Reinforcement learning

Dr. Aissa Boulmerka

The National School of Artificial Intelligence
aissa.boulmerka@ensia.edu.dz

2024-2025

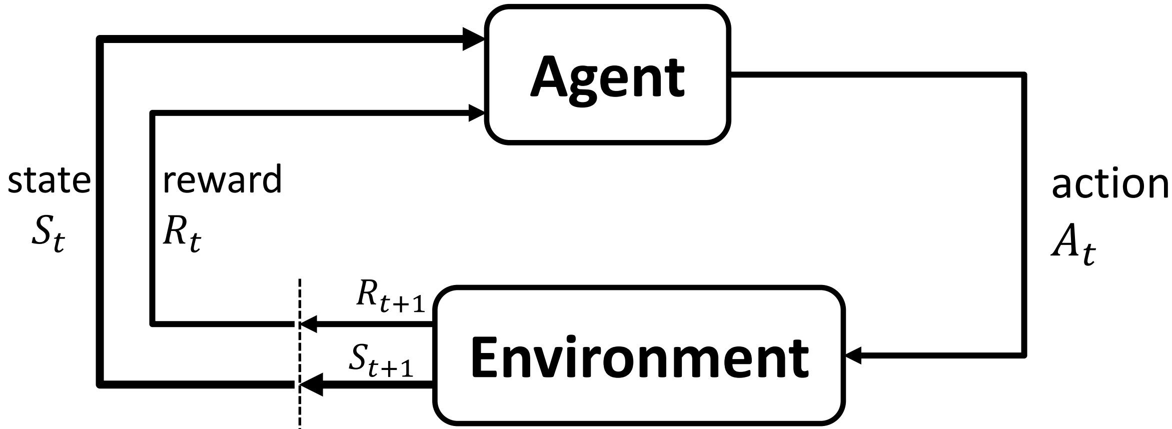
Chapter 3
Finite Markov Decision Processes

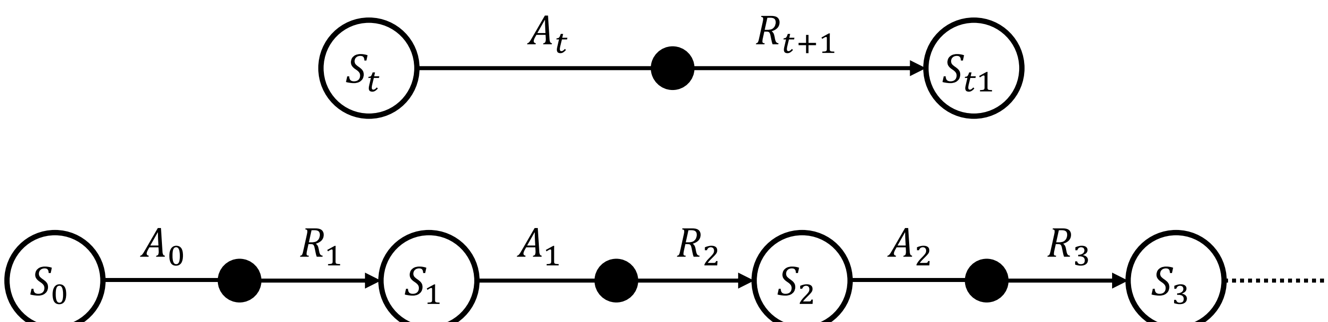
Outline

- Markov Decision Processes
- Bellman Equation
- Bellman's Optimality equations

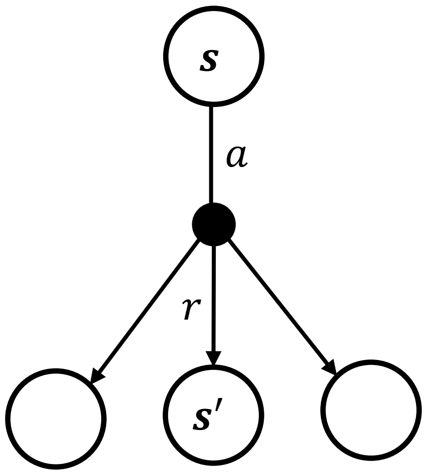
Markov Decision Processes

Introduction





The dynamics of MDP



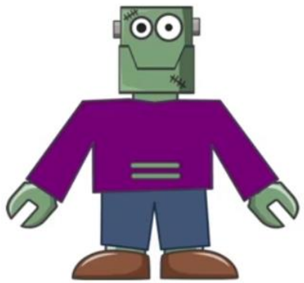
$$p(s', r | s, a)$$

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

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

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

Example 1: Recycling robot

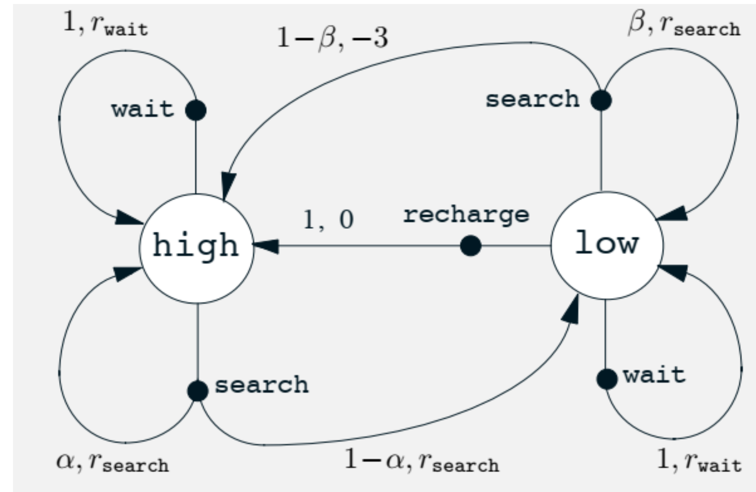


$$\mathcal{S} = \{\boldsymbol{low}, \boldsymbol{high}\}$$

$$\mathcal{A}(low) = \{\boldsymbol{search}, \boldsymbol{wait}, \boldsymbol{recharge}\}$$
$$\mathcal{A}(high) = \{\boldsymbol{search}, \boldsymbol{wait}\}$$

Example1: Recycling robot

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



Generalization of MDP formalism

- The MDP framework can be used to formalize a wide variety of sequential decision-making problems, in many different ways.

