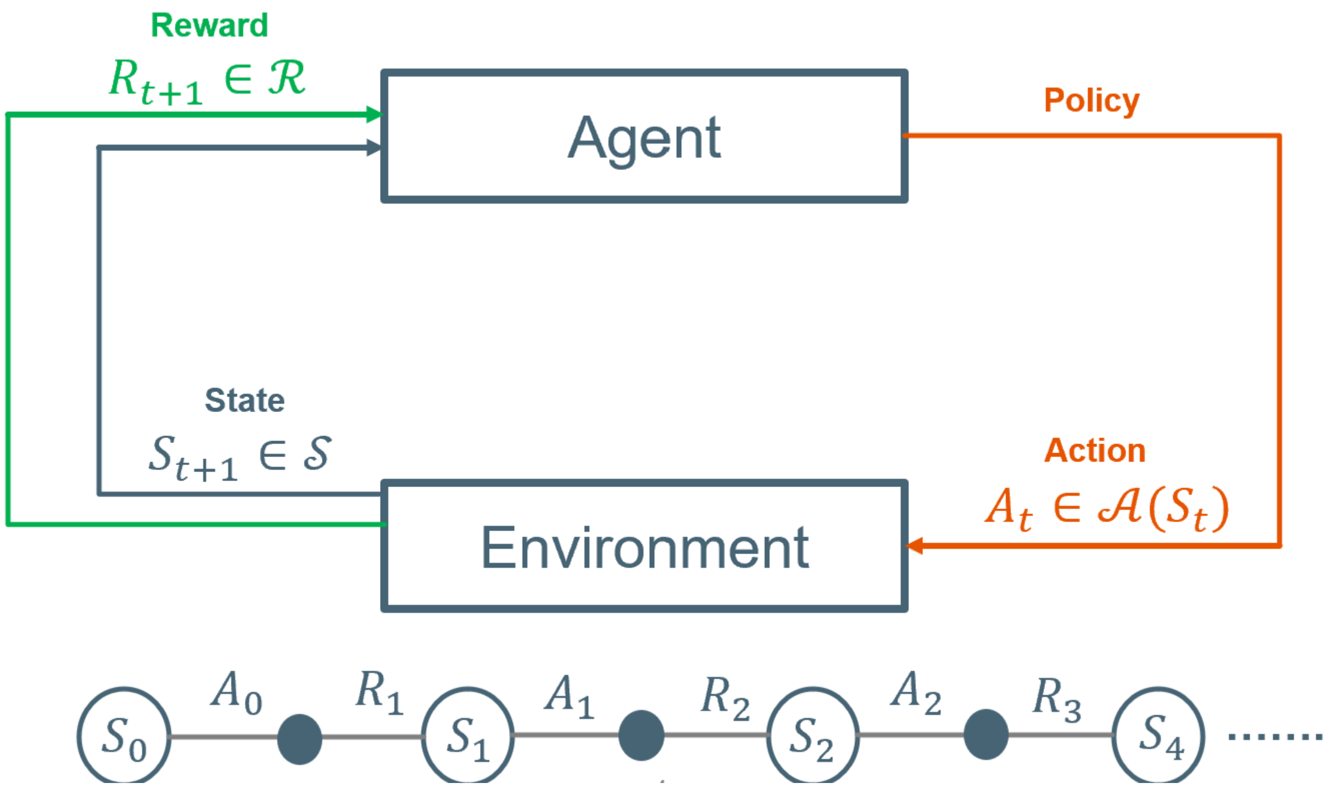
Reinforcement Learning
Fundamentals

APRAU 2025

Reinforcement Learning

- **Supervised Learning**
  - Dataset with labels. Specific outcome/answer. Predict test data to the correct outcome.
- Unsupervised Learning
  - Dataset with no labels. No specific outcome/answer. Find/analyze structure in the data
- **Reinforcement Learning**
  - No dataset nor labels. An environment with rules, rewards/penalties. Computer finds a sequence of decisions to maximize (long-term) reward, with repeated trials.

Reinforcement Learning



Reinforcement Learning

- **Goal**: Maximize the total reward by learning a sequence of actions

- **Steps**:

1. Formulate the problem into Markov Decision Process

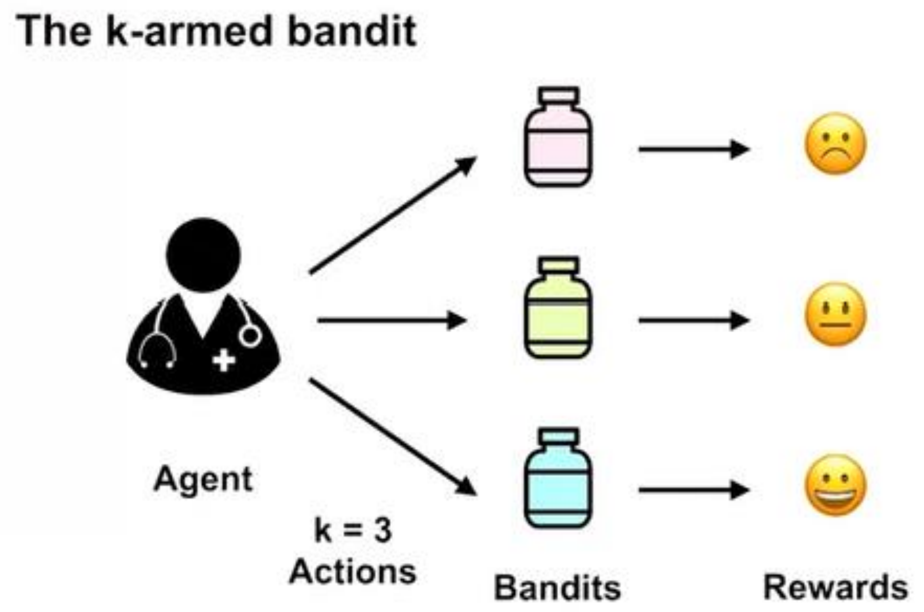
1. Set up a policy
   - Specify polices, value-functions, action values, deriving Bellman equations

2. Improving policy to become the optimal policy
   - Understand properties of optimal policies
   - Policy evaluation, policy control, policy iteration, policy improvement algorithms

K-Armed Bandit Problem

The k-armed bandit

In the k-armed bandit problem, we have an agent who chooses between “k” actions and receives a reward based on the action it chooses.



