REINFORCEMENT LEARNING

Nguyen Do Van, PhD

VIASM
Vietnam Institute for
Advanced Study in Mathematics

Reinforcement Learning

- Introduction
- Markov Decision Process
- Dynamic Programming

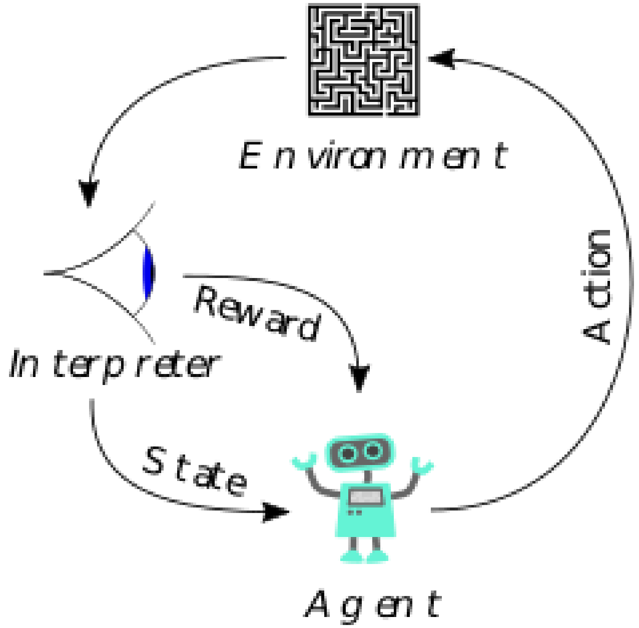
REINFORCEMENT LEARNING INTRODUCTION

Intelligent agents learning and acting

Sequence of decision, reward

Reinforcement learning: What is it?

- Making good decision to do new task: fundamental challenge in AI, ML
- Learn to make good sequence of decisions
- Intelligent agents learning and acting
  - ❑ Learning by trial-and-error, in real time
  - ❑ Improve with experience
  - ❑ Inspired by psychology:
    - Agents + environment
    - Agents select action to maximize *cumulative* rewards



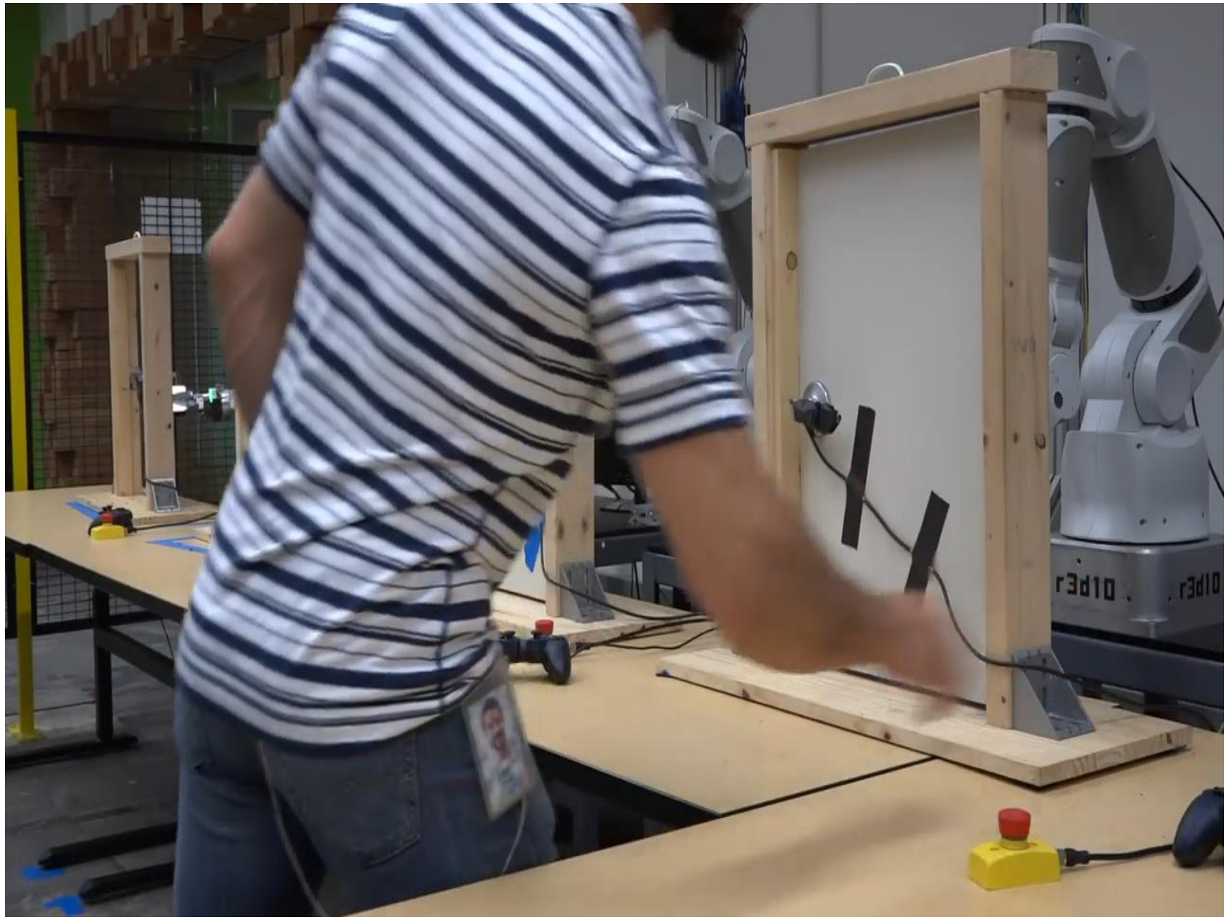
Characteristics of Reinforcement Learning

