Sequential Decision Making Problems

Changhoon Kevin, Jeong

Seoul National University

chjeong@bi.snu.ac.kr

References

- RL Course by David Silver, DeepMind
  - ✓ The presentation is referenced a lot in this material
- Artificial Intelligence Course by Prof. Zhang, Seoul National University
- Artificial Intelligence: A Modern Approach(3rd Edition), by S Russell, et al.
- Reinforcement Learning: An Introduction, by Richard S. Sutton, et al.
- CS 330: Deep Multi-Task and Meta Learning, Stanford University
- CS238: Decision Making under Uncertainty, Stanford University

Contents

I. Sequential Decision Problem

Sequential Decision Problem

- Model
  - A model predicts what the environment will do next
  - P predicts the next state : $P_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
  - R predicts the next (immediate) reward : $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$

- Two fundamental problems in sequential making
  - Reinforcement Learning
    - The environment is initially unknown
    - The agent interacts with the environment
    - The agent improves its policy
  - Planning
    - A model of the environment is known
    - The agent performs computations with its model(without any external interaction)
    - The agent improves its policy

Markov Processes

- Markov Property
  - The future is independent of the past given the present

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

  - State transition matrix P:

$$P_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$$

$$P = \begin{pmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{pmatrix}$$

- Markov Process (or Markov Chain) is a tuple (S, P)
  - S is a (finite) set of states
  - P is a state transition probability matrix, $P_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$

Markov Reward Processes

- A Markov Reward Process is a tuple (S, P, R, γ)
  - S is a finite set of states
  - P is a state transition probability matrix, $P_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$
  - R is a reward function, $R_s = \mathbb{E}[R_{t+1} | S_t = s]$
  - γ is a discount factor, $\gamma \in [0,1]$

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

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

  - Trade-off ?
    - γ close to 0 leads to "myopic" evaluation
    - γ close to 1 leads to "far-sighted" evaluation

Markov Reward Processes

- Why are most Markov reward and decision processes discounted?
  - Mathematically convenient to discount rewards
  - Avoids infinite returns in cyclic Markov processes
  - Uncertainty about the future may not be fully represented
  - If reward is financial, immediate rewards may earn more interest than delayed rewards
  - Animal/human behavior shows preference for immediate reward
  - It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$)

Markov Reward Processes

- Value function $v(s)$
  - The state value function $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

  - Bellman Equation for MRPs

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]
\end{aligned}
$$

Markov Reward Processes

- The Bellman equation can be expressed concisely using matrices

$$v = R + \gamma P v$$

$$\begin{pmatrix} v(1) \\ \vdots \\ v(n) \end{pmatrix} = \begin{pmatrix} R_1 \\ \vdots \\ R_n \end{pmatrix} + \gamma \begin{pmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{pmatrix} \begin{pmatrix} v(1) \\ \vdots \\ v(n) \end{pmatrix}$$

- The Bellman equation is a linear equation, and can be solved directly;

$$v = R + \gamma P v$$
$$(I - \gamma P) v = R$$
$$v = (I - \gamma P)^{-1} R$$

  - Computational complexity is $O(n^3)$ for n states
  - Direct solution only possible for small MRPs

Markov Decision Processes

- A Markov Decision Process is a tuple (S, A, P, R, γ)
  - S is a finite set of states
  - A is a finite set of actions
  - P is a state transition probability matrix : $P_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
  - R is a reward function : $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - γ is discount factor $\gamma \in [0,1]$

- Policies π is a distribution over actions given states

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

  - A policy fully defines the behavior of an agent
  - MDP policies depend on the current state(not history)

Markov Decision Processes

- Value function $v_\pi(s)$
  - The state value function $v_\pi(s)$ of an MDP is the expected return starting from state s, following policy π

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

- Q-function $q_\pi(s, a)$
  - The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

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

Markov Decision Processes

- Bellman Expectation Equation
  - State-value function

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

  - Action-value function

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

- Relationship between and v_π and q_π

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

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s\prime \in S} P_{ss'}^a v_\pi(s')$$

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s\prime \in S} P_{ss'}^a v_\pi(s') \right)$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s\prime \in S} P_{ss'}^a \sum_{a \in A} \pi(a'|s') q_\pi(s',a')$$