Action-values

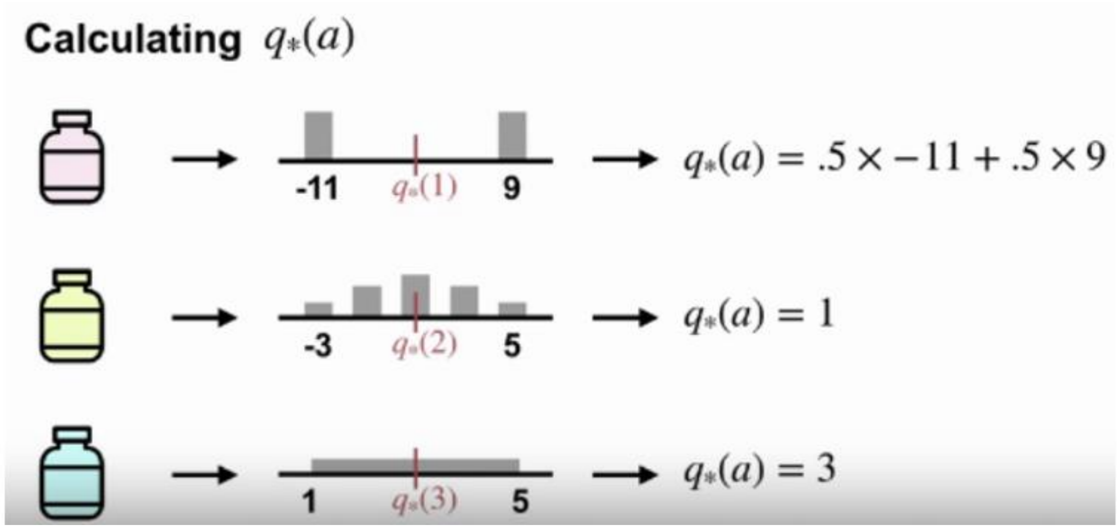
Action-Values

- **The value is the expected reward**

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a] \quad \forall a \in \{1, \ldots, k\}$$
$$= \sum_r p(r \mid a)\, r$$

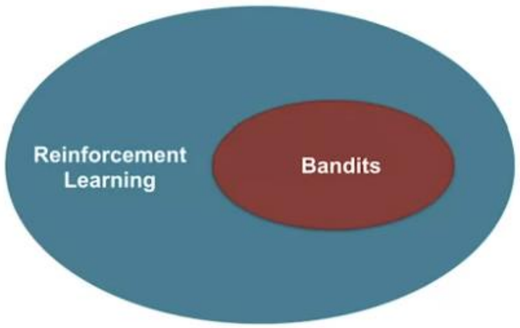
- **The goal is to maximize the expected reward**

$$\underset{a}{\operatorname{argmax}} \; q_*(a)$$

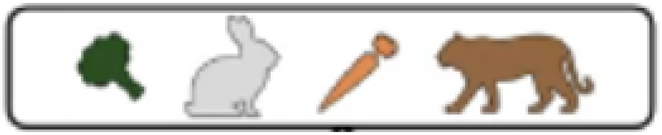


Markov Decision Process (MDP)

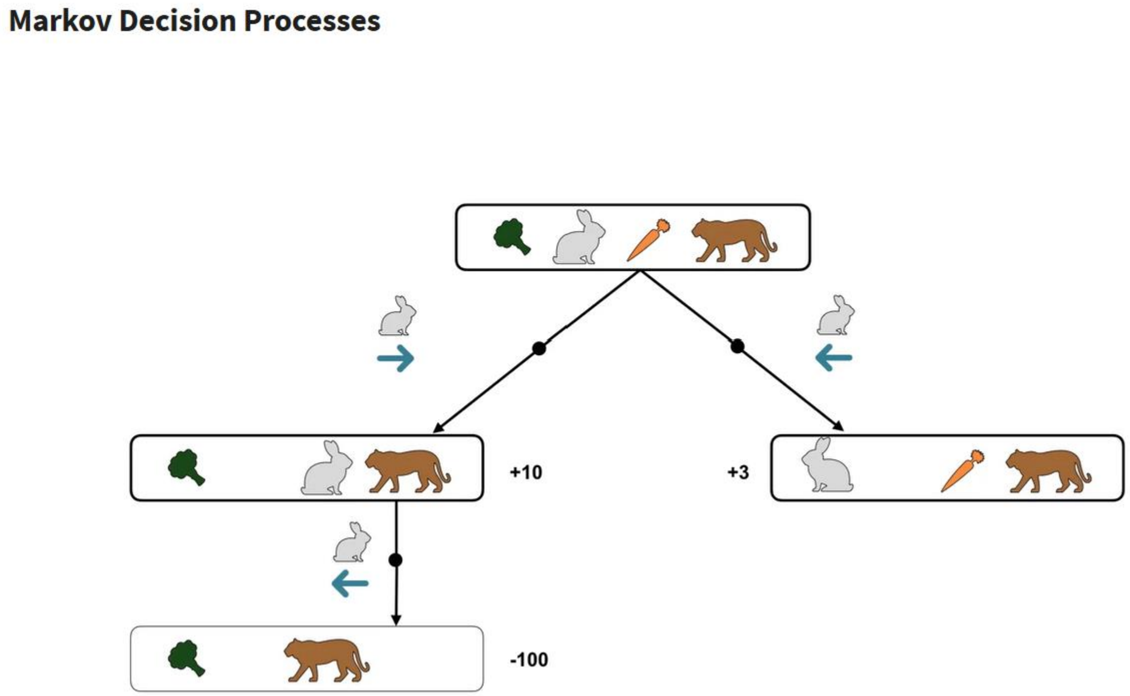
- The k-Armed Bandit problem
  - Does not account that different situations cal for different actions
  - The aggent is only concerned with immediate reward