- **States:**
  - States can be low-level sensory readings, for example, in the pixel values of the video frame.
  - They can also be high-level such as object descriptions.
- **Actions:**
  - Actions can be low-level, such as the wheel speed of this robot.
  - Actions can also be high-level, such as go to the charging station.
- **Time-steps:**
  - Time-steps can be very small or very large. For example, they can be one millisecond or one month.

Example 2: Robot arm in a pick-and-place task

Task: The goal of the robot is to pick-and-place objects.

There are many ways to formalize this task:
State: The state could be the readings of the joint angles and velocities.
Action: the amount of voltage applied to each motor.
Reward: **+100** for successfully placing each object.
-1 for each unit of energy consumed.

Goal of an agent: formal definition

- In RL, the goal of the agent is to maximize future reward.

- Formally, the return at time step t, is simply the sum of rewards obtained after time step t. We denote the return with the G_t.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots$$

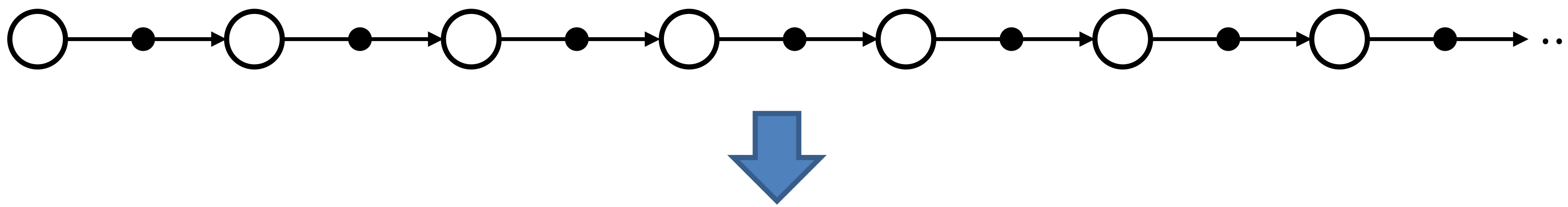
- The return is a random variable because the dynamics of the MDP can be **stochastic**.

- In general, many different trajectories from the same state are possible. This is why we **maximize the expected return**.

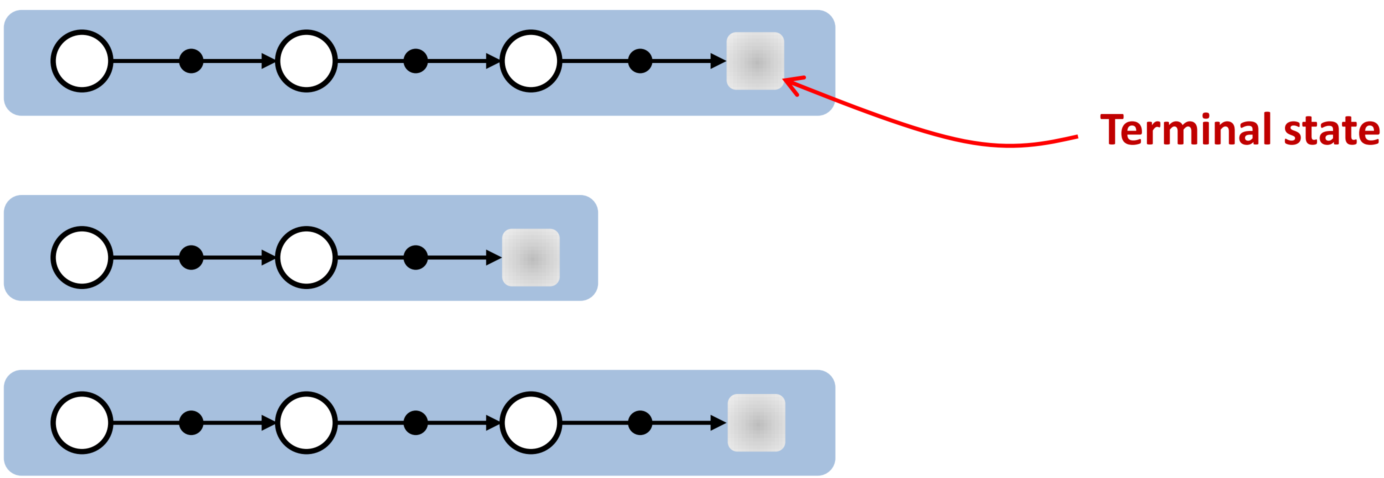
- For this to be well-defined, the sum of rewards must be finite.

$$\mathbb{E}(G_t) = \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \cdots R_T]$$

Episodic task



1 task = 3 episodes



Episodic and continuing tasks

Episodic task

- Interaction breaks naturally into episodes
- Each episode ends in a terminal state
- Episodes are independent

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots R_T$$

Continuing task

- Interaction goes on continually
- No terminal state

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots$$
$$= \infty?$$

$\Rightarrow$ **Discounting**

Discounting

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots R_{t+k} + \cdots$$

How to make sure G_t is **finite**?