- 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

RL Applications

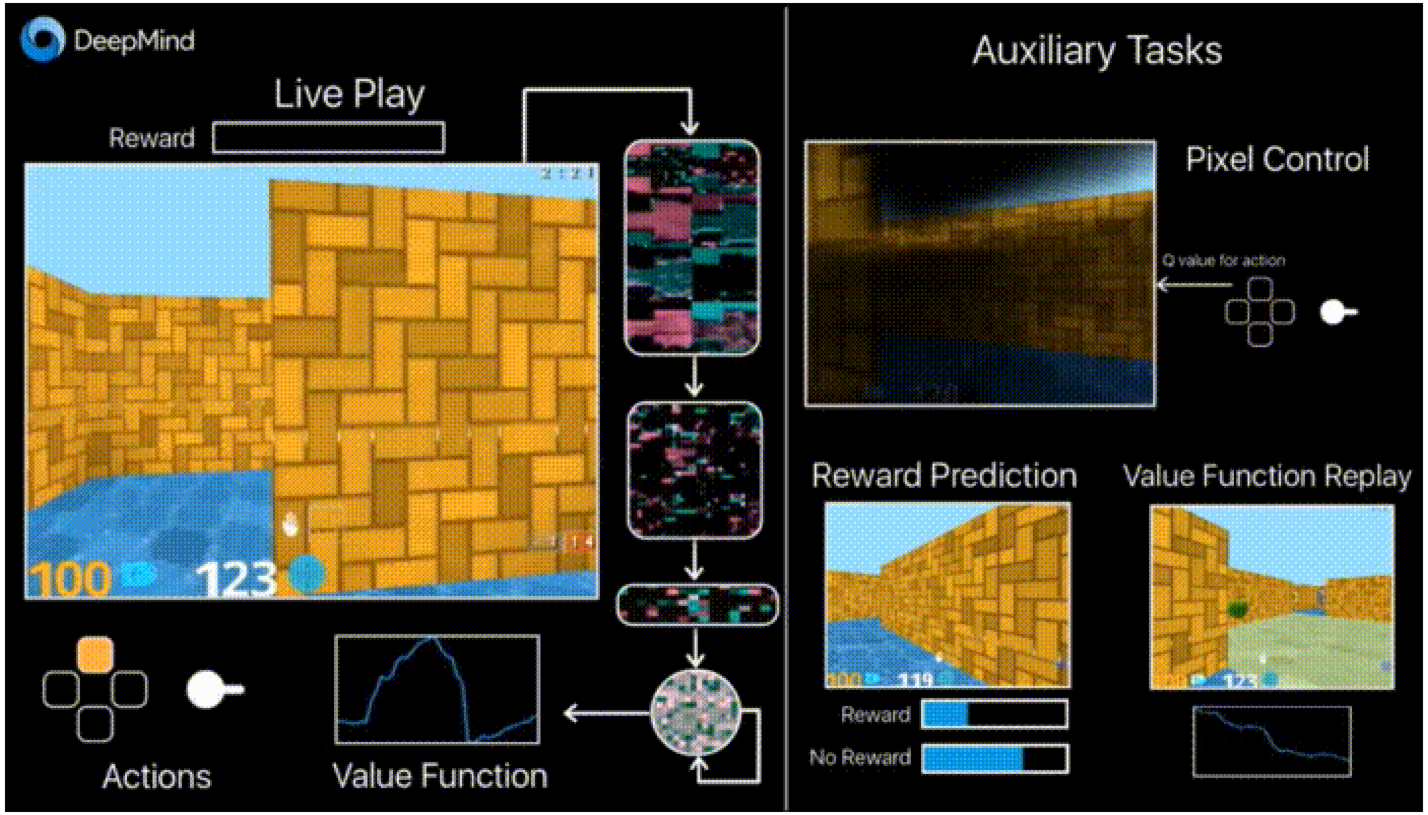
- Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM2017)
  - ❑ Robotics
  - ❑ Video games
  - ❑ Conversational systems
  - ❑ Medical intervention
  - ❑ Algorithm improvement
  - ❑ Improvisational theatre
  - ❑ Autonomous driving
  - ❑ Prosthetic arm control
  - ❑ Financial trading
  - ❑ Query completion

Robotics

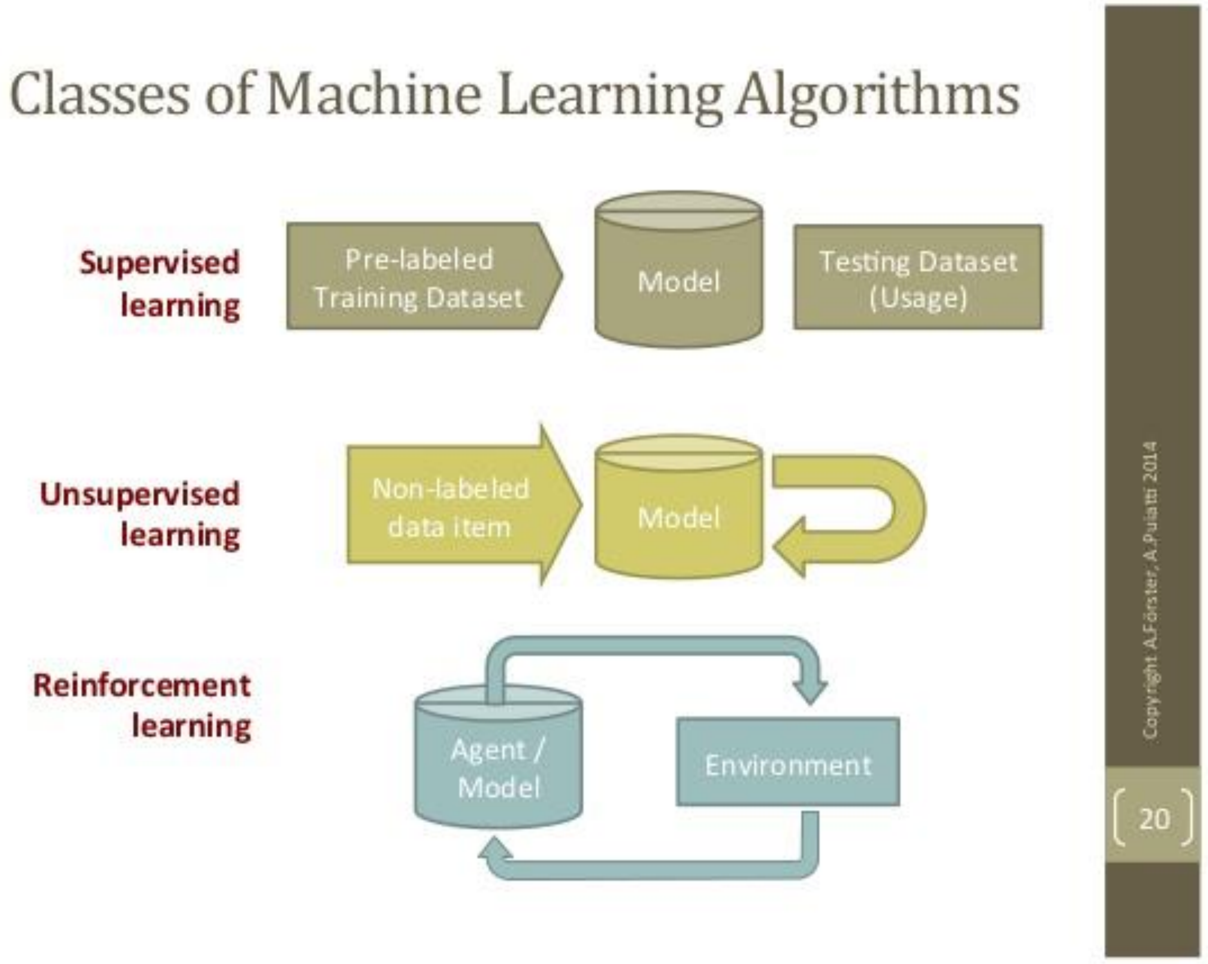


https://www.youtube.com/watch?v=ZBFwe1gF0FU

Gaming


DeepMind
Live Play
Reward
100
123
Actions
Value Function
Auxiliary Tasks
Pixel Control
Q value for action
Reward Prediction
Reward
No Reward
Value Function Replay

RL vs supervised and unsupervised learning



Classes of Machine Learning Algorithms

Copyright A.Förster, A.Puiatti 2014

Practical and technical challenges:

