Reinforcement Learning

Bootcamp

Machine Perception

7 May 2020

Advanced Interactive Technologies

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Overview

- Motivation
- Dynamic Programming and Planning
- Temporal Difference Learning

Overview Learning Methods

Supervised Learning

- Learn mapping between input and output
- Labelled ground-truth data guides the learning process

Unsupervised Learning

- Learn a structure in a dataset
- No ground-truth data available

[SR-GAN. Ledig et al. CVPR ‘17]

Reinforcement Learning

- Learn how to act
- Data revealed through interaction with an environment

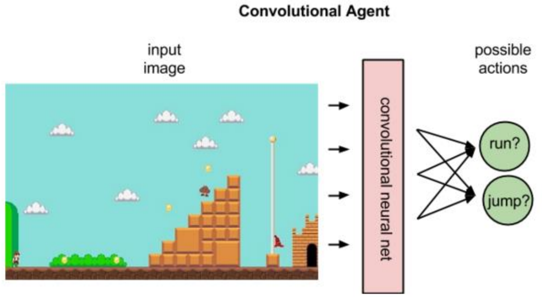
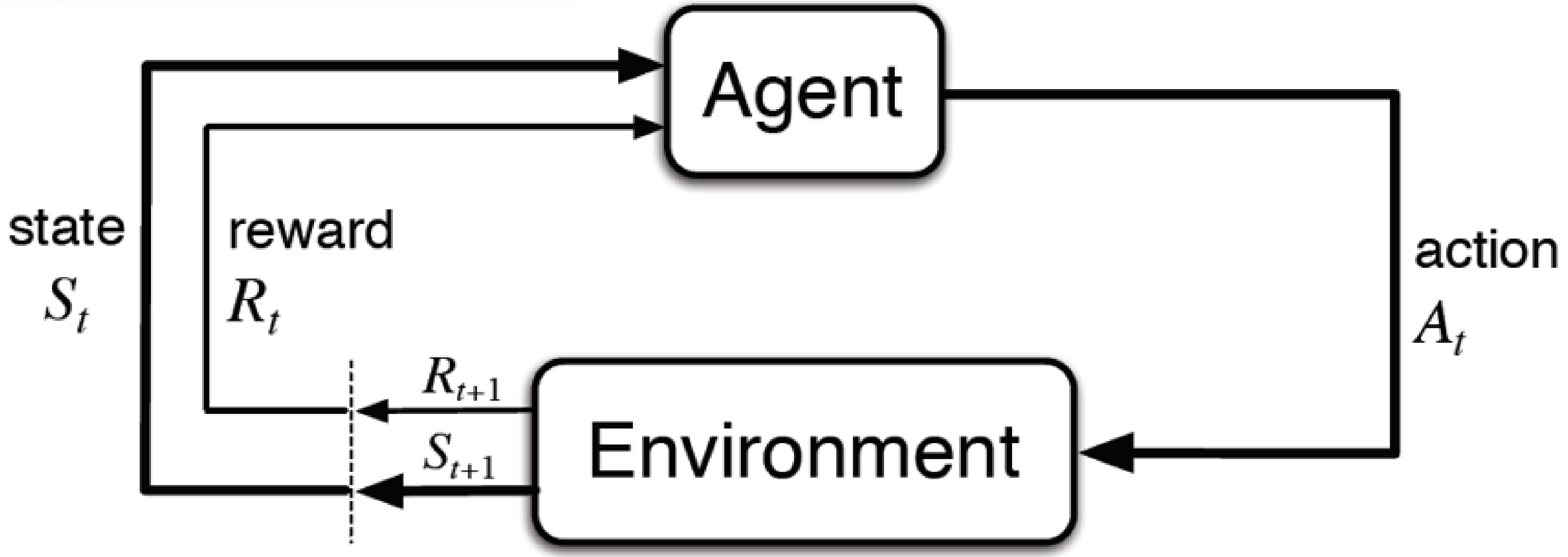