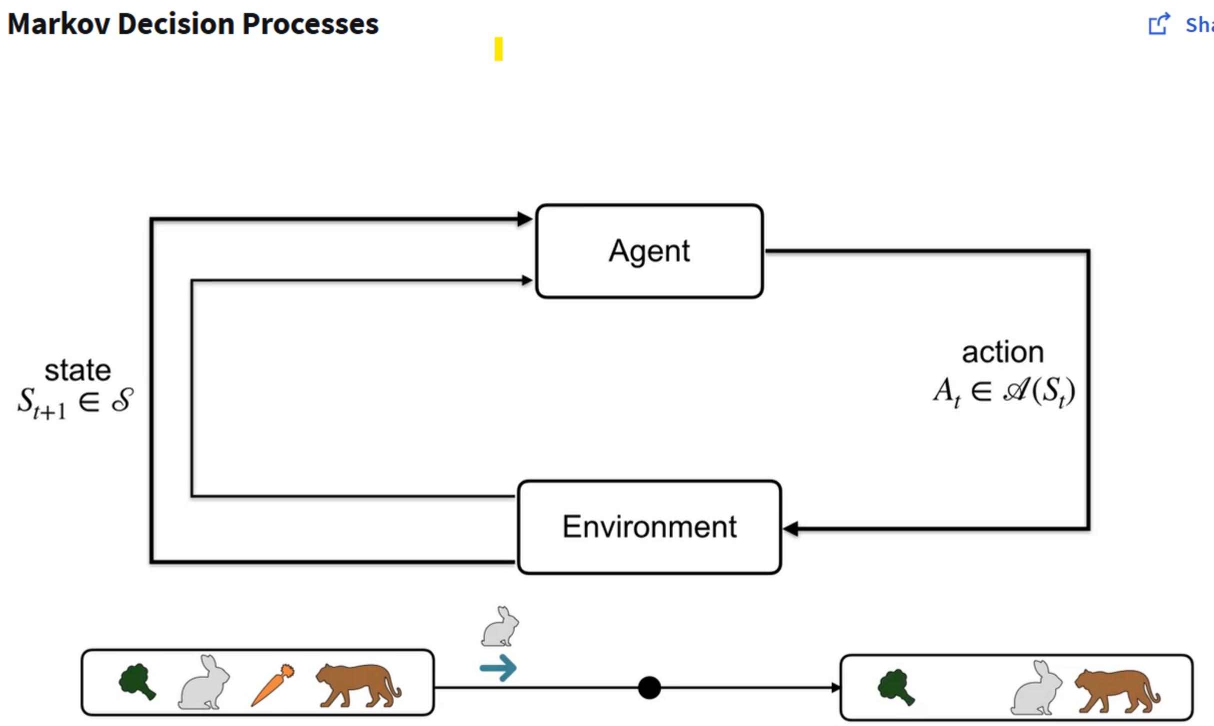
- A bandit rabbit would only be concerned about immediate reward
  - So it would go for the carrot…
  - But a better decision can be made considering the long-term impact of the decisions.

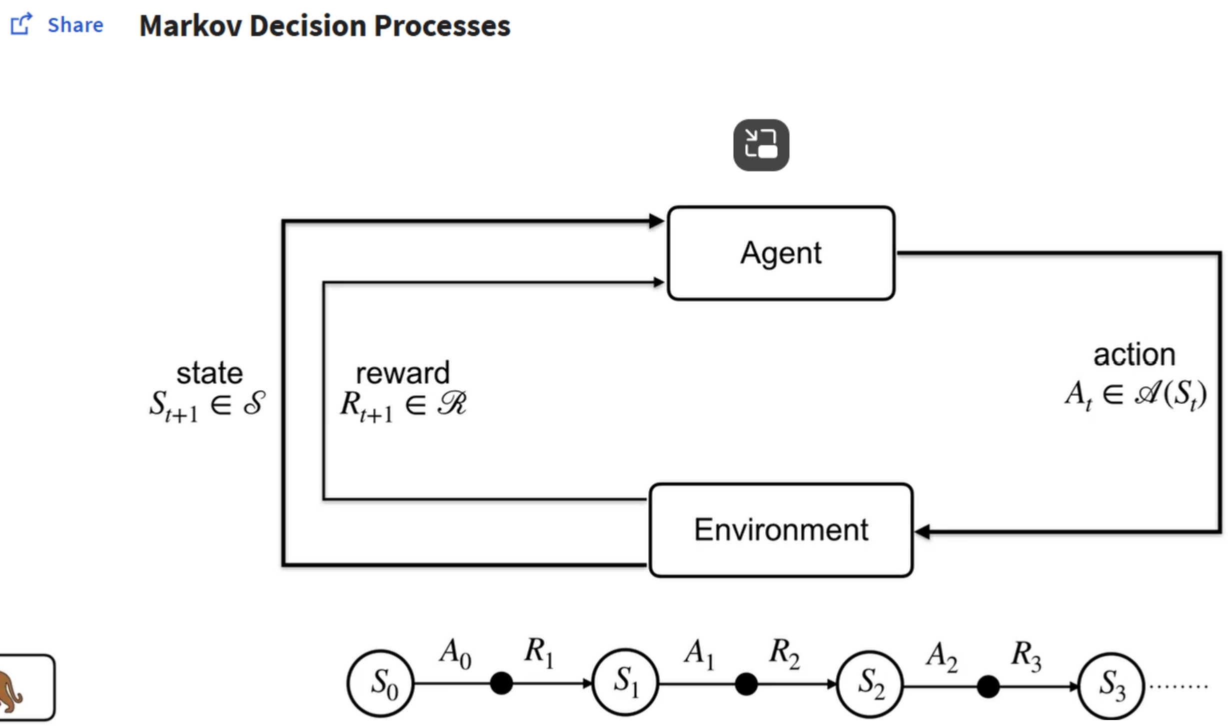


Markov Decision Process (MDP)