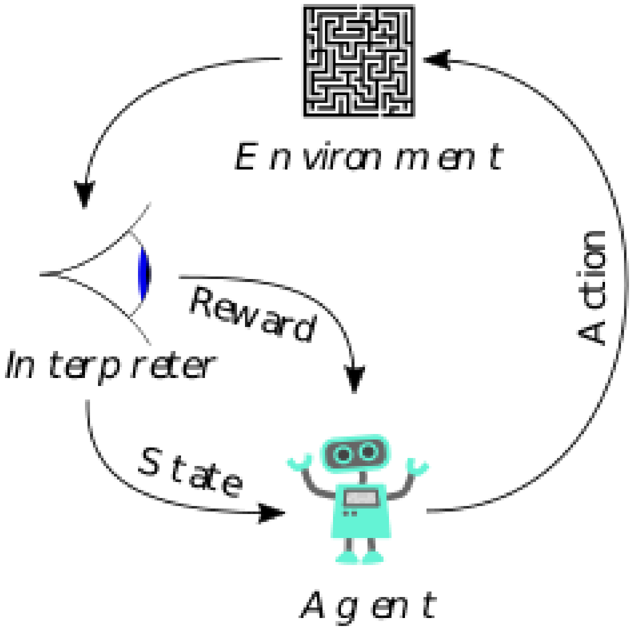
- Need to access to the environment
- Jointly learning AND planning from correlated sample
- Data distribution changes with action choice

Rewards

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximize cumulative reward
- Example:
  - ❑ Robot Navigation: (-) Crash wall, (+) reaching target…
  - ❑ Control power station: (+) producing power, (-) exceeding safety thresholds
  - ❑ Games: (+) Wining game, Killing enemy, collecting bloods, (-) mine

Agent and Environment

- At each step t the agent:
  - ❑ Executes action A_t
  - ❑ Receives observation O_t
  - ❑ Receives scalar reward R_t
- The environment:
  - ❑ Receives action A_t
  - ❑ Emits observation O_{t+1}
  - ❑ Emits scalar reward R_{t+1}
- t increments at environment step

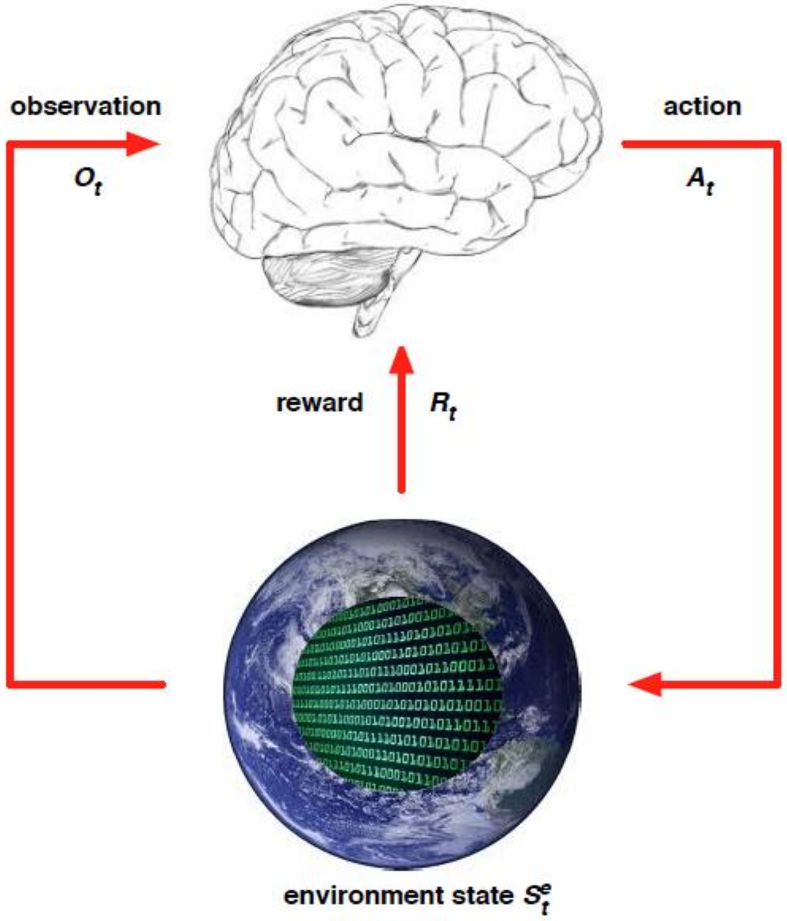


History and State

- History is the sequence of observations, actions and rewards

$$H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t$$

- State: the information to determine state in a trajectory
  - ❑ $S_t = f(H_t)$
  - ❑ Environment State: private representation of the environment
  - ❑ Agent State: agent internal representation
  - ❑ Information State (Markov Property): useful information from the history



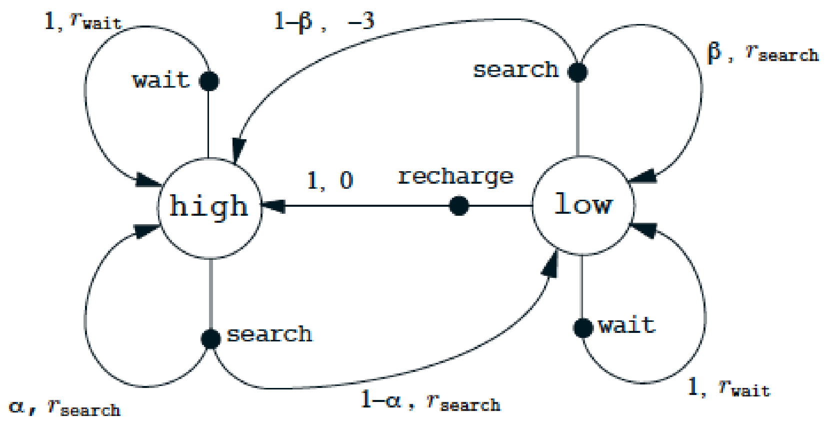
$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

Fully and Partially Observable Environments

- Full observation:
  - ❑ Agent fully observes environment state

$$O_t = S_t^a = S_t^e$$

  - ❑ Agent State = environment state = information state
  - ❑ Markov Decision Process (detail later)
- Partially observability: agent indirectly or partially observes environment
  - ❑ Robot with first view cameras
  - ❑ Agent state differ from environment state
  - ❑ Agent must construct its own state representation

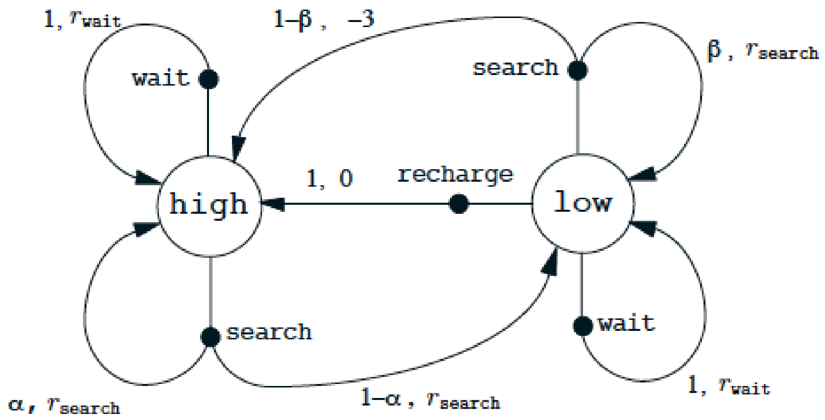


Major Component of an RL agent

- Policy - maps current state to action
- Value function - prediction of value for each state and action
- Model - agent’s representation of the environment.

