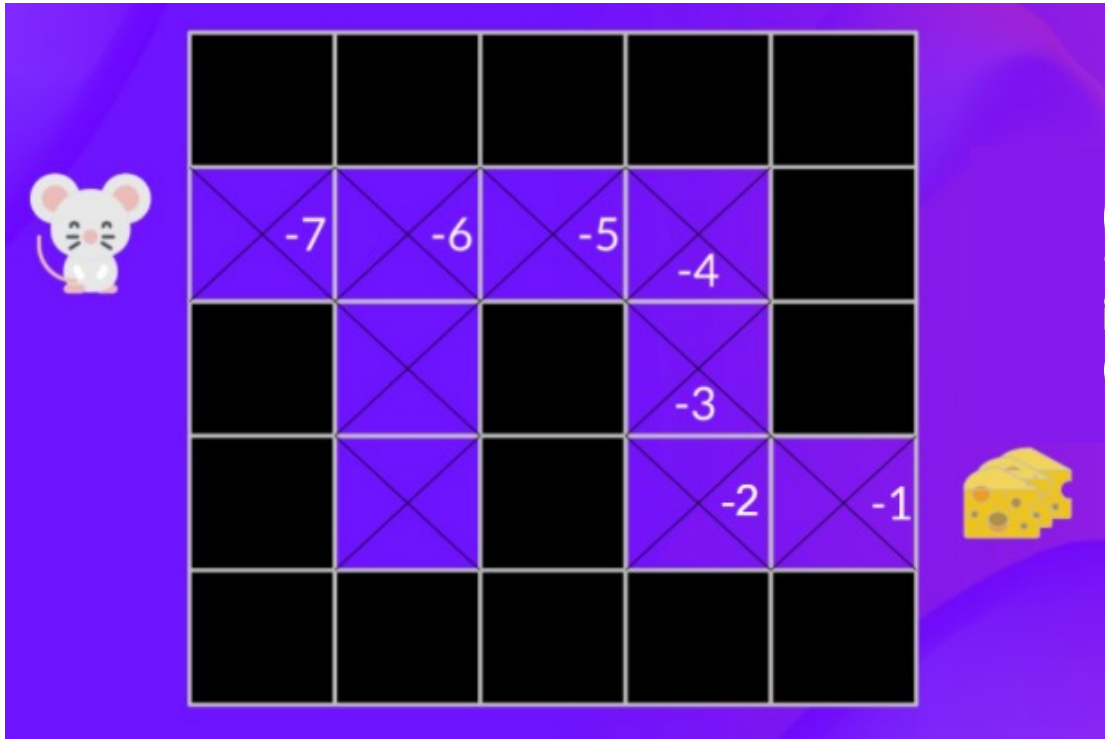
$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v_\pi(s')$$

  - This requires knowing the dynamics, so in general it's not easy to recover q_π from v_π.

Illustration



Value Function



State-Value Function
(just some state-action pairs are shown)

https://huggingface.co/learn/deep-rl-course/unit2/two-types-value-based-methods

Proof of $v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$

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

$$= \sum_{a \in \mathcal{A}} \pi(a|S_t = s) \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

$$= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

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

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} | S_t = s, A_t = a] + \gamma \mathbb{E}_\pi[q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} q_\pi(s', a') p(s', a' | s, a) \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} q_\pi(s', a') \pi(a'|s') p(s'|s,a) \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a') \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_\pi(s')
\end{aligned}
$$

since $v_\pi(s') = \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a')$

State-Action Value Function: Bellman equation

- q_π satisfies a Bellman equation very similar to v_π (proof is similar):

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[\sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a') \right]$$

- Viewing q_π as a vector (where entries correspond to state-action pairs), define the Bellman operator B_π:

$$B_\pi q_\pi(s,a) = q_\pi(s,a)$$

- The Bellman equation can be seen as a fixed point of the Bellman operator

$$B_\pi q_\pi = q_\pi$$

Value Functions

- Value functions measure the goodness of a particular state or state/ action pair:
  - How good is for the agent to be in a particular state or execute a particular action at a particular state, for a given policy.
- Optimal value functions measure the best possible goodness of states or state/action pairs under all possible policies.