Discount the rewards in the future by γ where $0 < \gamma$ <1

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

where γ is a parameter, $0 < \gamma$ <1, called the **discount rate**.

Effect of γ on agent behavior

$$\boldsymbol{G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \gamma^{k-1} R_{t+k} + \cdots}$$

- The **discount rate $\boldsymbol{\gamma}$** determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately.
  - $\boldsymbol{\gamma < 1}$, the infinite sum has a finite value as long as the reward sequence $\{R_k\}$ is bounded.
  - $\boldsymbol{\gamma = 0}$, in this case, the agent is concerned only with maximizing immediate rewards. The agent is called **Short-sighted agent.**
  - $\boldsymbol{\gamma \to 1}$, the return objective takes future rewards into account more strongly. The agent becomes **Far-sighted agent**.

Recursive nature of returns

- Returns at successive time steps are **related to each other** in a way that is important for the theory and algorithms of reinforcement learning:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

$$G_t = R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots)$$

$$\boldsymbol{G_t = R_{t+1} + \gamma G_{t+1}}$$

- Although the return is a sum of an **infinite** number of terms, it is still **finite** if the reward is nonzero and constant.
- **Example:** if $\gamma < 1$, if the reward is a constant $+\mathbf{1}$, then the return is:

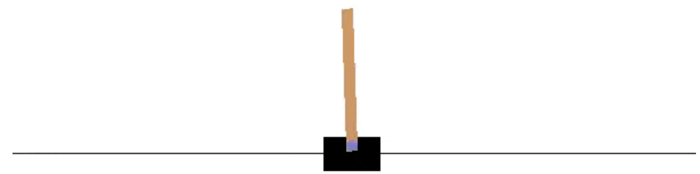
$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}.$$

Examples of episodic and continuing tasks

Pole-Balancing:

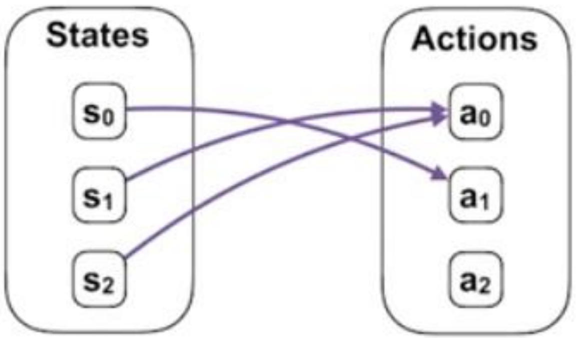
The goal is to move a cart along a track to **keep a hinged pole from falling**, also known as the inverted pendulum problem.

- It can be seen as an **episodic task**, where each attempt to balance the pole is an episode.
- A **reward of +1** can be given for each time step without failure, with the total reward being the number of steps before failure.
- Alternatively, it can be treated as a **continuous task** using discounting.



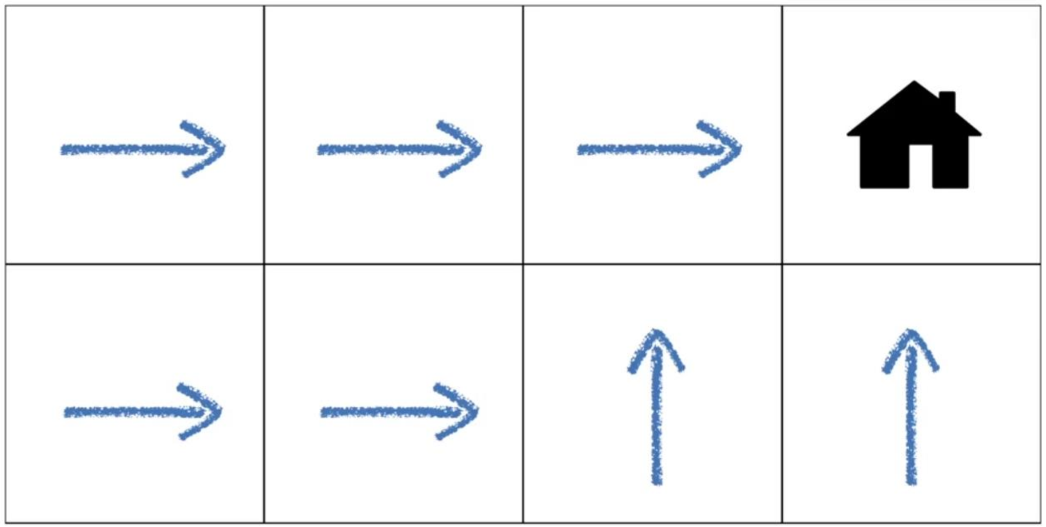
Deterministic policy notation

- **A policy is a mapping from states to probabilities of selecting each possible action.**