Markov Decision Processes

- Bellman Expectation Equation(Matrix Form)
  - The Bellman expectation equation can be expressed concisely using the induced MRP,

$$v_\pi = R^\pi + \gamma P^\pi v_\pi$$

  - With direct solution

$$v_\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

  - Computational complexity is $O(n^3)$ for n states
  - Direct solution only possible for small MDPs

Markov Decision Processes

- Optimal Value Function $v_*(s)$
  - The optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

  - The optimal action-value function $q_*(s,a)$ is the maximum action-value function over all policies

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

- Optimal Policies

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in A}{\operatorname{argmax}}\, q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

  - There is always a deterministic optimal policy for any MDP
  - If we know $q_*(s,a)$, we immediately have the optimal policy

Markov Decision Processes

- Bellman Optimality Equation
  - Optimal state-value function $v_*(s)$

$$v_*(s) = \max_a q_*(s,a)$$

$$v_*(s) = \max_a \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right)$$

  - Optimal action-value function $q_*(s,a)$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a')$$

Markov Decision Processes

- Bellman Optimality Equation
  - Optimal state-value function $v_*(s)$

$$v_*(s) = \max_a q_*(s,a)$$

$$v_*(s) = \max_a \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right)$$

non-linear
In general, no closed form solution
Solutions;

- Policy iteration(Planning)
- Value iteration(Planning)
- Sarsa(Learning)
- Q-learning(Learning)

  - Optimal action-value function $q_*(s,a)$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a')$$

II. Dynamic Programming

Dynamic Programming

✓Dynamic : sequential or temporal component to the problem

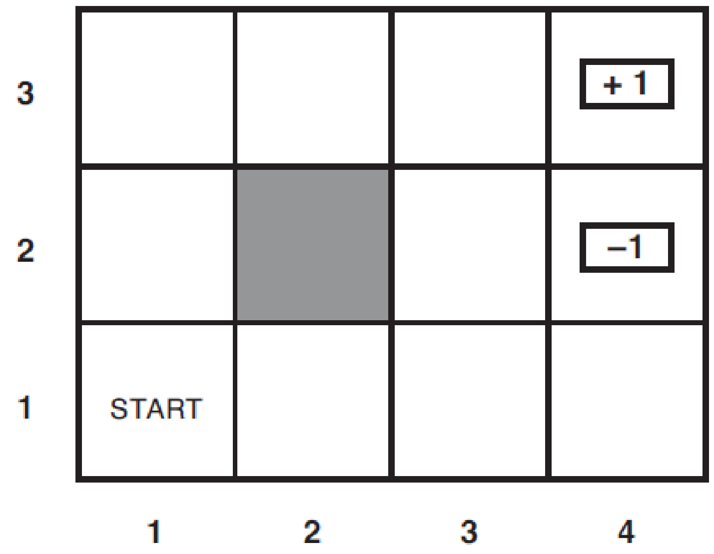
✓Programming : optimizing a "program", i.e. a policy

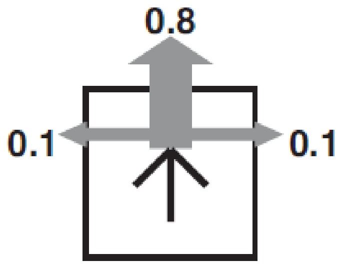
- A method for solving complex problems
- By breaking them down into sub-problems
  - Solve the sub-problems
  - Combine solutions to sub-problems
- DP assumes full knowledge of the MDP

- Prediction : MDP(S, A, P, R, γ) and policy π are given as inputs, and value function v_π is calculated as output
- Control : MDP(S, A, P, R, γ) is given as input, and optimal value function v_* and optimal policy π_* are calculated as outputs

4×3 Grid World Environment

- A simple example of sequence decision problem
- Markov Decision Processes(S, A, P, R, γ)
  - S: $(1,1), (1,2), \ldots, (4,2), (4,3)$ except $(2,2)$
  - A: *Up*, *Down*, *Left*, *Right*
  - P: “Intended” outcome with probability 0.8
  - R: $S(4,2) = -1, \;\; S(4,3) = 1, \;$ otherwise -0.04
  - $\gamma \in [0,1]$