Policy

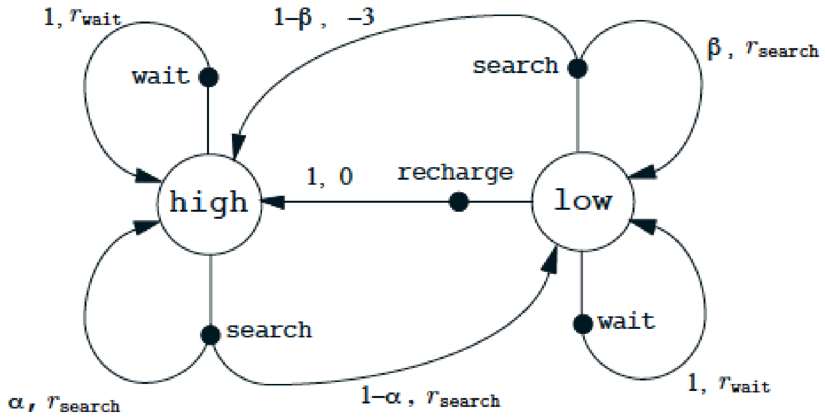
- Policy: agent's behavior, how is act in the environment
- Map from state to action
- Deterministic policy $a = \pi(s)$
- Stochastic: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$



Vietnam Institute for Advanced Study in Mathematics

Value Function

- Value Function: a prediction of future reward (how many, how much future reward the agents expect)
- Used to evaluate the goodness/badness of state
- Agent select action to chose the best state based on value function (with maximized expected reward)



$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s \right]$$

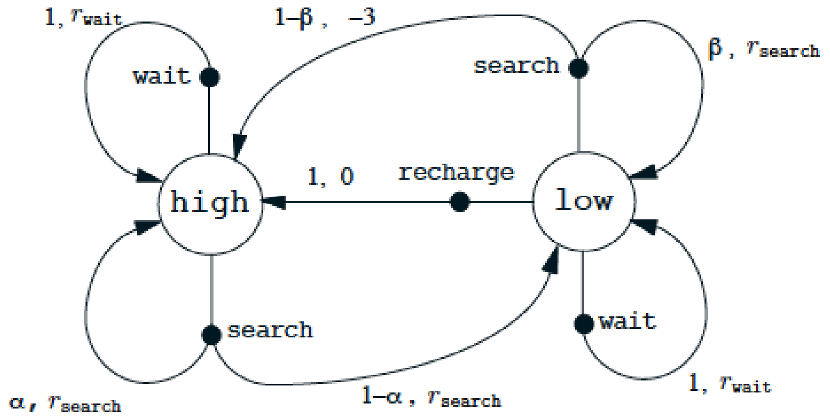
Model

- To model environments, predict what the environments will do
- P: to predict the next state

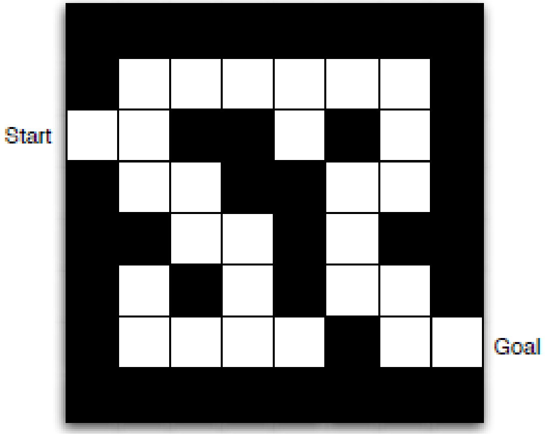
$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

- R: to predict immediate (not future) reward

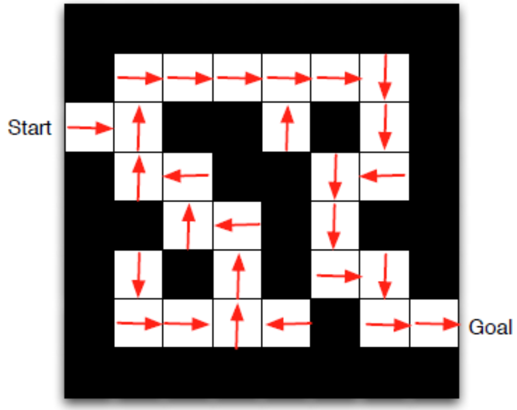
$$\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$



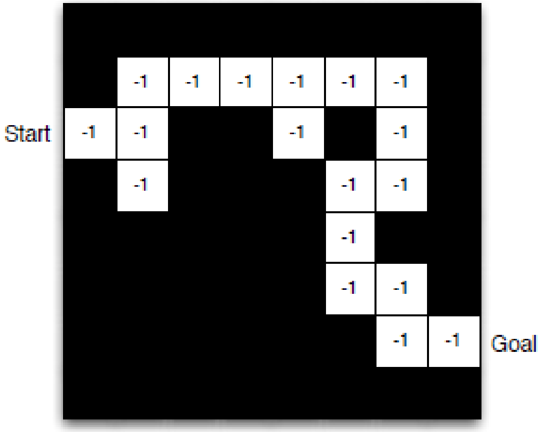
Maze Example



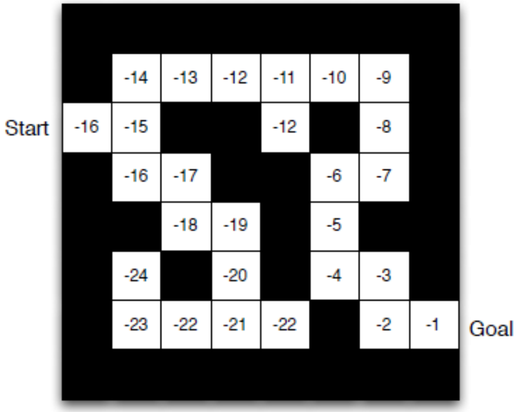
Rewards: -1 per time-step
Actions: N, E, S, W
States: Agent's location



Policy



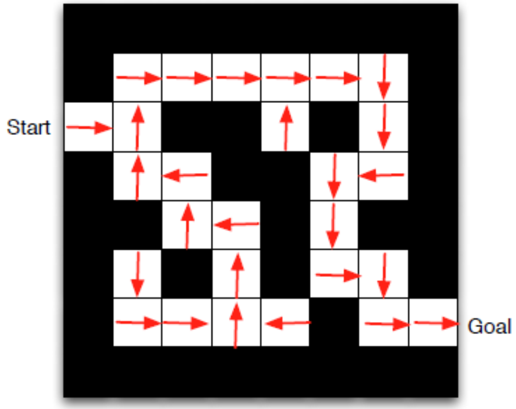
Model



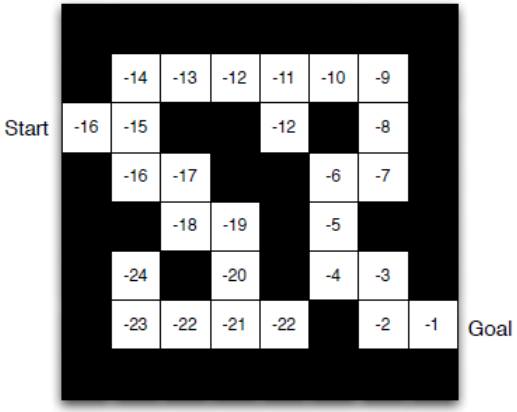
Value function

Categorizing Reinforcement Learning Agents

- Agents Action:
  - ❑ Value Based: Value function, no policy
  - ❑ Policy Based: Policy, no value function
  - ❑ Actor Critic: Both Policy and Value Function
- Modelling environment
  - ❑ Model Free: interacting directly environments
  - ❑ Model Based: Learn and model environments