State	Action
s_0	a_1
s_1	a_0
s_2	a_0

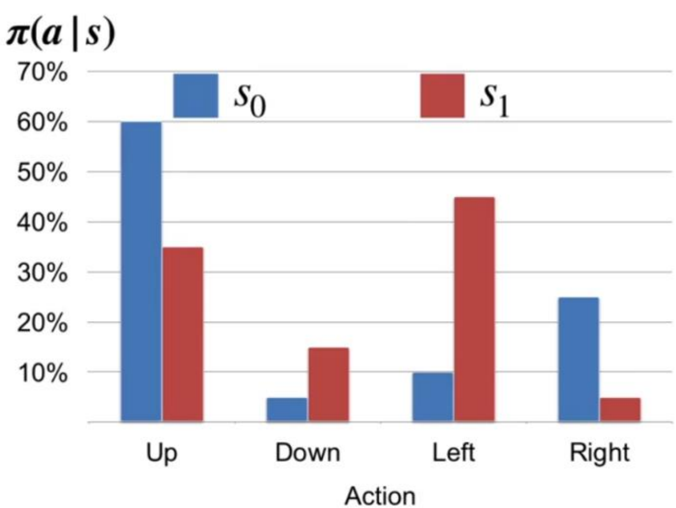
Example of deterministic policy

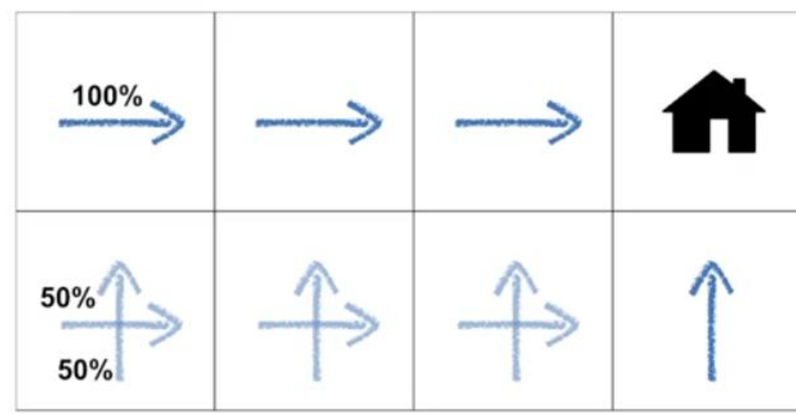


Stochastic policy

- If the agent is following a stochastic policy π at time t, then $\pi(a|s)$ is the **probability** that $A_t = a$ if $S_t = s$.
- **Note** that $\sum_{a \in \mathcal{A}(s)} \pi(a|s) = 1$ and $\pi(a|s) \leq 1$.

Example of stochastic policy





Value Functions

- The **value function** of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following the policy π.
- For MDPs, we can define v_π formally by

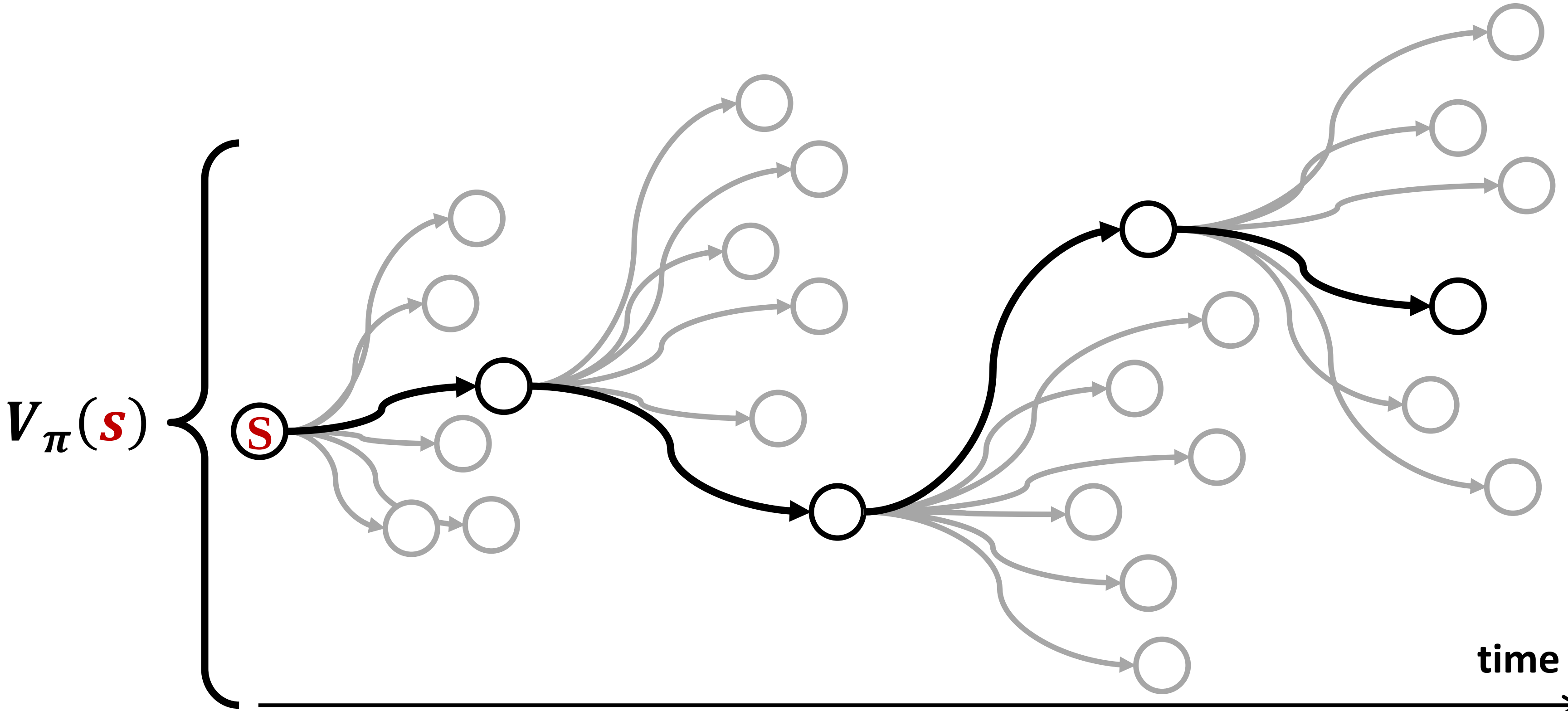
$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | \, S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \quad \text{for all } s \in \mathcal{S}$$

- Note that the value of the terminal state, if any, is always zero. We call the function v_π the ***state-value function for policy π***.
- Similarly, the value of taking action a in state s under a policy π : $q_\pi(s, a)$, is the expected return starting from s, taking the action a, and following policy π :

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

- We call $q_\pi(s, a)$ the **action-value function for policy π**.