Image source: pathmind.com

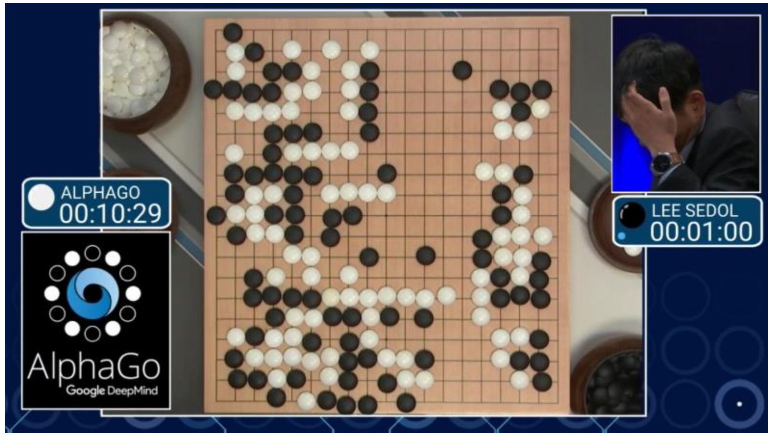
Reinforcement Learning Problem Statement

Reinforcement Learning is a problem, not a method:

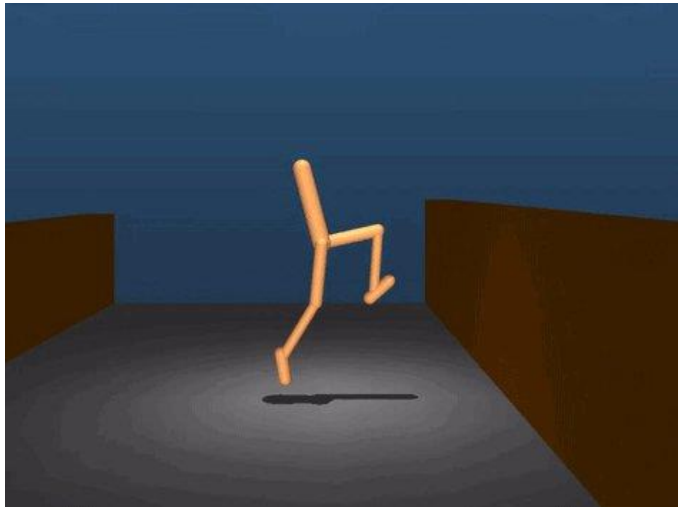
- Learning how to map states to actions
- Maximize a numerical reward signal
- Unknown and uncertain environment