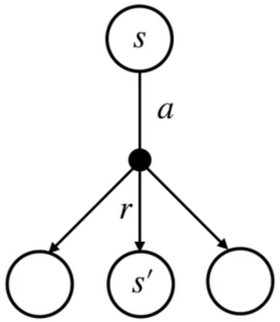


Markov Decision Process (MDP)





Markov Decision Process (MDP)



$$p(s', r \mid s, a)$$

$$p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$$

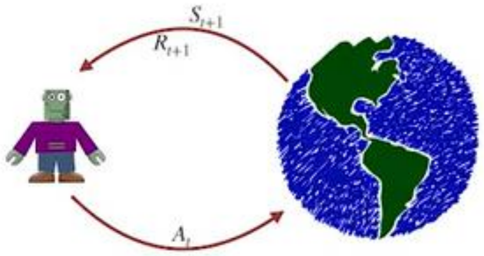
$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

The present state contains all the information necessary to predict the future

Summary

Summary

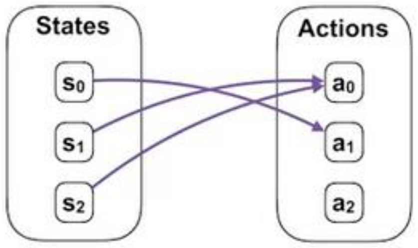
- MDPs provide a general framework for sequential decision-making
- The dynamics of an MDP are defined by a probability distribution



Value Functions and Bellman Equations

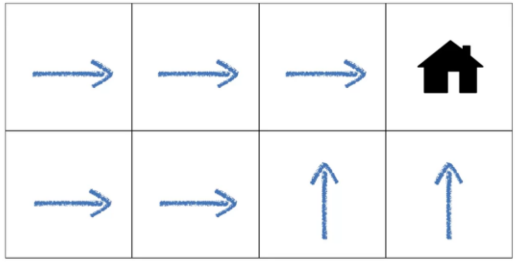
- Specifying Policies.
  - Policy is a distribution over actions for each state
    - Deterministic policy notation

$$\pi(s) = a$$



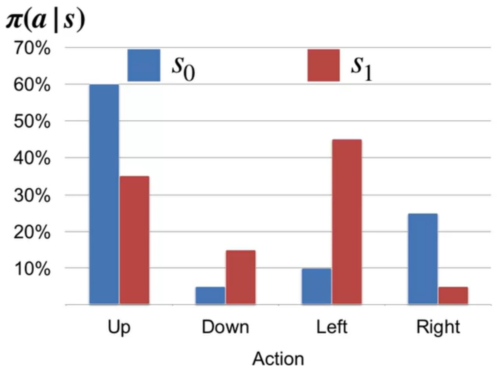
State	Action
s_0	a_1
s_1	a_0
s_2	a_0

    - Deterministic policy notation



Value Functions and Bellman Equations

- Value Functions
  - Stocastic policy notation

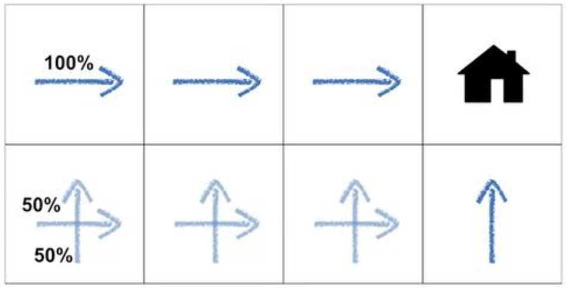


$$\pi(a|s)$$

$$\sum_{a \in \mathcal{A}(s)} \pi(a|s) = 1$$

$$\pi(a|s) \geq 0$$

  - Deterministic policy notation



Summary

Summary

- A policy maps the current state onto a set of probabilities for taking each action
- Policies can only depend on the current state

Value Functions and Bellman Equations

- Bellman equation

- Connect the value of a state and value of future states
- Using,
  - recursive relationship
  - the property that action choices only depend on the current state
  - next state action and reward only depends on the current state and action
  - Neither policy nor p depends on time