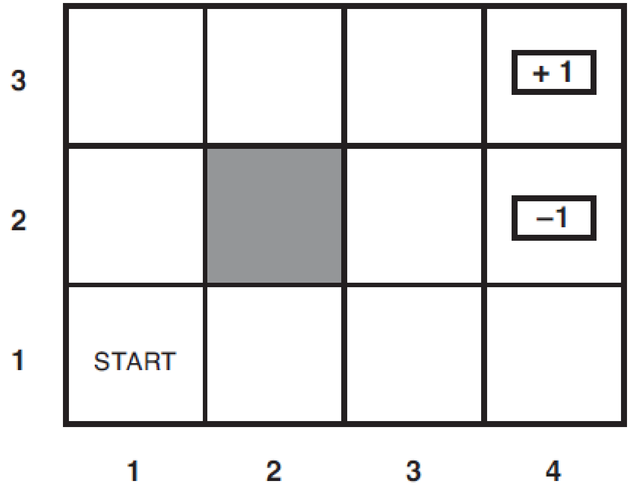


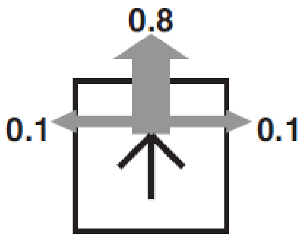
Value Iteration

- If we know the solution to sub-problems $v_*(s')$
- Then solution $v_*(s')$ can be found by one-step lookahead;

$$v(s) = R_s^a + \gamma \max_{a \in A} \sum_{s' \in S} P_{ss'}^a \, v(s')$$

- The idea of value iteration is to apply these updates iteratively





$$v(1,1) = -0.04 + \gamma \max \begin{array}{l} \left[\begin{array}{c} 0.8v(1,2) + 0.1v(2,1) + 0.1v(1,1), \\ 0.9v(1,1) + 0.1v(1,2), \\ 0.9v(1,1) + 0.1v(2,1), \\ 0.8v(2,1) + 0.1v(1,2) + 0.1v(1,1) \end{array}\right] \end{array} \begin{array}{l} \text{Up} \\ \text{Left} \\ \text{Down} \\ \text{Right} \end{array}$$

Convergence of Value Iteration

- Contraction
  - A function when applied to two difference inputs, produces two output values that are "close together", by some constant factor, than the original inputs
  - A function "divided two" → a contraction

$$4 \div 2 = 2$$
$$6 \div 2 = 3$$
$$\text{A fixed point} = 0(\text{unchanged})$$

  - Two properties of contractions
    - A contraction has only one fixed point
    - The value must get closer to the fixed point(repeated application of a contraction always reaches the fixed point in the limit)

Convergence of Value Iteration

- Contraction
  - Suppose Bellman update as operator B, and V_k as vector of value function

$$V_{k+1} \leftarrow BV_k$$

  - L_p norm

$$\|x\|_p = \left(\sum_{i \in I} |x_i|^p\right)^{1/p} = (|x_1|^p + \cdots + |x_1|^p)^{1/p}$$

$$\|x\|_\infty = \max(|x_1|, \ldots, |x_n|)$$

  - The max norm(L_∞ norm) between two vectors $\|V - V'\|$
    - The maximum difference between any two corresponding elements

$$\|BV_k - BV_k'\| \le \gamma \|V_k - V_k'\|$$

    - That is, the Bellman update is a contraction by factor of $\gamma (\gamma < 1)$