Motivation

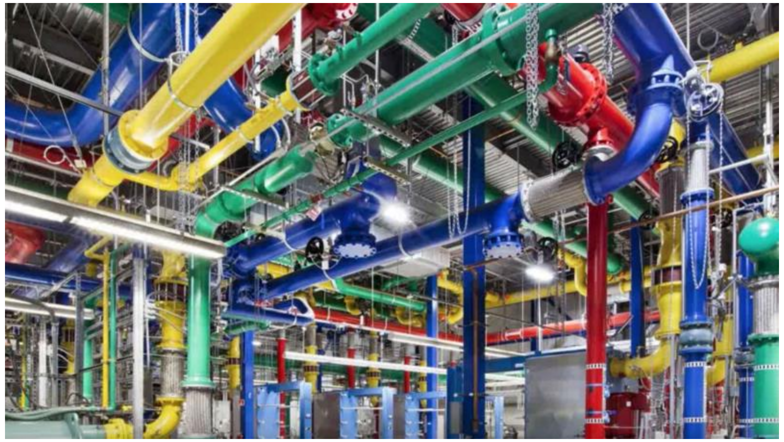


Games



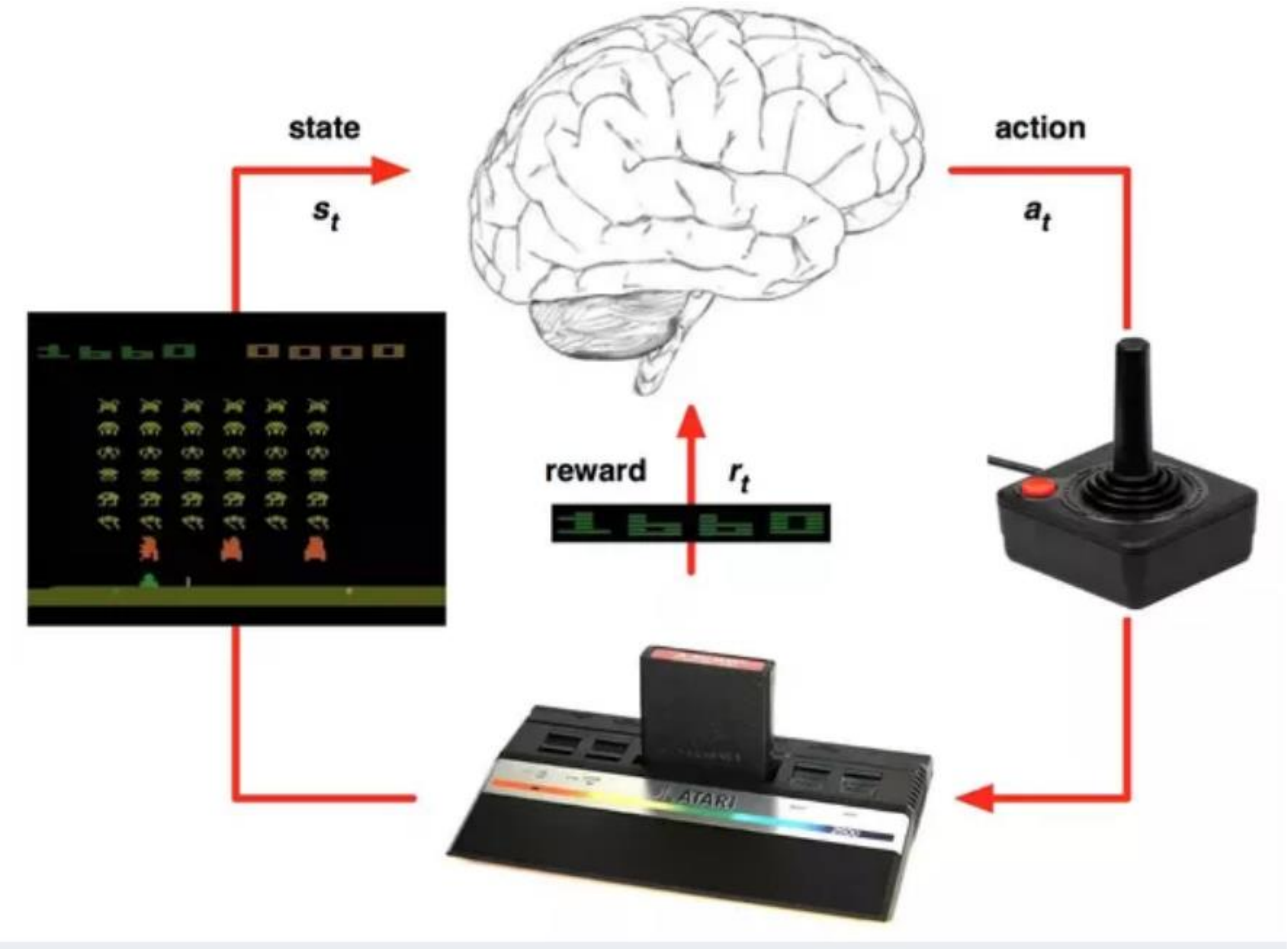
Robot Control

Attention based models



Logistics and Operations

Example: Atari



[Image source: RLSS summer school – Olivier Pietquin]