State-value functions

State-value functions

$$v\ (s) \doteq \mathbb{E}\ \left[G_t \mid S_t = s\right]$$

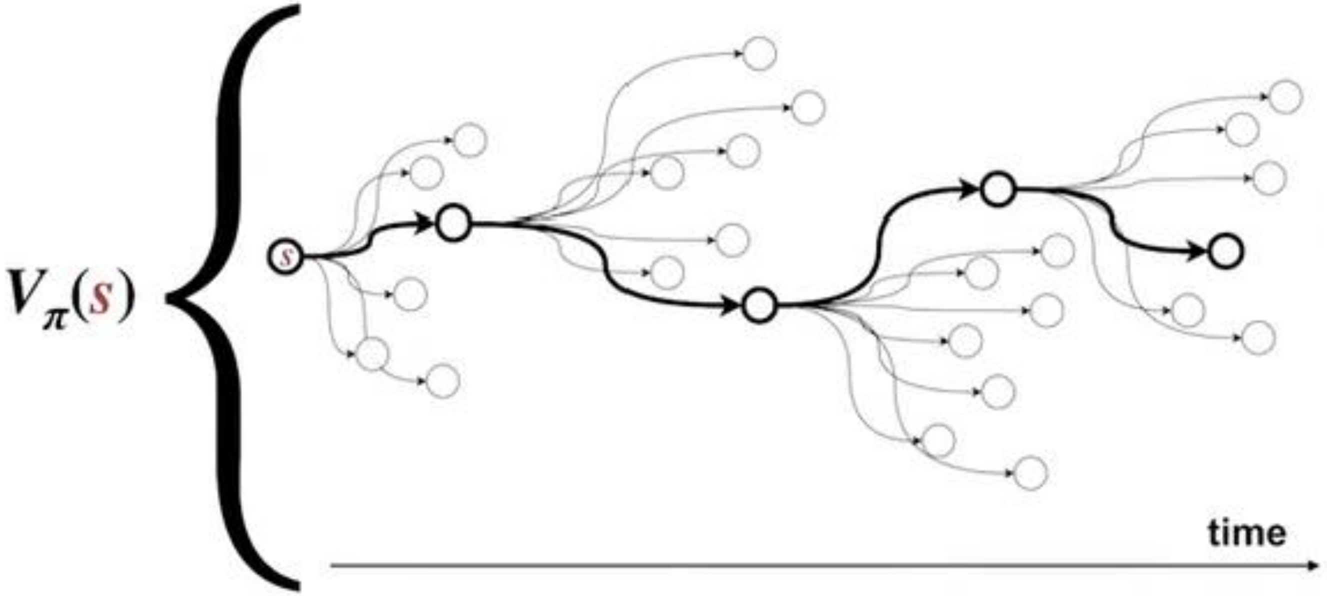
Recall that

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

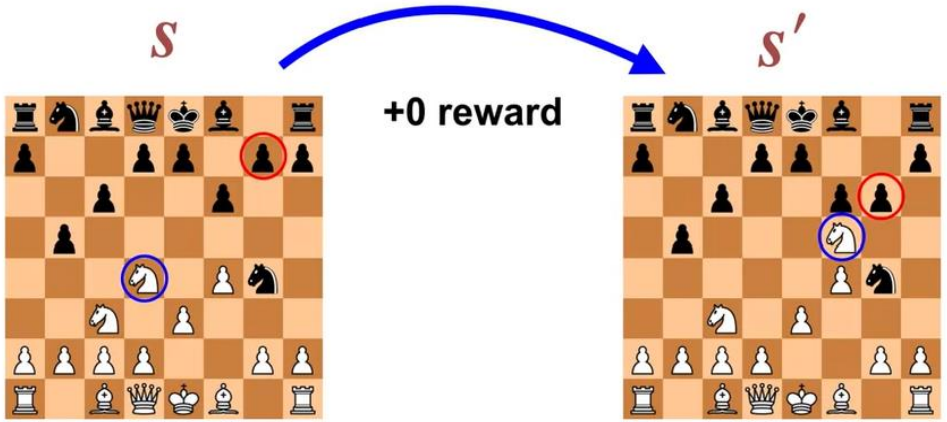
Action-value functions

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

Value functions predict rewards into the future

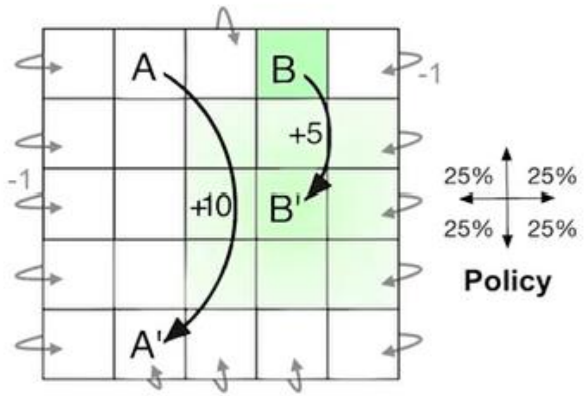


Value function example: Chess



$P(\text{win}) = V_\pi(s) = 0.34$ $\quad$ $V_\pi(s') = 0.31$

- Grid world example



$\gamma = 0.9$

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Value functions

Summary

- State-value functions represent the expected return from a given state under a specific policy
- Action-value functions represent the expected return from a given state after taking a specific action, later following a specific policy

Dynamic programming algorithms

- Use the Bellman equations to define iterative algorithms for both policy evaluation and control.

Policy evaluation vs control

- Policy evaluation

$$\pi \longrightarrow v_\pi$$

Recall that

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

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

Policy evaluation vs control

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

Recall that

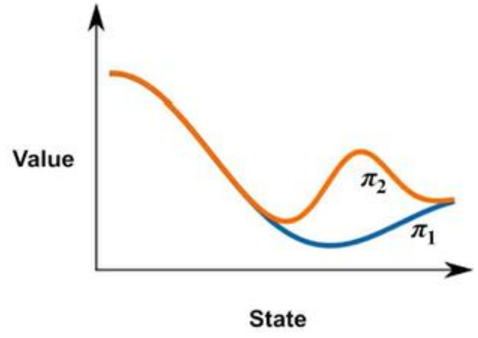
$$\pi, p, \gamma \longrightarrow \boxed{\text{Linear System Solver}} \longrightarrow v_\pi$$