Policy



	-14	-13	-12	-11	-10	-9	
-16	-15			-12		-8	
	-16	-17			-6	-7	
		-18	-19		-5		
	-24		-20		-4	-3	
	-23	-22	-21	-22		-2	-1

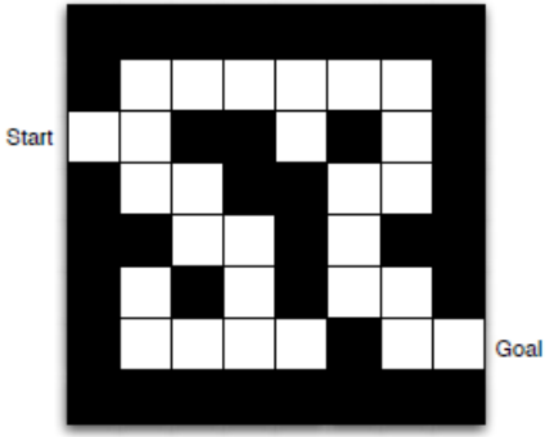
Value function

Vietnam Institute for Advanced Study in Mathematics

Learning and Planning

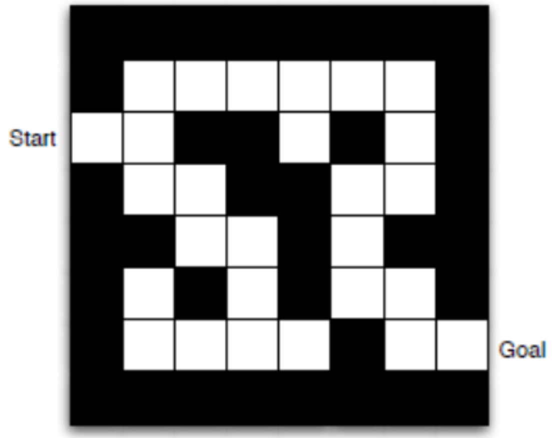
Sequence Decision Making

- ❑ Reinforcement Learning
  - Environments is initially unknown
  - Agent interacts with the environment
  - Agent improves policies
- ❑ Planning
  - Models of environment are known
  - Action by functional computation
  - Agent improve policies



Exploration and Exploitation

- Solve problem in trial-error learning
- Agents must learn to have good policies
- Agents learn from acting with their environments
- Reward may not response each step, it may be at the end of games
- Exploration: discovering the environment
- Exploitation: planning with maximal reward
- Trading between exploration and exploitation



Recap on RL introduction

- Sequence of decision, reward
- State, fully observation, partially observation
- Main components: Policy, Value Function, Model
- Categorizing RL agents
- Learning and Planning

MARKOV DECISION PROCESS

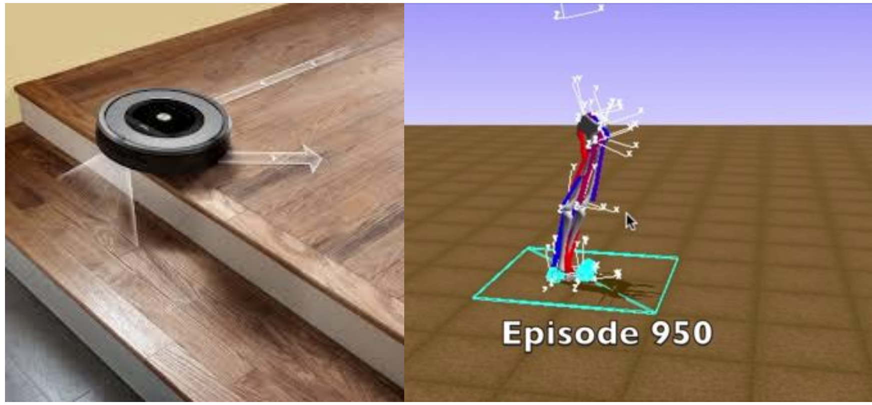
Markov decision process: Model of finite-state environment

Bellman Equation

Dynamic Programming

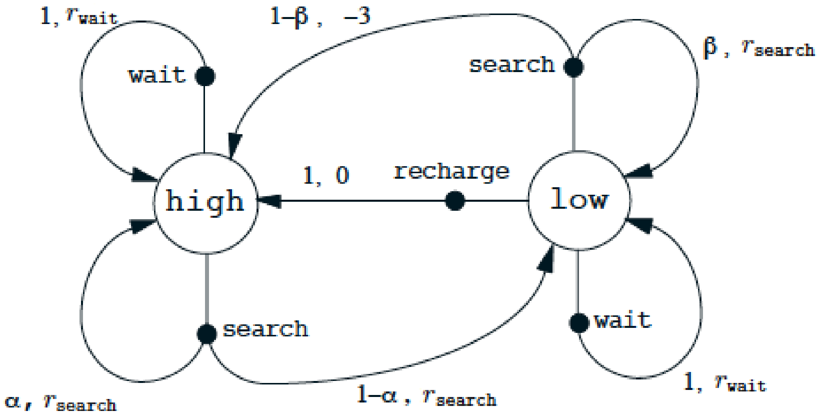
Markov Decision Process (Model of the environment)

- Terminologies:



In a Markov Decision Process:

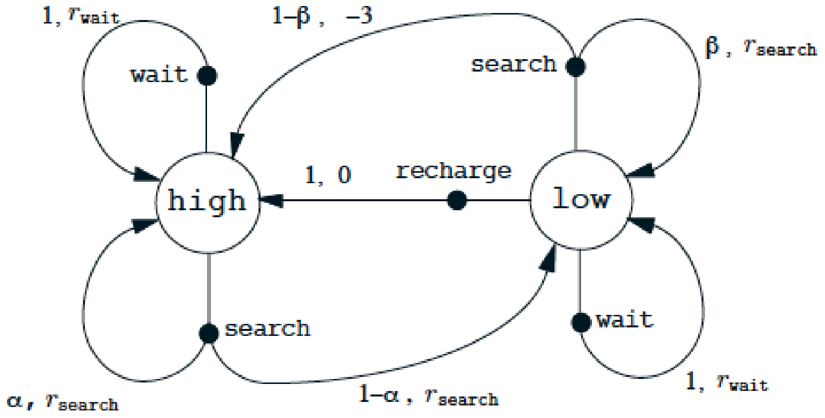
s, s'	states
a	action
r	reward
$\mathcal{S}$	set of all nonterminal states
$\mathcal{S}^+$	set of all states, including the terminal state
$\mathcal{A}(s)$	set of all actions possible in state s
$\mathcal{R}$	set of all possible rewards
t	discrete time step
$T, T(t)$	final time step of an episode, or of the episode including time t
A_t	action at time t
S_t	state at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
$p(s', r \| s, a)$	probability of transition to state s' with reward r, from state s and action a
$p(s' \| s, a)$	probability of transition to state s', from state s taking action a



Markov Decision Process

- Markov property: The distribution over future states **depends only on the present state and action**, not on any other previous event.