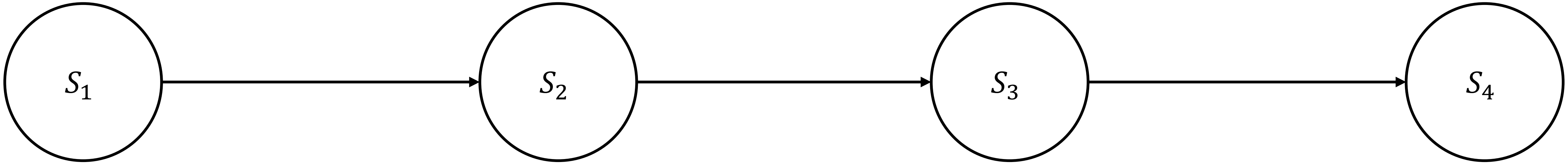
Objective: Achieve a high score

State: Images of the game state

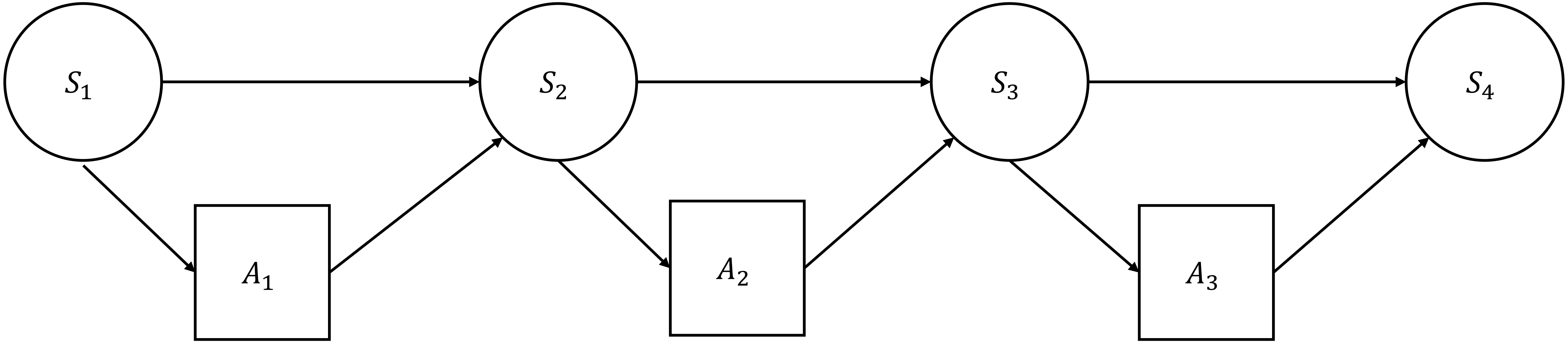
Actions: Move the joystick

Reward: Score of the game

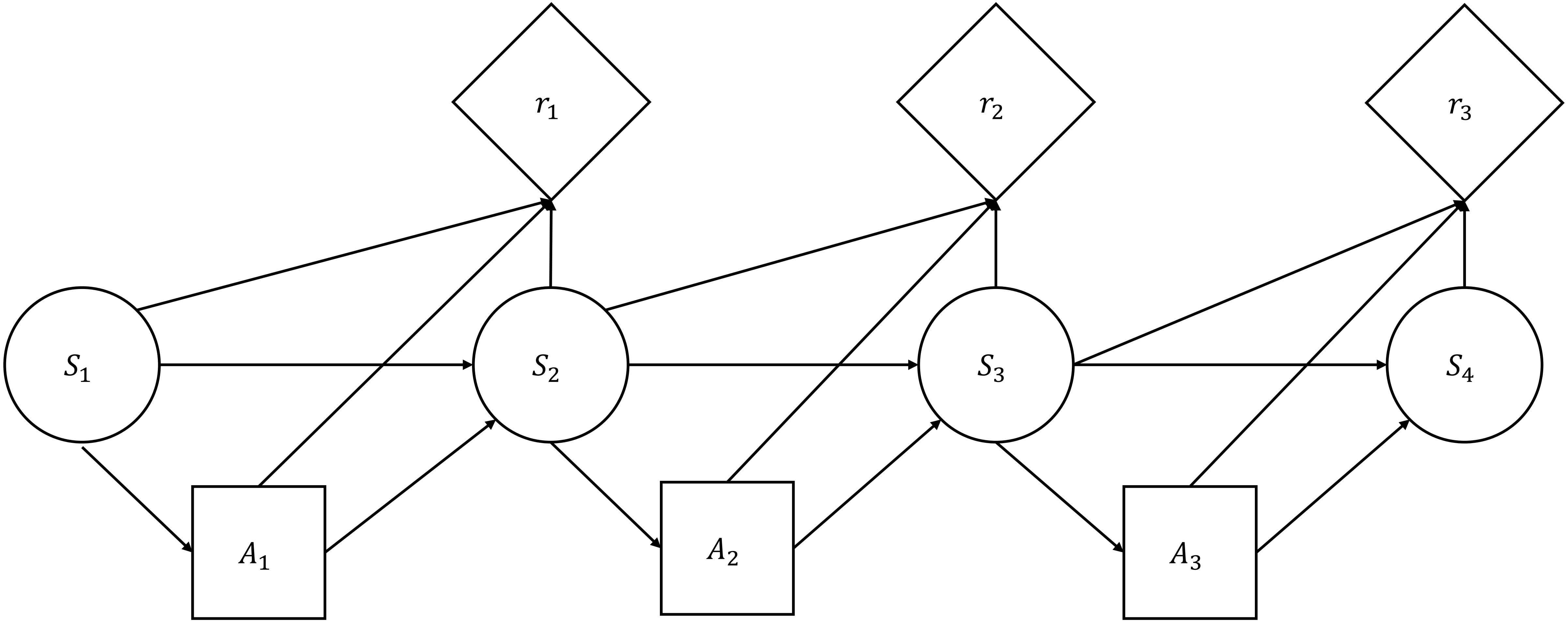
Recap Markov Decision Processes



Recap Markov Decision Processes



Recap Markov Decision Processes



Recap Markov Decision Processes

We model the environment as a Markov Decision Process

- Assumption: Transitions, initial state, rewards and policies are deterministic → Simpler notation.

An MDP consists of $(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \mathbb{P}, \boldsymbol{\gamma})$

- a set of states $\mathcal{S}$
- a set of actions $\mathcal{A}$
- a reward function $r: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (deterministic, can also be a distribution)
- a transition function $p: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$(deterministic, can also be a distribution)
- initial state $s_0 \in \mathcal{S}$
- Discount factor: γ

Problem

The core problem in solving MDPs is to find a "policy" $\pi(s)$ for the decision maker

- Once we know the optimal policy, for each state of the decision process we know how to act
- The chosen action is fully determined by $\pi(s)$

The optimal policy will maximize some cumulative quantity over the random rewards

Value Function

Important concept: Value function $V_\pi : S \to \mathbb{R}$ of a policy π.

V_π expresses the expected cumulative reward (return) for each state that we achieve when following the policy π

Bellman equation for v_π

$G_{t+1} = \sum_{k=0}^{\infty} \gamma^k R_{(t+1)+k+1}$

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

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

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

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

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

$$= E_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = S\right]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma E_\pi[G_{t+1} | S_{t+1} = S']]$$

Bellman equation for v_π

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

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

$$= E_\pi\,[R_{t+1} + \gamma G_{t+1} | S_t = S\,]$$

$$= \sum_a \boxed{\pi(a|s)} \sum_{s\prime} \sum_r p(s', r | s, a)[r + \gamma E_\pi[G_{t+1} | S_{t+1} = S']]$$

Probability of taking an action given a state

Bellman equation for v_π

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

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

$$= E_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = S \right]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r \boxed{p(s', r | s, a)} [r + \gamma E_\pi[G_{t+1} | S_{t+1} = S']]$$

Joint probability of next state and the reward given the current state and action.