	State values	Action values
Prediction	v_π	q_π
Control	v^*	q^*

Dynamic Programming

Greedy Policy

- Suppose you're in state s. You get to pick one action a, and then follow (fixed) policy π from then on. What do you pick?

$$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a')$$

- You get the **deterministic greedy policy** $\pi(s) = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a')$ for state s, i.e., $\pi(a|s) = 1$ if $a = \pi(s)$, otherwise $\pi(a|s) = 0$

Greedy Policy

- Remember that $v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$. For a deterministic policy π, it follows that

$$v_\pi(s) = q_\pi\big(s, \pi(s)\big)$$

- If a deterministic policy π^* is optimal, then it must be the case that for any state s:

$$\pi^*(s) = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, q_{\pi^*}(s, a'),$$

otherwise there exists (s,u) such as $\pi^*(s) < q_{\pi^*}(s,u)$, hence you could improve the policy π^* by changing $\pi^*(s) = u$.

Optimal Policy

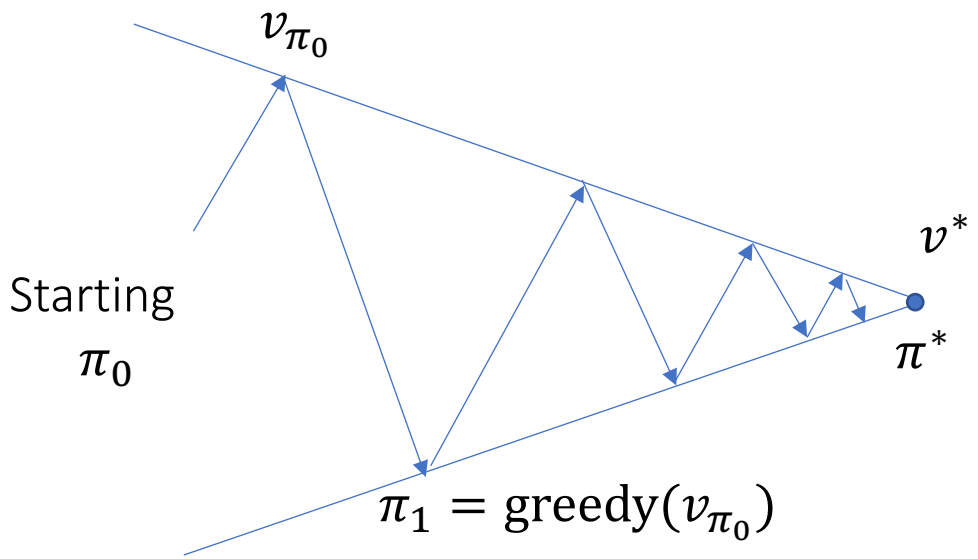
- The optimal greedy policy $\pi^*(s) = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, q_{\pi^*}(s, a')$ satisfies for any state s:

$$\pi^*(s) = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, q_{\pi^*}(s, a') = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \left(r(s, a') + \gamma \sum_{s' \in S} p(s'|s, a') v_{\pi^*}(s') \right)$$

- It is denoted $\pi^* = \text{greedy}(v_{\pi^*})$

- This leads to the **policy iteration algorithm**

Principle of Policy Iteration



- Policy evaluation: estimate v_{π_k}
- Policy improvement: generate π_{k+1} based on v_{π_k}

Policy Iteration for estimating π^*

- 1 - Initialize:
  - A small threshold $\theta > 0$
  - $V(s)$ arbitrarily for all $s \in \mathcal{S}$ and $V(terminal)$ to 0
  - $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

- 2- Policy Evaluation
  - Loop:
    - $\Delta \leftarrow 0$
    - Loop for each $s \in \mathcal{S}$:
      - $v \leftarrow V(s)$
      - $V(s) \leftarrow \sum_{a \in A} \pi(a|s)[r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s')]$
      - $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$

    until $\Delta < \theta$

- 3- Policy Improvement
  - $policy\_stable \leftarrow true$
  - For each $s \in \mathcal{S}$:
    - $old\_action \leftarrow \pi(s)$
    - $\pi(s) \leftarrow \underset{a' \in \mathcal{A}}{\text{argmax}}(r(s,a') + \gamma \sum_{s' \in S} p(s'|s,a') v_\pi(s'))$
    - If $old\_action \neq \pi(s)$, then $policy\_stable \leftarrow false$
  - If $policy\_stable$ is true,
    - then stop and return $V \approx v_{\pi^*}$ and $\pi \approx \pi^*$

    else
    - go to 2

A faster solution?

- One drawback to policy iteration is that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set.

- If policy evaluation is done iteratively, then convergence exactly to v_π occurs only in the limit. Must we wait for exact convergence, or can we stop short of that?