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



- Maximize return
  - Episodic task: consider return over finite horizon (e.g. games, maze).

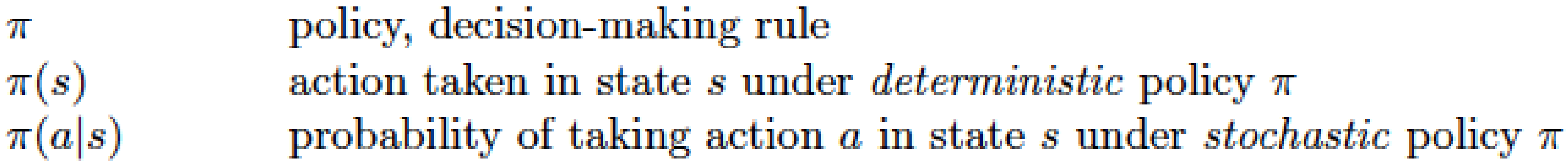
$$U_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_T$$

  - Continuing task: consider return over infinite horizon (e.g. juggling, balancing).

$$U_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} \ldots = \sum_{k=0:\infty} \gamma^k r_{t+k}$$

How we get good decision?

- Defining behavior: the policy
  - ❑ Policy: defines the action-selection strategy at every state



π	policy, decision-making rule
$\pi(s)$	action taken in state s under *deterministic* policy π
$\pi(a\|s)$	probability of taking action a in state s under *stochastic* policy π

- Goals: finds the policy that maximizes expected total reward

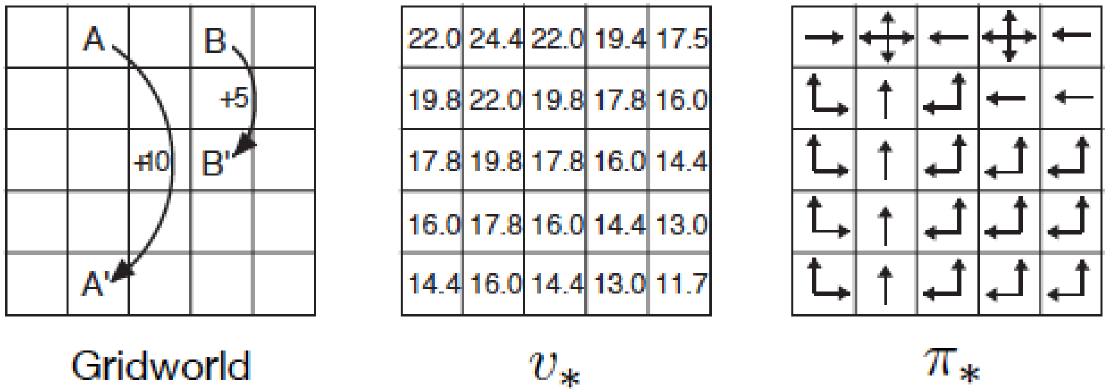
$$\textit{argmax}_{\pi}\, E_{\pi}\,[\, r_0 + r_1 + \ldots + r_T \mid s_0 \,]$$

Value functions

- The expected return of a policy for a state is call value function

$$V^{\pi}(s) = E_{\pi}\,[r_t + r_{t+t} + \ldots + r_T \mid s_t = s\,]$$

- Strategy to find optimal policy
  - Enumerate the space of all policies
  - Estimate the expected return of each one
  - Keep the policy that has maximum expected return



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

Gridworld v_* π_*

Gridworld example
- Reward to Off grid: -1
- Reward to On grid: 0
- Reward exception at A, B

Value functions

- Value of a policy

$$V^{\pi}(s) = E_{\pi}\,[r_t + r_{t+1} + \ldots + r_T \mid s_t = s\,]$$

$$V^{\pi}(s) = E_{\pi}\,[r_t\,] + E_{\pi}\,[\,r_{t+1} + \ldots + r_T \mid s_t = s\,]$$

$$V^{\pi}(s) = \underbrace{\textstyle\sum_{a \in A} \pi(s,a) R(s,a)}_{\textit{Immediate reward}} + \underbrace{E_{\pi}\,[\,r_{t+1} + \ldots + r_T \mid s_t = s\,]}_{\textit{Future expected sum of rewards}}$$

$$V^{\pi}(s) = \textstyle\sum_{a \in A} \pi(s,a) R(s,a) + \underbrace{\textstyle\sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s')}_{\textit{Expectation over 1-step transition}} E_{\pi}\,[r_{t+1} + \ldots + r_T \mid s_{t+1} = s'\,]$$

Note: $T(s,a,s') = p(s'|s,a)$

$$V^{\pi}(s) = \textstyle\sum_{a \in A} \pi(s,a) R(s,a) + \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s')\, \underbrace{V^{\pi}(s')}_{\textit{By definition}}$$

Bellman’s equation

- State value function (for a fixed policy with discount)

$$V^{\pi}(s) = \textstyle\sum_{a\in A} \pi(s,a) \, [\, \underbrace{R(s,a)}_{\textit{Immediate}} + \gamma \underbrace{\textstyle\sum_{s'\in S} T(s,a,s') V^{\pi}(s') \,]}_{\textit{Future expected sum of rewards}}$$

- State-action value function (Q-function)

$$Q^{\pi}(s,a) = R(s,a) + \gamma \textstyle\sum_{s'} T(s,a,s')[\textstyle\sum_{a'\in A} \pi(s',a') Q^{\pi}(s',a')]$$

- When S is a finite set of states, this is a system of linear equations (one per state)
- Belman’s equation in matrix form: $V^{\pi} = R^{\pi} + \gamma \, T^{\pi} V^{\pi}$

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'\in S} p(s'|a,s) V^{\pi}(s')$$