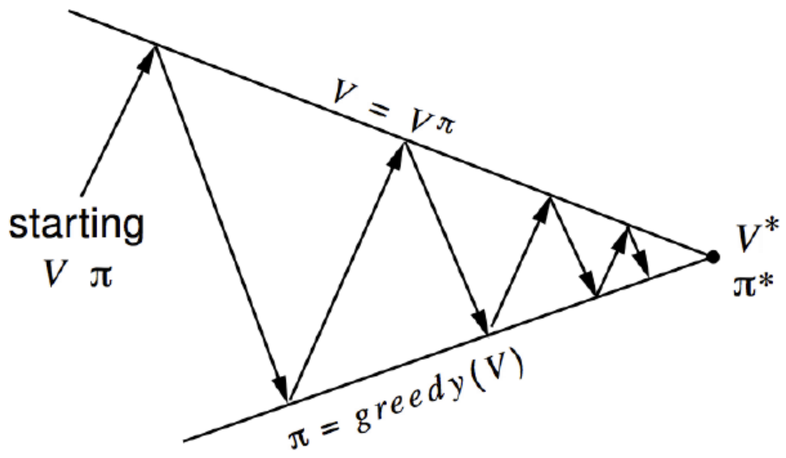
Policy Iteration

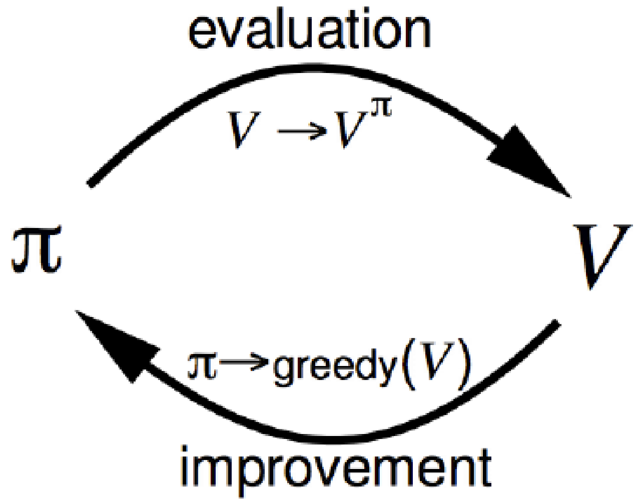
- Policy Iteration alternate the following two steps:
  - Policy Evaluation : evaluate policy π using Bellman expectation equation(for all states)

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

  - Policy Improvement : improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$





Limitation of DP and Open Challenges

- Limitation of Dynamic Programming
  - High computation in large scale MDPs
  - Need known model
- One of Solutions : (Reinforcement) Learning
  - Monte carlo, Sarsa, Q-learning, …
  - For overcoming tabular MDPs → Deep Reinforcement Learning

Limitation of DP and Open Challenges

- Limitation of Dynamic Programming
  - High computation in large scale MDPs
  - Need known model
- One of Solutions : (Reinforcement) Learning
  - Monte carlo, Sarsa, Q-learning, …
  - For overcoming tabular MDPs → Deep Reinforcement Learning
- What should be next challenge? → Sequential Reasoning!
- But how?(Gary Marcus vs Yoshua Bengio)

GOFAI (Good Old-Fashioned AI) → Deep Neural Networks

Gradient-based Learning

GOFAI (Good Old-Fashioned AI)

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

III. Model-Free Reinforcement Learning

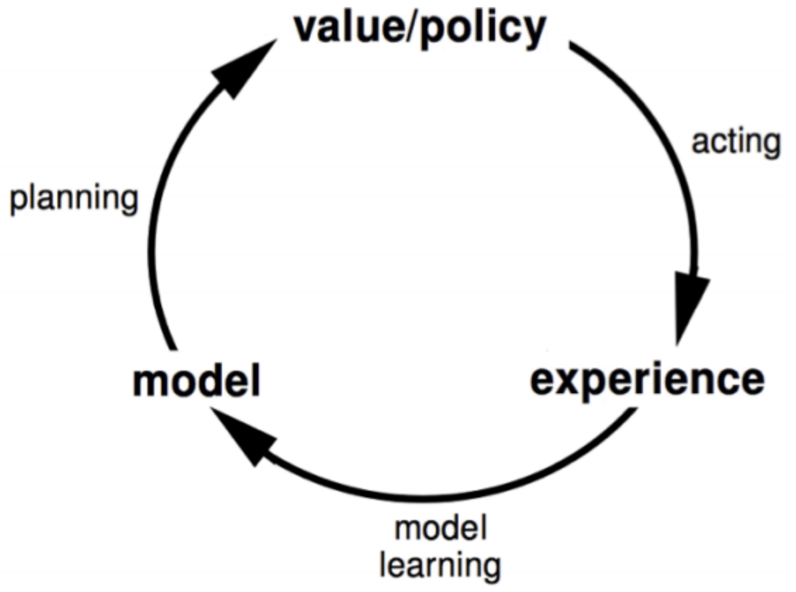
Model-Free Reinforcement Learning

- Dynamic Programming
  - Solve a known MDPs using planning method

- Reinforcement Learning
  - Estimate the value function(or policies) of unknown MDPs