- One important special case is when policy evaluation is stopped after just one sweep (one update of each state).

Principle of Value Iteration

- From the optimal policy, it follows that

$$v_{\pi^*}(s) = q_{\pi^*}\left(s, \pi^*(s)\right) = \max_{a' \in \mathcal{A}} \left(r(s, a') + \gamma \sum_{s' \in S} p(s'|s, a') v_{\pi^*}(s') \right)$$

- The principle of the so called **value iteration algorithm** is to use this equation instead to the two-step computation (evaluation + improvement) in the policy iteration algorithm

Value Iteration for estimating π^*

- Initialize:
  - A small threshold $\theta > 0$
  - $V(s)$ arbitrarily for all $s \in \mathcal{S}$ and $V(terminal)$ to 0
  - $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

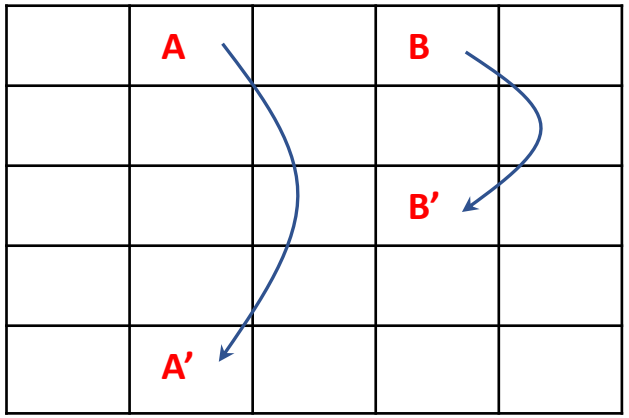
- Loop:
  - $\Delta \leftarrow 0$
  - Loop for each $s \in \mathcal{S}$:
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \max_{a' \in \mathcal{A}} (r(s, a') + \gamma \sum_{s' \in S} p(s'|s, a') V(s'))$
    - $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$

  until $\Delta < \theta$

- Output a deterministic policy $\pi \approx \pi^*$ such that

$$\pi(s) = \operatorname*{argmax}_{a' \in \mathcal{A}} \left(r(s, a') + \gamma \sum_{s' \in S} p(s'|s, a') V(s') \right)$$

Example: Gridworld

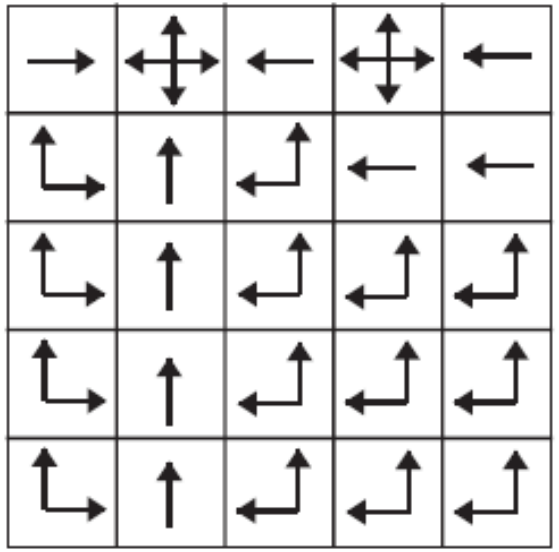


- Where there are multiple arrows in a cell, all of the corresponding actions are optimal.

$v_* =$

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

$\pi_* =$



Conclusion

Conclusion

- Reinforcement learning
  - One of three machine learning paradigms, alongside supervised learning and unsupervised learning.
  - Set of applications still being expanded

- Many reinforcement learning algorithms utilize dynamic programming techniques to derive exact solutions but
  - The state and action spaces to be small enough;
  - The dynamics of the model must be known.

- Learning the dynamics
  - Monte Carlo methods (actual or simulated interaction with an environment)

Appendix

Proof of convergence of the value iteration algorithm

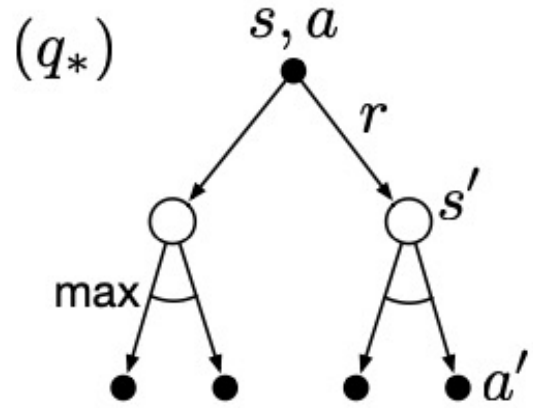
- The proof of convergence is based on the fixed point algorithm.
- The following slides detail this approach.