Optimal Value, Q and policy

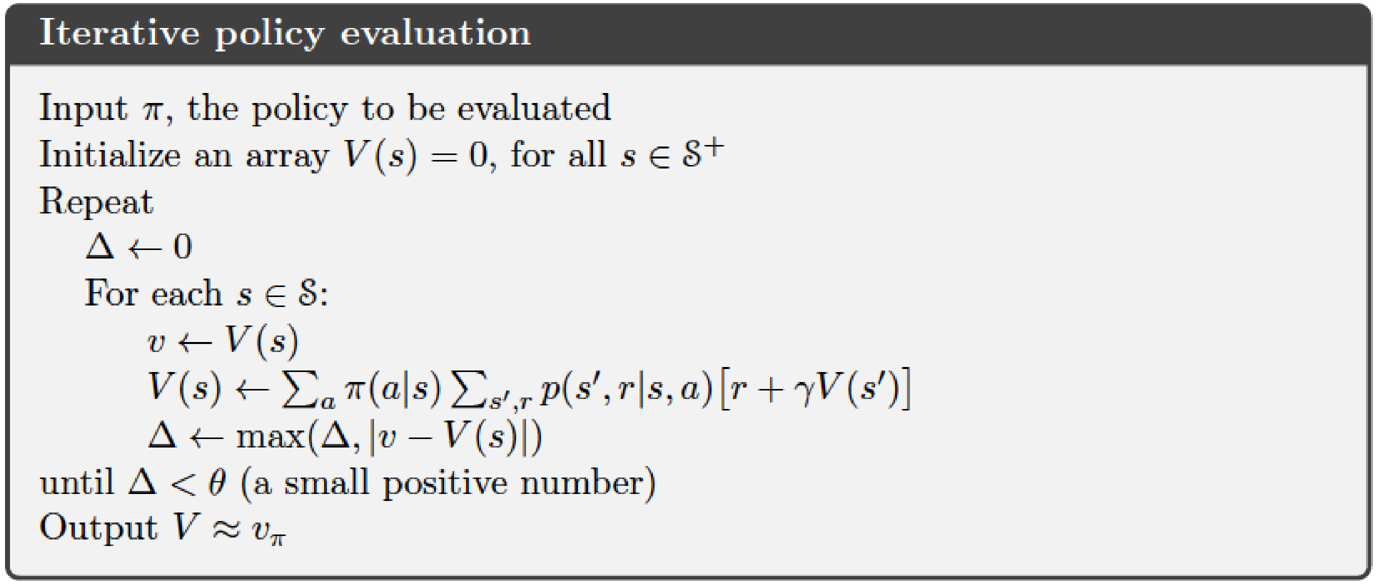
- Optimal V: the highest possible value for each s under any possible policy
- Satisfies the bellman Equation $V^*(s) = \max_a \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V^*(s') \right]$
- Optimal Q-function $Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V^*(s')$
- Optimal policy: $\pi^*(s,a) = \arg\max_a Q^*(s,a)$

Dynamic Programming (DP)

- Assuming full knowledge of Markov Decision Process
- It is used for planning in an MDP
- For prediction
  - ❑ Input: MDP (S,A,P,R,γ) and policy π
  - ❑ Output: value function v_π
- For controlling
  - ❑ Input: MDP (S,A,P,R,γ) and policy π
  - ❑ Output: Optimal value function v_* and optimal policy π_*

DP: Iterative Policy Evaluation

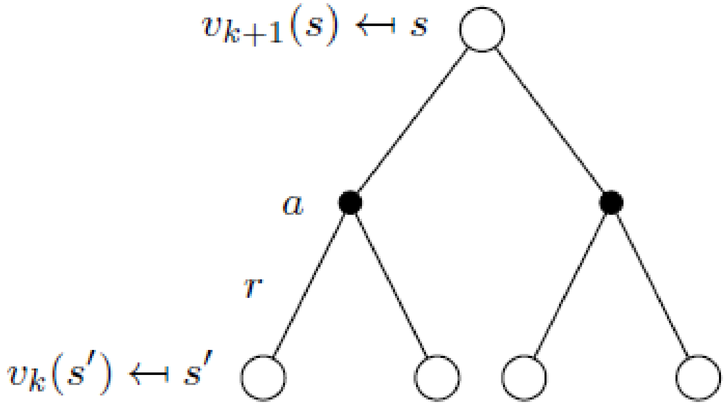
- Main idea of Dynamic Programming: turn Bellman equations to update rules
- Problem: evaluate a given policy π
- Iterative policy evaluation: Fix policy



```
Iterative policy evaluation

Input $\pi$, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$
```

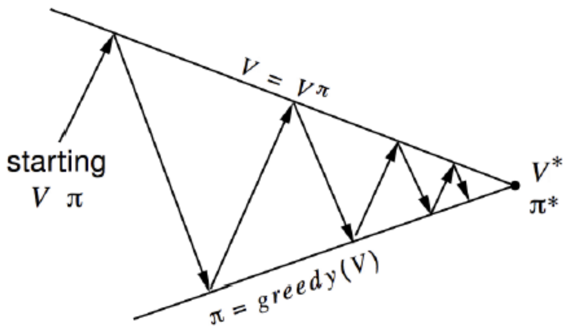
Bellman eq: $V^\pi = R^\pi + \gamma P^\pi V^\pi$



$$v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_k(s')\right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \mathbf{v}^k$$

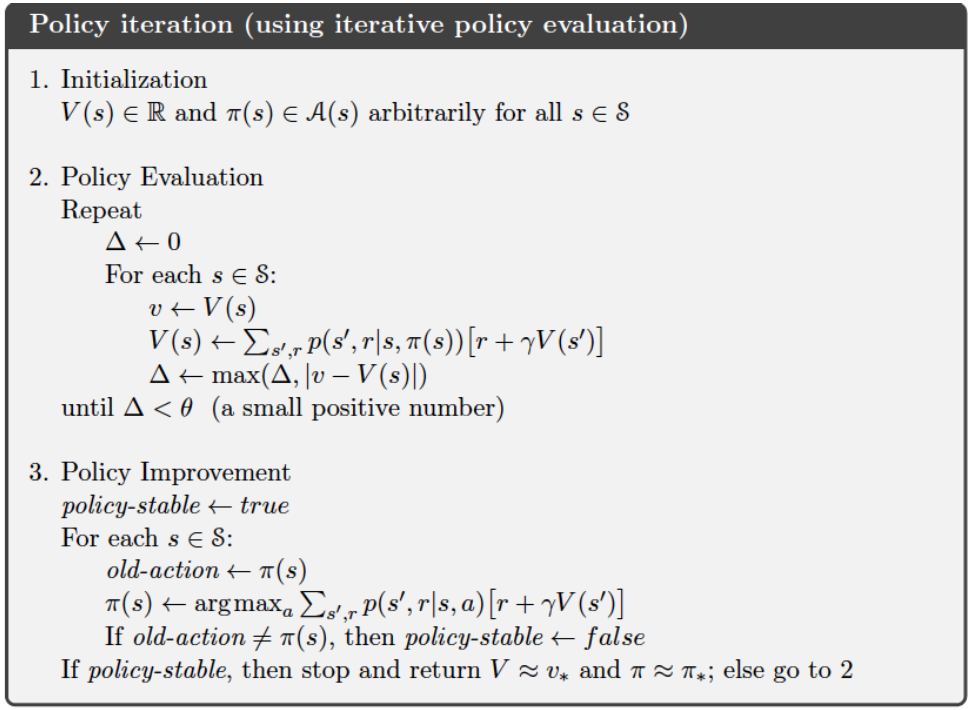
DP: Improving a Policy



- Finding a good policy: Policy

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

- Start with an initial policy π_0 (e.g. random)
- Repeat:
  - Compute V^π, using iterative policy evaluation.
  - Compute a new policy π' that is greedy with respect to V^π
- Terminate when $\pi = \pi'$