Value function predict rewards into the future



Bellman Equation

State-value Bellman equation

- A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy **recursive relationships**.
- For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\big] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \qquad \text{for all } s \in \mathcal{S}
\end{aligned}
$$

- This equation is the **Bellman equation** *for* v_π.
- It expresses a relationship between the value of a state and the values of its **successor states**.

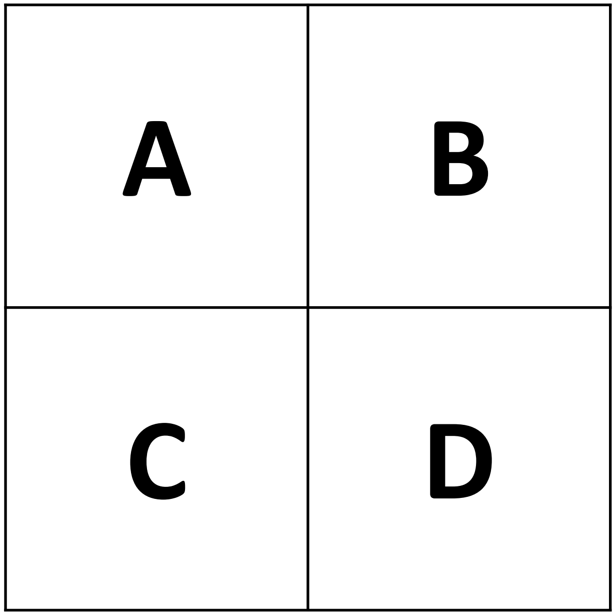
Action-value Bellman equation

- A similar equation for the **action-value function**.
- It will be a recursive equation for the value of a state action pair in terms of its possible successors state action pairs.

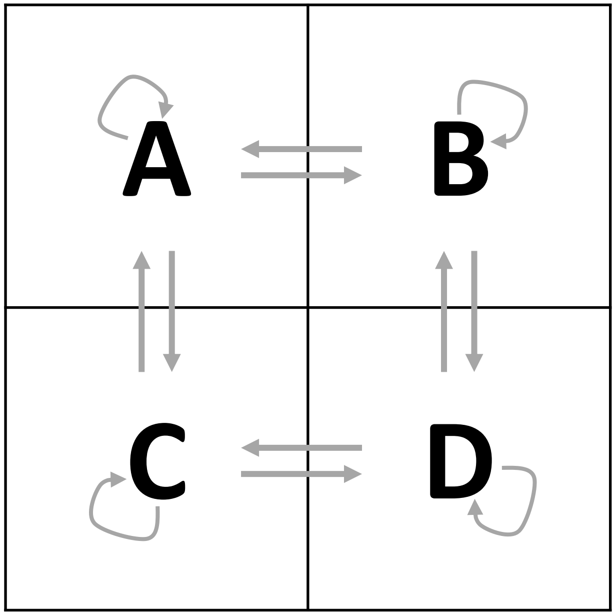
$$
\begin{aligned}
\mathrm{q}_\pi(s,a) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s, \mathrm{A}_t = \mathrm{a}] \\
&= \sum_{s'}\sum_{r} p(s',r|s,a)\big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\big] \\
&= \sum_{s'}\sum_{r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s')\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s', \mathrm{A}_{t+1} = a']\right] \\
&= \sum_{s'}\sum_{r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s')\mathrm{q}_\pi(s',a')\right]
\end{aligned}
$$

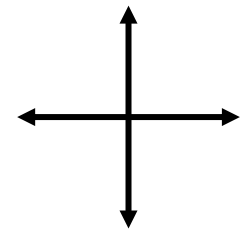
- This equation is the Bellman equation for the action-value function $\mathrm{q}_\pi(s,a)$.
- Similar to state-value function, this equation provide relationships between the state-action pair and the possible **future state-action** pairs.