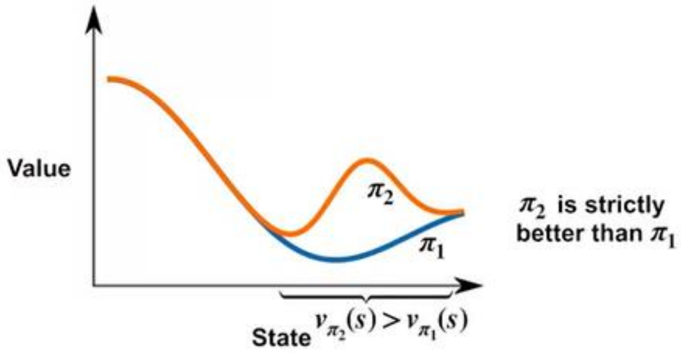
In practice

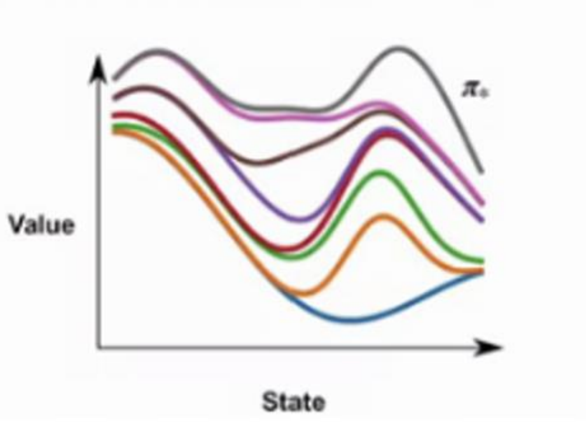
$$\pi, p, \gamma \longrightarrow \boxed{\text{Dynamic Programming}} \longrightarrow v_\pi$$

Policy Evaluation vs Control

- Control is the task of improving a policy







Policy Evaluation vs Control

$\pi, p, \gamma \longrightarrow$ Dynamic Programming $\longrightarrow v_\pi$ **Policy Evaluation**

$p, \gamma \longrightarrow$ Dynamic Programming $\longrightarrow \pi_*$ **Control**

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r \textbf{?}(s', r | s, a)\left[r + \gamma v_\pi(s')\right]$$

$$q_\pi(s,a) = \sum_{s'} \sum_r \textbf{?}(s', r | s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\right]$$

$$v_*(s) = \max_a \sum_{s'} \sum_r \textbf{?}(s', r | s, a)\left[r + \gamma v_*(s')\right]$$

$$q_*(s,a) = \sum_{s'} \sum_r \textbf{?}(s', r | s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]$$

Policy Evaluation vs Control

- Summary

  - Policy evaluation is the task of determining the state-value function v_π, for a particular policy π
  - Control is the task of improving an existing policy
  - Dynamic programming techniques can be used to solve both these tasks, if we have access to the dynamics function p