Optimal State-Action Value Function

- Bellman equation for optimal policy π^*:

$$q_{\pi^*}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) q_{\pi^*}\big(s', \pi^*(s')\big)$$

$$q_{\pi^*}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} q_{\pi^*}(s',a')$$



- Now $q_{\pi^*} = q^*$ is the optimal state-action value function, and we can rewrite the optimal Bellman equation without mentioning π^*:

$$q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} q^*(s',a') = B^* q^*(s,a)$$

- Turns out this is sufficient to characterize the optimal policy. So we simply need to solve the fixed point equation $B^* q^* = q^*$, and then we can choose

$$\pi^*(s) = \operatorname*{argmax}_{a' \in \mathcal{A}} q^*(s,a')$$

Bellman Fixed Points

- Some interesting problems could be reduced to finding fixed points of Bellman backup operators:
  - Evaluating a fixed policy π

$$B_\pi q_\pi = q_\pi$$

  - Finding the optimal policy

$$B^* q^* = q^*$$

- Idea: keep iterating the operator over and over again

$q \leftarrow B_\pi q$ (policy evaluation)

$q \leftarrow B^* q$ (finding the optimal policy)

- We're treating q_π or q^* as a vector with $|\mathcal{S}||\mathcal{A}|$ entries.
- This type of algorithm is an instance of dynamic programming.

Finding the Optimal State-Action Function

- Let's use dynamic programming to find q^*.

- Value Iteration:
  - Start from an initial function q_1 for q_π
  - For each $k = 1, 2, \ldots,$

$$q_k(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} q_k(s',a')$$

- Observe: a fixed point of this update is exactly a solution of the optimal Bellman equation

Proof of Fixed Point Convergence

- We get

$$|B^*q_1(s,a) - B^*q_2(s,a)| = \left\| \left[r(s,a) + \gamma \sum_{s\prime \in \mathcal{S}} p(s'|s,a) \max_{a\prime \in A} q_1(s',a') \right] - \left[r(s,a) + \gamma \sum_{s\prime \in \mathcal{S}} p(s'|s,a) \max_{a\prime \in \mathcal{A}} q_2(s',a') \right] \right\|$$

$$= \gamma \left\| \left[\sum_{s\prime \in \mathcal{S}} p(s'|s,a) \left[\max_{a\prime \in \mathcal{A}} q_1(s',a') - \max_{a\prime \in \mathcal{A}} q_2(s',a') \right] \right] \right\| \leq \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \left| \max_{a' \in \mathcal{A}} q_1(s',a') - \max_{a' \in \mathcal{A}} q_2(s',a') \right|$$

$$\leq \gamma \max_{s' \in \mathcal{S}, a' \in \mathcal{A}} |q_1(s',a') - q_2(s',a')| \sum_{s' \in \mathcal{S}} p(s'|s,a) = \gamma \max_{s' \in \mathcal{S}, a' \in \mathcal{A}} |q_1(s',a') - q_2(s',a')|$$

$$= \gamma \|q_1 - q_2\|_\infty$$

where $\|q_1 - q_2\|_\infty$ denotes the L^∞ norm.

- This is true for any (s,a), so

$$\|B^*q_1 - B^*q_2\|_\infty \leq \gamma \|q_1 - q_2\|_\infty$$

- B^* is a contraction map.

Proof of Fixed Point Convergence

- From

$$\|B^* q_1 - B^* q_2\|_\infty \leq \gamma \|q_1 - q_2\|_\infty$$

It follows that

$$\left\|(B^*)^{(k)} q_1 - (B^*)^{(k)} q_2\right\|_\infty \leq \gamma^k \|q_1 - q_2\|_\infty$$

where $(B^*)^{(k)}$ denotes the operator B^* iterated k times: $B^* \circ B^* \circ \cdots \circ B^*$

- Let q^* be a fixed point of B^*. Then for any q,

$$\left\|(B^*)^{(k)} q - q^*\right\|_\infty \leq \gamma^k \|q - q^*\|_\infty$$

- Hence, iterated application of B^*, starting from any q, converges exponentially to a unique fixed point.