Example: Gridworld



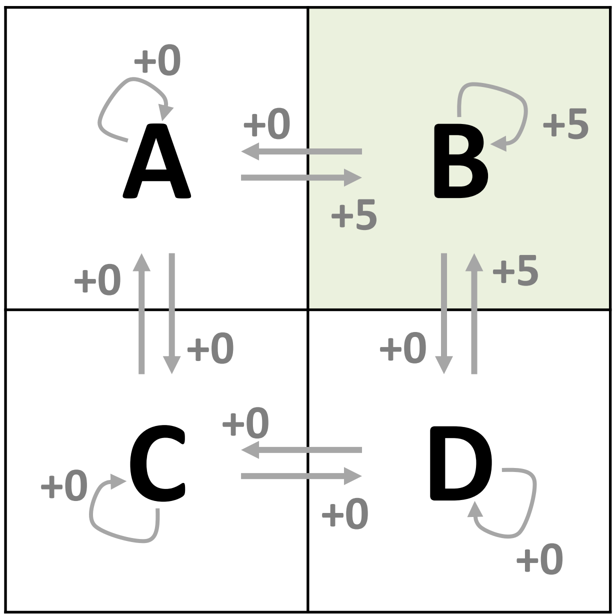
Example: Gridworld

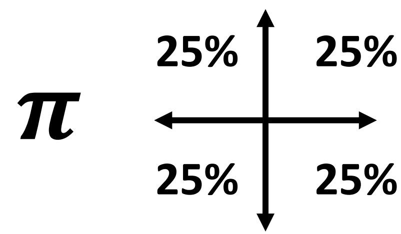




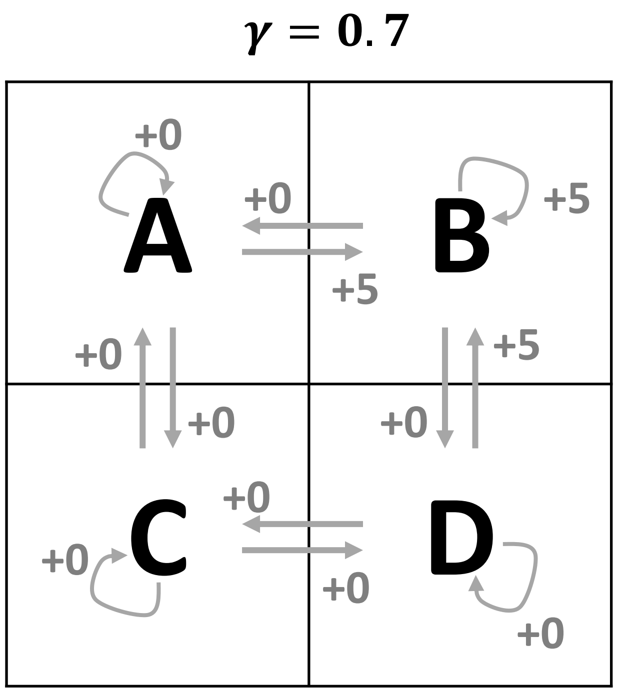
Example: Gridworld

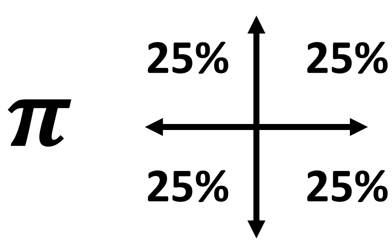
$\gamma = 0.7$





Example: Gridworld





$$V_\pi(A) \doteq \mathbb{E}_\pi[G_t \mid S_t = A]$$

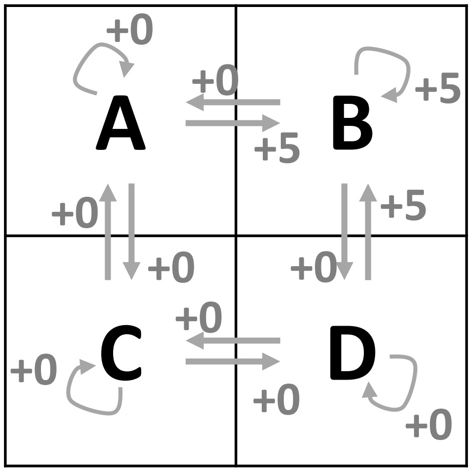
$$V_\pi(B) \doteq \mathbb{E}_\pi[G_t \mid S_t = B]$$

$$V_\pi(C) \doteq \mathbb{E}_\pi[G_t \mid S_t = C]$$

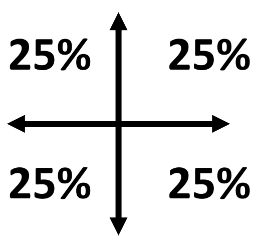
$$V_\pi(D) \doteq \mathbb{E}_\pi[G_t \mid S_t = D]$$

Example: Gridworld

$\gamma = 0.7$



π



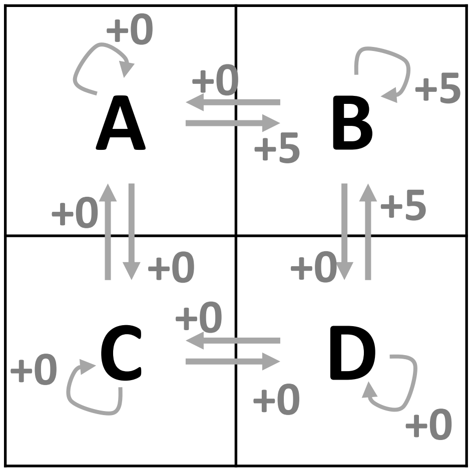
$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$$

$$\begin{cases} V_\pi(A) = \frac{1}{4}\left(5 + 0.7V_\pi(B)\right) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(A) \\ V_\pi(B) = \frac{1}{2}\left(5 + 0.7V_\pi(B)\right) + \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) \\ V_\pi(C) = \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) + \frac{1}{2}0.7V_\pi(C) \\ V_\pi(D) = \frac{1}{4}\left(5 + 0.7V_\pi(B)\right) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(D) \end{cases}$$

A system of for equations for 4 variables that can be solved by hand or by an automatic equation solver.