Transition Matrix

Bellman equation for v_π

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

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

$$= E_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = S \right]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \boxed{[r + \gamma E_\pi[G_{t+1} | S_{t+1} = S']]}$$

The expected reward given a new state

Bellman equation for v_π

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

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

$$= E_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = S\right]$$

$$= \sum_a \pi(a|s) \boxed{\sum_{s'} \sum_r p(s', r | s, a)[r + \gamma E_\pi[G_{t+1} | S_{t+1} = S']]}$$

A weighted sum of rewards given the current state and action

Bellman equation for v_π

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

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

$$= E_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = S\right]$$

$$= \boxed{\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma E_\pi[G_{t+1} | S_{t+1} = S']]}$$

A weighted sum of rewards given the current state

Bellman equation for v_π

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

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

$$= \; E_\pi \; [R_{t+1} + \gamma G_{t+1} | S_t = S \,]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma E_\pi[G_{t+1} | S_{t+1} = S']]$$

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

Next Steps

How can we solve the Bellman Equation?

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

- **Dynamic Programming**
  - Requires MDP with given environment model (transition probabilities)
  - Value function based on estimates of subsequent values

- **Monte-Carlo Methods** (not covered here)
  - No need to known environment model
  - Value function based on gathered data and only updated along full trajectories

- **Temporal-Difference Learning**
  - No need to known environment model
  - Combination of DP and MC principles
  - Value function updated based on estimates of subsequent values along trajectory

Dynamic Programming

Overview

- Collection of algorithms that can compute optimal policies given a perfect model of the world (MDP).
- Can be applied to continuous space, we focus on discrete.
- Limited utility, great theoretical importance.
  - All RL methods have a similar goal, with less computation and no perfect model assumption.

Overview – 3 Key ideas

Value Iteration for estimating $\boldsymbol{\pi}_*$

1) Compute optimal state value function v_*
2) Obtain policy π_* based on v_*

先 initialize V_0
iterate V until converge
then we got π^*

Iterative Policy Evaluation

1) Compute the state value function v_π for a given policy π

Policy Iteration for estimating $\boldsymbol{\pi}_*$

1) Compute the state value function v_π for any arbitrary policy π
2) Update policy π given $v_\pi \;\rightarrow\; \pi'$
3) Iterate till $\pi \approx \pi'$

Overview – 3 Key ideas

Value Iteration for estimating $\boldsymbol{\pi}_*$

1) Compute optimal state value function v_*
2) Obtain policy π_* based on v_*

Iterative Policy Evaluation

1) Compute the state value function v_π for a given policy π

Policy Iteration for estimating $\boldsymbol{\pi}_*$

1) Compute the state value function v_π for any arbitrary policy π
2) Update policy π given $v_\pi \rightarrow \pi'$
3) Iterate till $\pi \approx \pi'$

Example Environment

0	**0**	**0**	**Goal** **100**
0	**0**	**Trap** **-100**	**Trap** **-100**
Start	**0**	**0**	**0**

Goal: Find optimal path to goal

States: Grid fields
Start state blue field
Terminal states trap/goal

Actions: Move left/right/up/down

Reward: 100 upon reaching the goal
-100 landing in the trap
-5 for each step taken

Other info: $\gamma = 1$
Initial states initialized to 0
Deterministic transitions

Value Function

- Given a value function V_π we can derive a greedy policy π':

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}(r(s,a) + \gamma V_\pi(p(s,a)))$$

- One can show that $V_{\pi'}(s) \geq V_\pi(s)$

Value Function

- For the optimal policy π^*: The greedy policy of V_{π^*} is π^* itself.

- V_{π^*} fulfills the Bellman Optimality Equation:

$$V^*(s) = \max_{a \in A}\{r(s, a) + \gamma V^*\big(p(s, a)\big)\}$$

Example of Value Iteration ($\gamma = 1$, step cost=5)

Initial value is 0 for every state

0	**0**	**0**	**Goal 100**
0	**0**	**Trap -100**	**Trap -100**
Start 0	**0**	**0**	**0**

Example of Value Iteration ($\gamma = 1$, step cost=5)

Iteration 1

$= -5 + 100$

-5	-5	95	**Finish** **100**
-5	-5	**Trap** **-100**	**Trap** **-100**
Start **-5**	-5	-5	-5