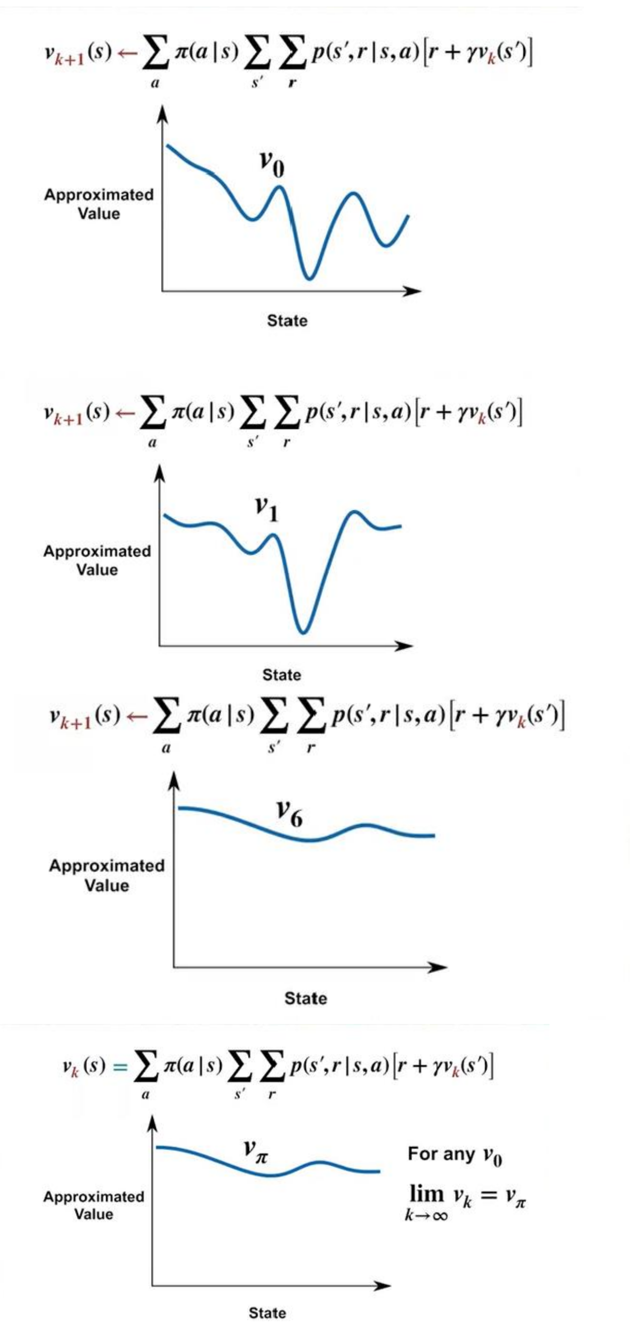
Iterative Policy Evaluation

Iterative Policy Evaluation in a Nutshell

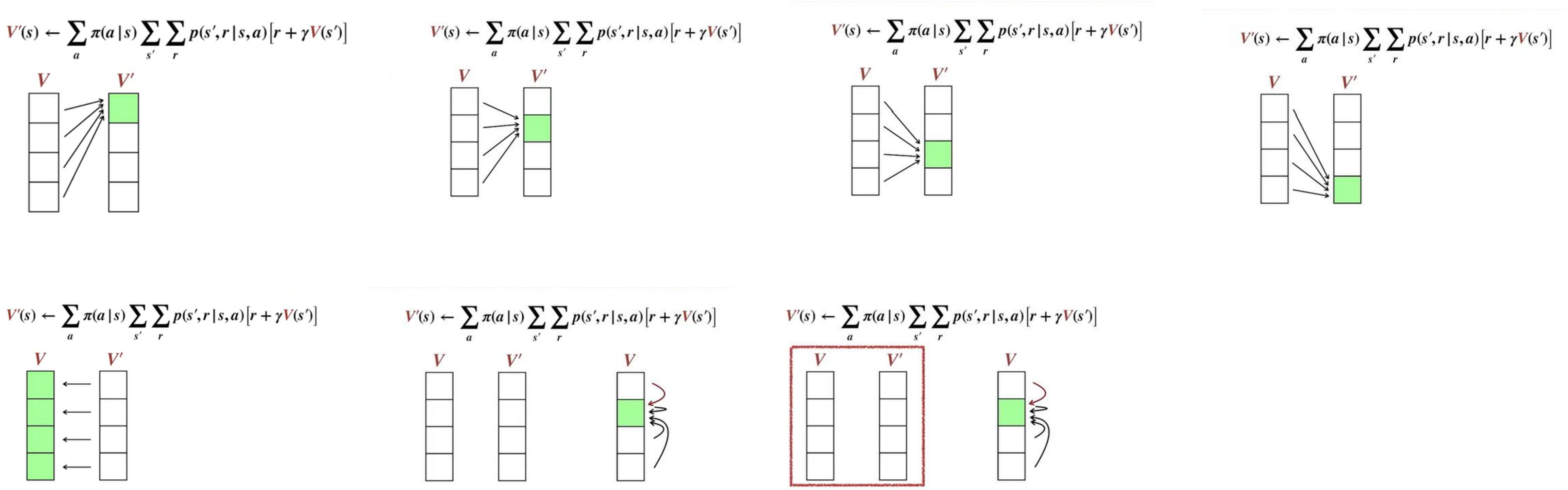
$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

$$\downarrow$$

$$v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma v_k(s')\right]$$

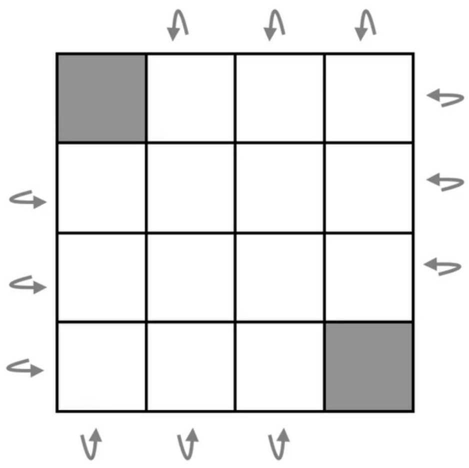


Iterative Policy Evaluation



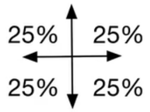
Iterative Policy Evaluation

$$V'(s) \leftarrow \sum_{a} \pi(a \mid s) \sum_{s'} \sum_{r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$



$R = -1$

$\gamma = 1$



Policy

V

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

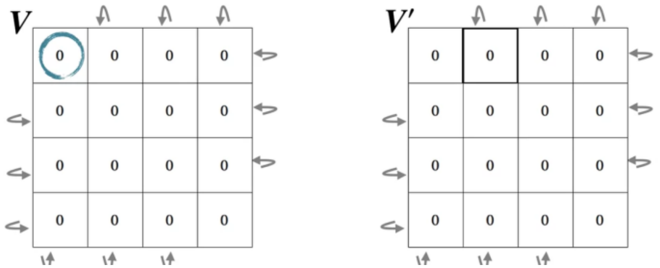
V′

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Iterative Policy Evaluation

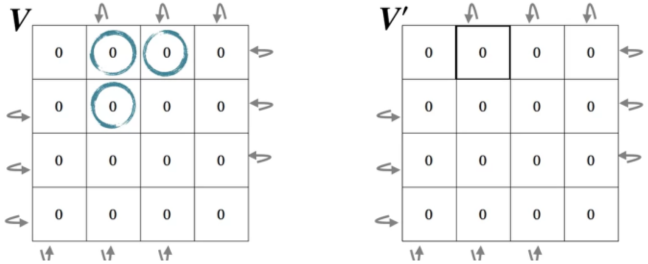
$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

$0.25 * (-1 + 0)$

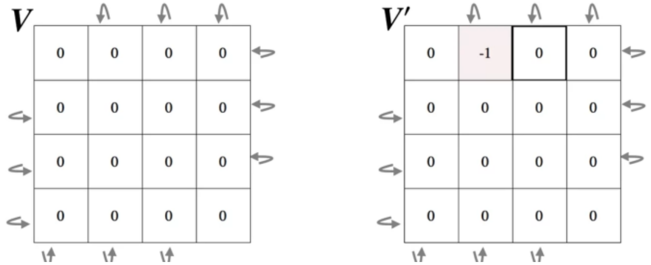


$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

$0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) = -1$

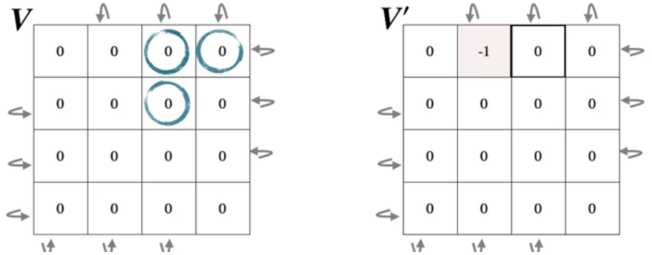


$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$



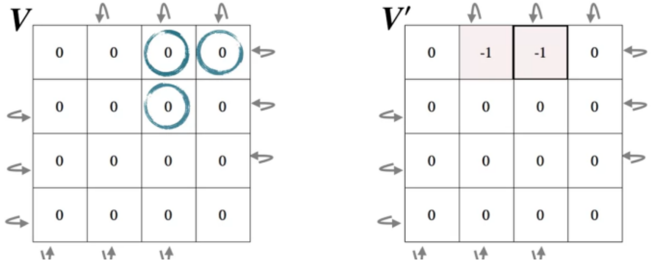
$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