Example: Gridworld

$\gamma = 0.7$



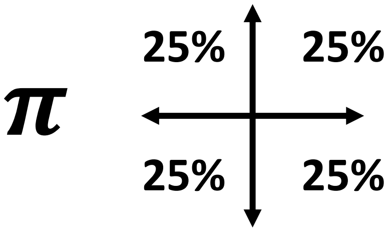
$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$$

$$V_\pi(A) = 4.2$$

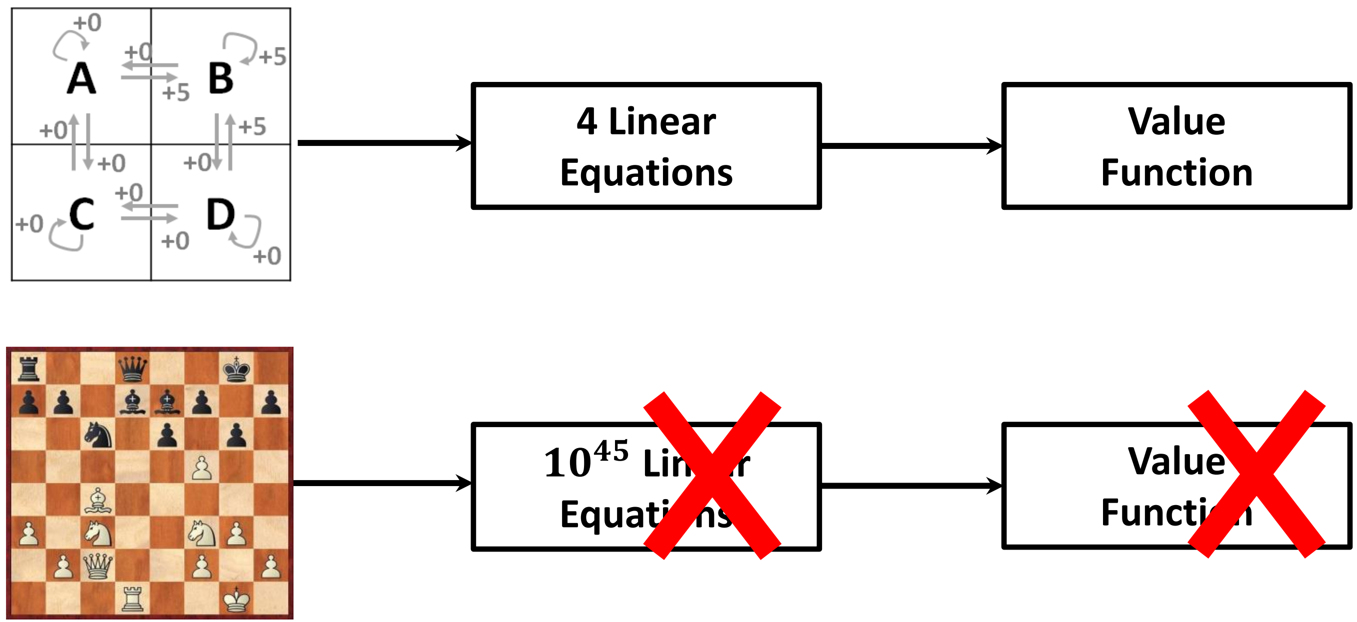
$$V_\pi(B) = 6.1$$

$$V_\pi(C) = 2.2$$

$$V_\pi(D) = 4.2$$



We can only directly solve small MDPs



- We can use **Bellman Equations** to solve for a **value function** by writing a **system of linear equations**.
- We can only solve **small MDPs** directly, but **Bellman Equations** will factor into solutions we see later for **large MDPs**.

Optimal Policies

- A policy π_1 is defined to be **better than or equal** to a policy π_2 if its **expected return** is greater than or equal to that of π_2 for all states. In other words,

$$\boldsymbol{\pi_1 \geq \pi_2 \text{ if } v_{\pi_1}(s) \geq v_{\pi_2}(s),\ \forall\, s \in \mathcal{S}.}$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy $\boldsymbol{\pi_*}$ that is better than or equal to all other policies,*

$$\boldsymbol{\pi_* \ \geq \ \pi, \forall \pi}$$

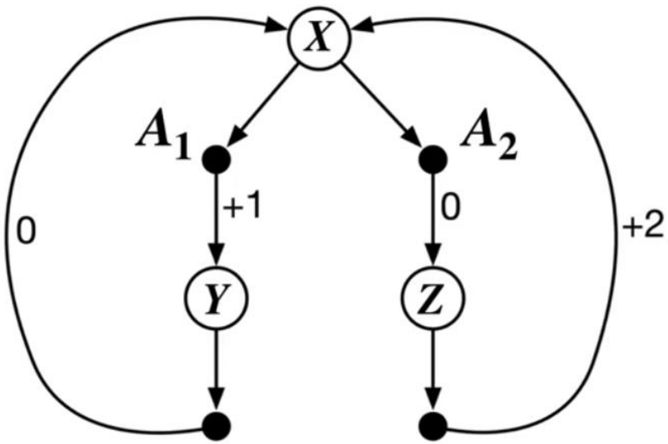
- *All optimal policies achieve the optimal value function,*

$$\boldsymbol{v_{\pi_*}(s) = v_*(s)}$$

- *All optimal policies achieve the optimal action-value function,*

$$\boldsymbol{q_{\pi_*}(s, a) = q_*(s, a)}$$

Example



$\pi_1(X) = A_1 \quad \pi_2(X) = A_2$

What is the optimal policy?

$\gamma = 0$

$v_{\pi_1}(X) = 1$

$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

$v_{\pi_1}(X) = 1 + 0.9 \times 0 + 0.9^2 \times 1 + \cdots = \sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1 - 0.9^2} \approx 5.3$

$v_{\pi_2}(X) = 0 + 0.9 \times 2 + 0.9^2 \times 0 + \cdots = \sum_{k=0}^{\infty} (0.9)^{2k+1} * 2 = \frac{0.9}{1 - 0.9^2} * 2 \approx 9.5$

We can only directly solve small MDPs

- In general it is not possible to implement this solution **exactly**. Even if we limit ourselves to deterministic policies, the number of possible policies is $|\mathcal{A}|^{|\mathcal{S}|}$.
- We can use a **brute force search** to compute the value function for every policy to find the optimal policy ⇒ **intractable** for even moderately large MDPs.
- Fortunately, there's a better way to organize the **search of the policy space**.
- The solution will come in the form of yet another set of Bellman equations, called the **Bellman's Optimality equations**. We consider a variety of such methods in the following chapters.