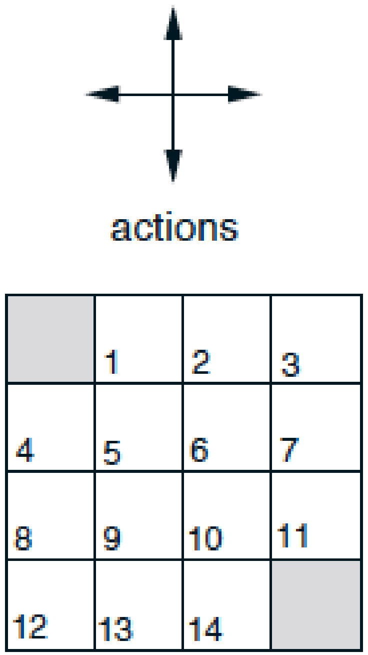
Policy iteration (using iterative policy evaluation)

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
       $\Delta \leftarrow 0$
       For each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

Gridworld example



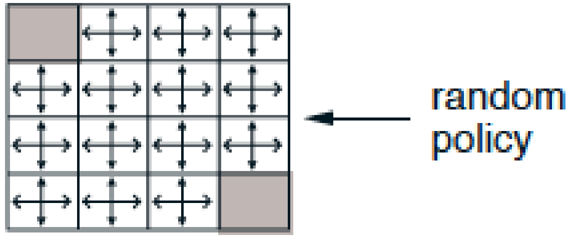
$R = -1$
on all transitions

v_k for the Random Policy

Greedy Policy w.r.t. v_k

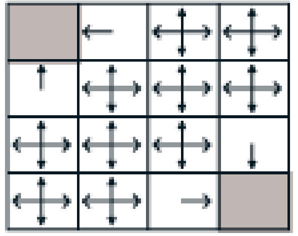
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



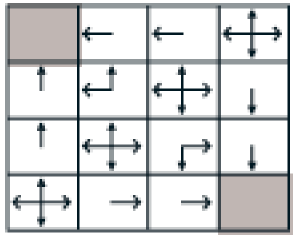
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

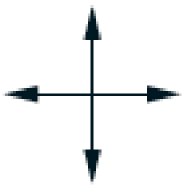


$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Gridworld example



actions

▨	1	2	3
4	5	6	7
8	9	10	11
12	13	14	▨

$R = -1$
on all transitions

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

▨	←	←	←↓
↑	←↑	←↓	↓
↑	↑→	→↓	↓
↑→	→	→	▨

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

▨	←	←	←↓
↑	←↑	←↓	↓
↑	↑→	→↓	↓
↑→	→	→	▨

$k = \infty$

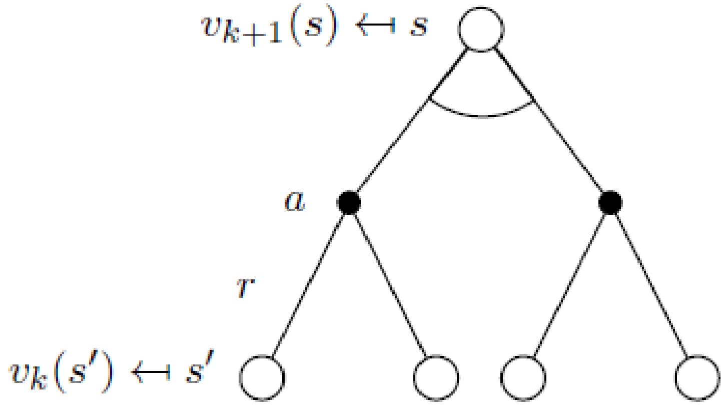
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

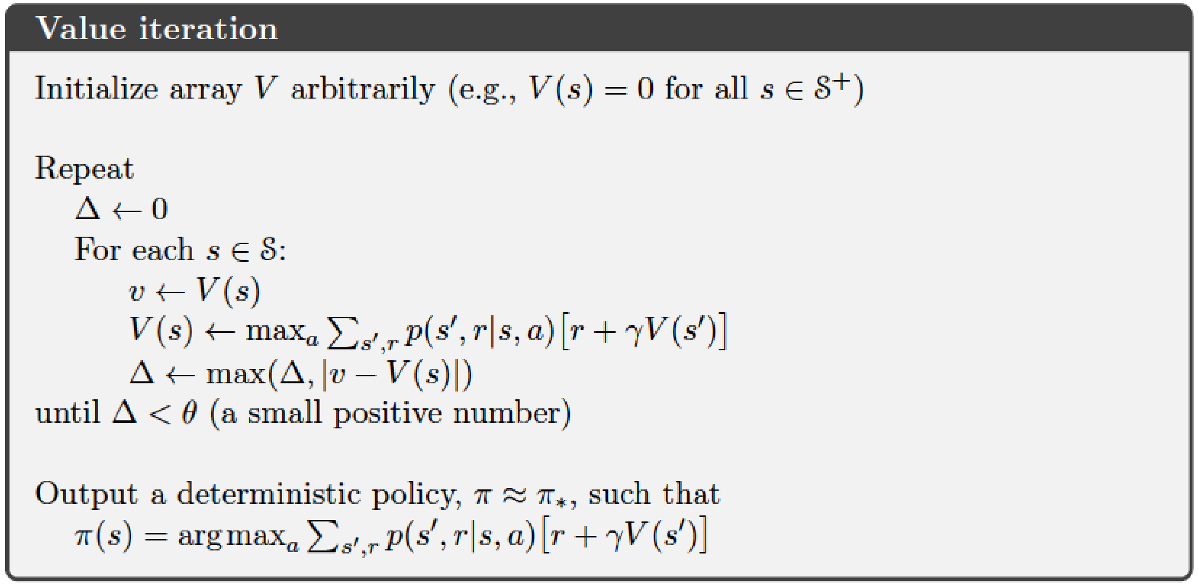
▨	←	←	←↓
↑	←↑	←↓	↓
↑	↑→	→↓	↓
↑→	→	→	▨

optimal policy

DP:Value Iteration

- Finding a good policy: Value iteration
  - ❑ Drawback of policy iteration: evaluate policy also needs iteration
  - ❑ Main idea: Turn the Bellman optimality equation into an iterative update rule (same policy evaluation)





```
Value iteration

Initialize array $V$ arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\mathbf{a}} + \gamma \boldsymbol{\mathcal{P}}^{\mathbf{a}} \mathbf{v}_k$$

DP: Pros and Cons

- Rarely use Dynamic programming in real applications
  - ❑ To calculate we must access environment model, fully observe with knowledge of environment.
  - ❑ Extending to continues actions and state
- However:
- Mathematically exact, expressible and analyzable
  - ❑ Good deals for small problem.
  - ❑ Stable, simple and fast

Visualization and Codes

- https://cs.stanford.edu/people/karpathy/reinforcejs/index.html

Recap on Reinforcement Learning

- Introduction on RL
  - Intelligent agents learning and acting
  - Sequence of decision, reward
- Markov Decision Process
  - Model of finite-state environment
  - Bellman Equation
  - Dynamic Programming
- Next:
  - Online Learning

Questions?

THANK YOU!

VIASM
Vietnam Institute for
Advanced Study in Mathematics