$0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) = -1$

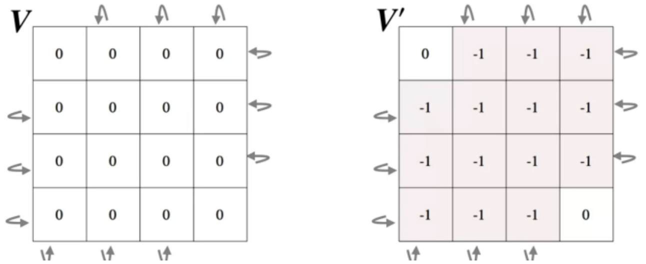


$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

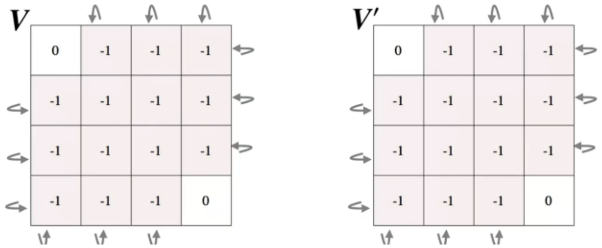
$0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) = -1$



$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

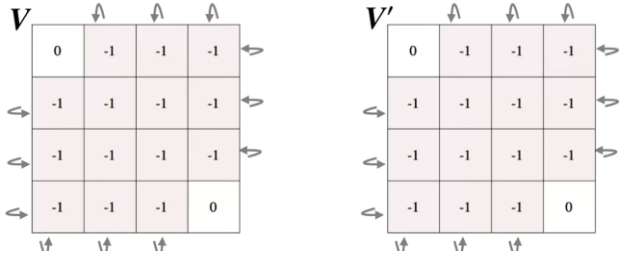


$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$



$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

$v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$



Iterative Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated

$V \leftarrow \vec{0}, V' \leftarrow \vec{0}$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s',r \mid s,a)[r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$$

$V \leftarrow V'$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

$\theta = 0.001$

V

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

V'

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

$\theta = 0.001$ $\Delta = 1.0$

V

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

V'

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

V

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

V'

0	-1.7	-2	-2
-1.7	-2	-2	-2
-2	-2	-2	-1.7
-2	-2	-1.7	0

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

V

0	-1.7	-2	-2
-1.7	-2	-2	-2
-2	-2	-2	-1.7
-2	-2	-1.7	0

V'

0	-2.4	-2.9	-3
-2.4	-2.9	-3	-2.9
-2.9	-3	-2.9	-2.4
-3	-2.9	-2.4	0

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

V

0	-2.4	-2.9	-3
-2.4	-2.9	-3	-2.9
-2.9	-3	-2.9	-2.4
-3	-2.9	-2.4	0

V'

0	-3.1	-3.8	-4
-3.1	-3.7	-3.9	-3.8
-3.8	-3.9	-3.7	-3.1
-4	-3.8	-3.1	0

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

V

0	-3.1	-3.8	-4
-3.1	-3.7	-3.9	-3.8
-3.8	-3.9	-3.7	-3.1
-4	-3.8	-3.1	0

V'

0	-3.7	-4.7	-4.9
-3.7	-4.5	-4.8	-4.7
-4.7	-4.8	-4.5	-3.7
-4.9	-4.7	-3.7	0

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

V

0	-3.7	-4.7	-4.9
-3.7	-4.5	-4.8	-4.7
-4.7	-4.8	-4.5	-3.7
-4.9	-4.7	-3.7	0

V'

0	-4.2	-5.5	-5.8
-4.2	-5.2	-5.6	-5.5
-5.5	-5.6	-5.2	-4.2
-5.8	-5.5	-4.2	0

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

$\Delta < 0.001$

V

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

V'

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

- Summary
    - We can turn the Bellman equation into an update rule, to iteratively compute value functions

Monte Carlo methods for prediction and control

- In RL Monte Carlo methods allow us to estimate values directly from experience, from sequences of states, actions and rewards.
- Learning from experience is striking because the agent can accurately estimate a value function without prior knowledge of the environment dynamics.
- To use a pure Dynamic Programming approach, the agent needs to know the environments transition probabilities.
  - In some problems we do not know the environment transition probabilities
  - The computation can be error-prone and tedious estimate without saving lists of returns.

What is Temporal Difference (TD) Learning?

- Let's discuss a small modification to Monte Carlo policy evaluation method.
- We can use this formula to incrementally update our estimated value.
- This algorithm forms a Monte Carlo estimate without saving lists of returns.
- To compute the return, we must take samples of full trajectories. This means we don’t learn during the episode, but we want to be able to learn incrementally before the end of the episode.
- We must produce a new update target to achieve this goal.

What is Q-learning?

- Temporal Difference Learning Method
  - Solves the Bellman equation using samples from the environment.
    - Instead of using the standard Bellman equation, Q-learning uses the Optimality Equation for action values.
    - The optimality equations enable Q-learning to directly learn instead of switching between policy improvement and policy evaluation steps

Q-learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t))$$

$$Q_*(s, a) = \sum_{s', r} p(s', r | s, a) \left(r + \gamma \max_{a'} q_\pi(s', a') \right)$$

Q-learning algorithm

- Q-learning learns about the **best action** it could possibly take rather than the actions it actually takes.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
        $S \leftarrow S'$
    until S is terminal

References

- Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto
- Reinforcement Learning Fundamentals, Coursera https://www.coursera.org/learn/fundamentals-of-reinforcement-learning
- Reinforcement Learning Specialization, Coursera