Example of Value Iteration ($\gamma = 1$, step cost=5)

Iteration 2

-10	90	95	**Goal** **100**
-10	-10	**Trap** **-100**	**Trap** **-100**
Start **-10**	-10	-10	-10

Example of Value Iteration ($\gamma = 1$, step cost=5)

Iteration 3

85	90	95	**Goal** 100
-15	85	**Trap** -100	**Trap** -100
Start -15	-15	-15	-15

Example of Value Iteration ($\gamma = 1$, step cost=5)

Iteration 4

85	**90**	**95**	**Goal** **100**
80	**85**	**Trap** **-100**	**Trap** **-100**
Start **-20**	**80**	**-20**	**-20**

Example of Value Iteration ($\gamma = 1$, step cost=5)

Iteration 4

85	90	95	**Goal** 100
80	85	**Trap** -100	**Trap** -100
Start 75	80	75	-25

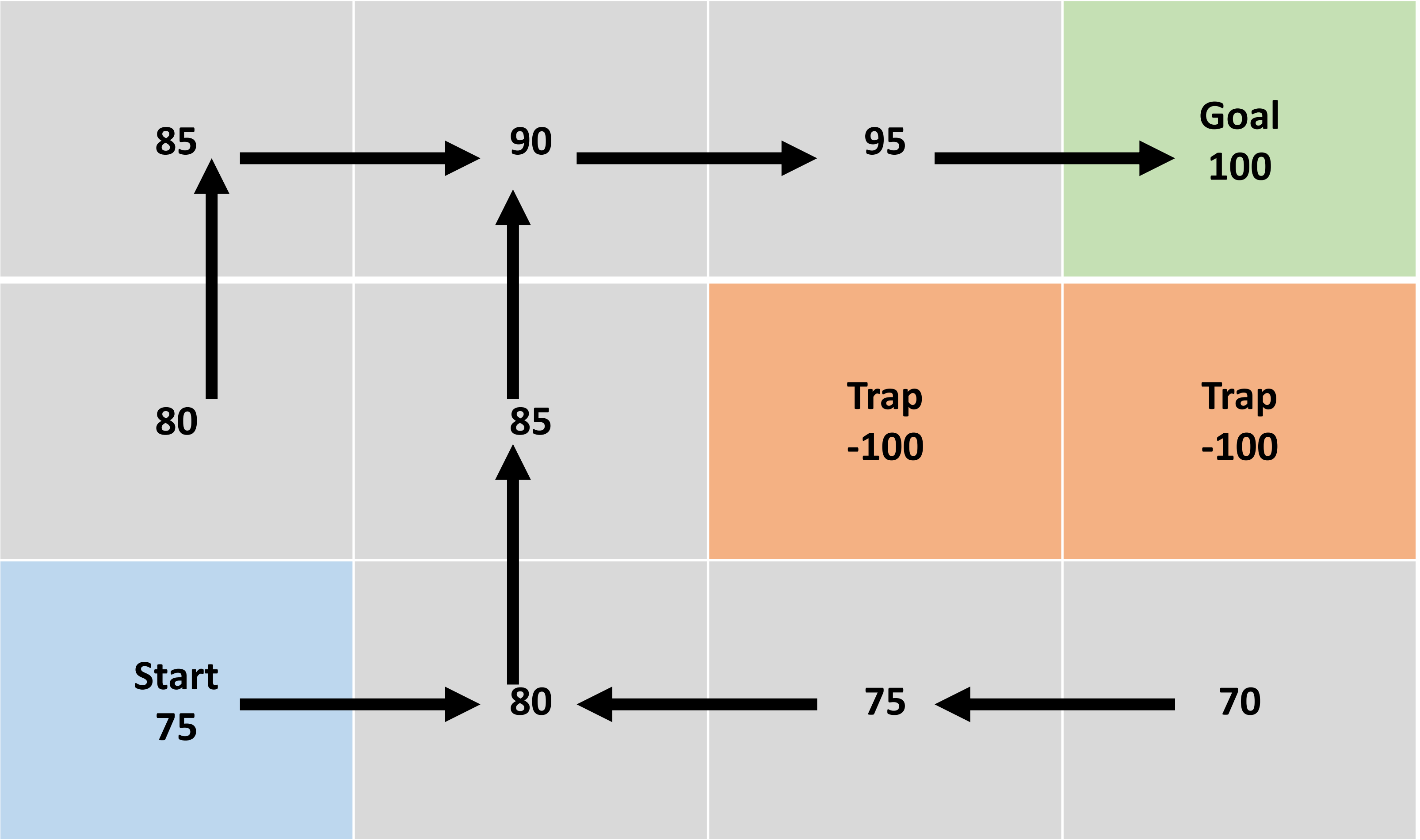
Example of Value Iteration ($\gamma = 1$, step cost=5)