Bellman's Optimality Equations

Optimal value functions

Recall that

$\boldsymbol{\pi_1 \geq \pi_2}$ **if and only if** $\boldsymbol{v_{\pi_1}(s) \geq v_{\pi_2}(s)}$ **for all** $\boldsymbol{s \in \mathcal{S}}$

$\boldsymbol{v_*}$ $\quad v_{\pi_*}(s) \doteq \mathbb{E}_{\pi_*}[G_t | S_t = s] = \max_{\pi} v_\pi(s)$ for all $s \in \mathcal{S}$

$\boldsymbol{q_*}$ $\quad q_{\pi_*}(s,a) = \max_{\pi} q_\pi(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$

Optimal value functions

Recall that

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

$$v_*(s) = \sum_a \pi_*(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_*(s')]$$

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

Bellman Optimality Equation for v_*

Optimal value functions

Recall that

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

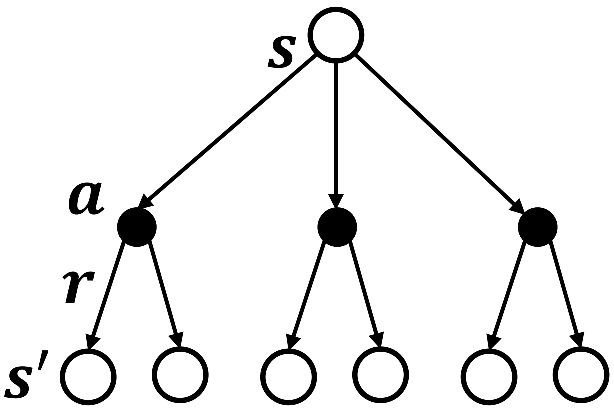
$$\boldsymbol{q}_*(s,a) = \sum_{s'} \sum_{r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi_*(a'|s')\, q_\pi(s',a') \right]$$

$$\boldsymbol{q}_*(s,a) = \sum_{s'} \sum_{r} p(s',r|s,a) [r + \gamma \max_{a'} \boldsymbol{q}_*(s',a')]$$

Bellman Optimality Equation for $\boldsymbol{q}_*$

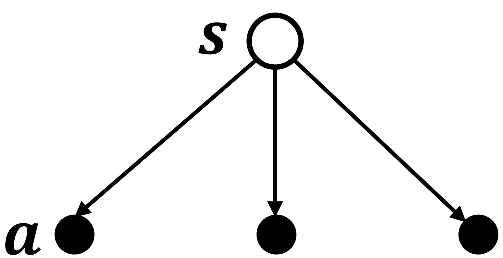
Finding an Optimal Policy

- An optimal policy can be found by maximizing over $v_*(s)$



$$\pi_*(s) = \operatorname*{\mathbf{argmax}}_a \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma \boldsymbol{v}_*(\boldsymbol{s}')]$$

- Similarly, it can be found by maximizing over $q_*(s, a)$

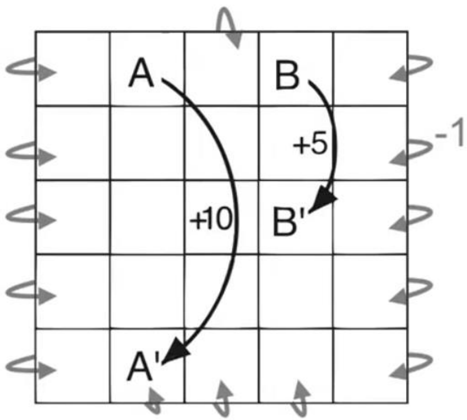


$$\pi_*(s) = \operatorname*{\mathbf{argmax}}_a \boldsymbol{q}_*(\boldsymbol{s}, \boldsymbol{a})$$

Example: Solving the Gridworld

- Suppose we solve the **Bellman optimality equation** for v_* for the Gridworld.
- State $\boldsymbol{A}$ is followed by a reward of $+10$ and transition to state $\boldsymbol{A'}$, while state **B** is followed by a reward of **+5** and transition to state $\boldsymbol{B'}$ and supposing $\boldsymbol{\gamma = 0.9}$
- The **optimal policy** π_* can be found using the **Bellman optimality equation** :

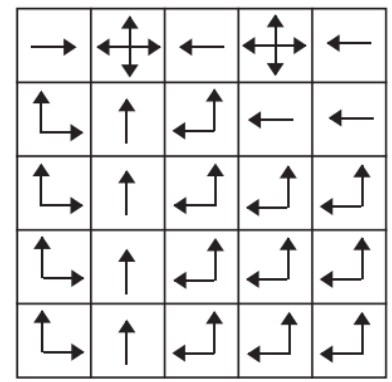
$$\pi_*(s) = \underset{a}{\mathbf{argmax}} \sum_{s'} \sum_{r} p(s', r | s, a)[r + \gamma v_*(s')]$$



Gridworld

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

v_*



π_*

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

Solving the optimal value functions

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

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

- Bellman Optimality Equation is **non-linear**

Non linear

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

- No closed (exact) form solution (in general).
- Many **iterative solution** methods such as : Value Iteration, Policy Iteration, Q-learning, Sarsa etc.

Thank you!
Q/A