Iteration 5. Done as there is no change anymore.

85	90	95	Goal 100
80	85	Trap -100	Trap -100
Start 75	80	75	70

Example of Value Iteration ($\gamma = 1$, step cost=5)

Compute greedy policy



Value Iteration: Step-wise

1. $V_0^*(s) =$ optimal value for state s when H=0
   - $V_0^*(s) = 0 \ \forall s$
2. $V_1^*(s) =$ optimal value for state s when H=1
   - $V_1^*(s) = \max_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma V_0^*(s')]$
3. $V_2^*(s) =$ optimal value for state s when H=2
   - $V_2^*(s) = \max_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma V_1^*(s')]$
4. $V_k^*(s) =$ optimal value for state s when H=k
   - $V_k^*(s) = \max_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma V_{k-1}^*(s')]$

Value Iteration: Algorithm

Instead of continuously doing policy evaluation and policy improvement, we can do the improvement step in each evaluation iteration.

$$
\begin{aligned}
& i \leftarrow 0 \\
& V_0 \leftarrow 0 \\
& while \; \triangle > \theta \; \{ \\
& \quad \triangle \leftarrow 0 \\
& \quad for \; all \; s \in S \{ \\
& \qquad v \leftarrow V(s) \\
& \qquad V(s) \leftarrow \max_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] \\
& \qquad \triangle \leftarrow \max(\triangle, |v - V(s)|) \\
& \quad \} \\
& \} \\
& \pi_i \leftarrow \text{greedy policy from } V_i
\end{aligned}
$$

always 1 or 0 in our case due to deterministic environment

Value Iteration vs. Policy Iteration Algorithm

Value Iteration:

1. Iterate until value function converges.
2. Obtain optimal policy through the greedy policy.

Policy Iteration:

1. Initialize arbitrary policy.

Repeat until policy does not improve:

2. Evaluate the value function given the current policy.
3. Improve the policy.

Pros and Cons of Dynamic Programming

Pros:

- Exact methods
- Policy/Value iteration are guaranteed to converge in finite number of iterations
- Value iteration is typically more efficient

Cons:

- Need to know the transition probability matrix
- Need to iterate over the whole state space (very expensive)
- Requires memory proportional to the size of the state space

Summary

This Lecture

Dynamic Programming

- Uses value of neighbouring states to update current state (given π)
- Visits every state
- Requires knowledge of the transition probabilities

Monte Carlo Methods (not covered in detail)

- Run in episodes, update state values at end of episode.
- Experience based (no need to know system dynamics)
- High Variance

Temporal Differences

- Run in episodes, update state values at each step.
- Combination of DP and MC ideas
- Exploration/Exploitation → Local minima

Next Lecture

- TD-Learning

- Function Approximation
  - Extend the ideas to arbitrary large state spaces (e.g. images).

- Policy based gradient methods
  - Replace action-value methods with parametrised policies.

- Deep Reinforcement Learning applications

- Research in Deep Reinforcement Learning

References

Book by Sutton:
http://incompleteideas.net/book/the-book-2nd.html

Lecture by David Silver:
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Lecture by John Schulman:
https://www.youtube.com/watch?v=aUrX-rP_ss4