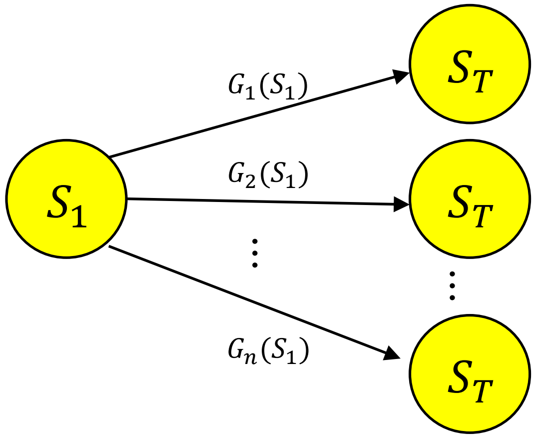
- Model-based Reinforcement Learning
  - Not cover in this presentation
  - Advantages
    - Can efficiently learn model by SL methods
    - Can reason about model uncertainty

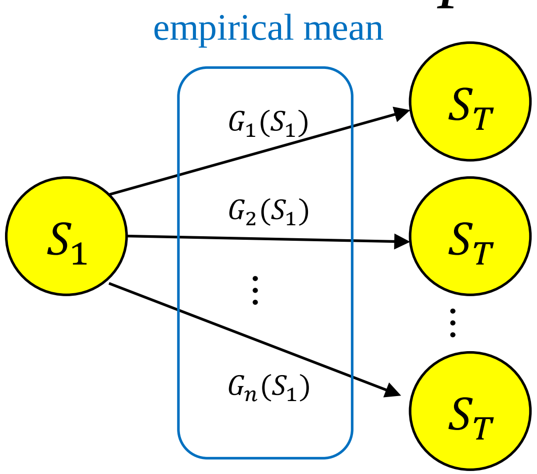
  - Disadvantages
    - Two sources of approximation error (model, value/policy)



Monte-Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC method: Model-Free(No knowledge of MDP transition / rewards)
- Can only apply MC to episodic MDPs
  - All episodes must terminate
- Goal: learn v_π from episodes of experience under policy π
  - $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$
  - $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
- MC policy evaluation uses *empirical mean return* instead of *expected return*





Monte-Carlo Incremental Implementation

$$v_\pi(s) \sim \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s) \ \rightarrow Estimate\ a\ value\ fuction\ as\ the\ average\ of\ the\ returns$$

Proof of Monte Carlo Prediction

$$\begin{aligned} V_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} G_i = \frac{1}{n}(G_n + \sum_{i=1}^{n-1} G_i) \\ &= \frac{1}{n}\left(G_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} G_i\right) \\ &= \frac{1}{n}(G_n + (n-1)V_n) \\ &= V_n + \frac{1}{n}(G_n - V_n) \end{aligned}$$

$$\therefore V_{new}(s) \leftarrow V_{old}(s) + \alpha(G(s) - V_{old}(s))$$

Monte-Carlo Incremental Implementation

$$v_\pi(s) \sim \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s) \;\rightarrow \textit{Estimate a value fuction as the average of the returns}$$

Proof of Monte Carlo Prediction

$$V_{n+1} = \frac{1}{n}\sum_{i=1}^{n} G_i = \frac{1}{n}(G_n + \sum_{i=1}^{n-1} G_i)$$

$$= \frac{1}{n}\left(G_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} G_i\right)$$

$$= \frac{1}{n}(G_n + (n-1)V_n)$$

$$= V_n + \frac{1}{n}(G_n - V_n)$$

$$\therefore V_{new}(s) \leftarrow V_{old}(s) + \alpha(G(s) - V_{old}(s))$$

Temporal-Difference Reinforcement Learning

- TD methods learn directly from episodes of experience
- TD methods: Model-Free(No knowledge of MDP transition / rewards)
- TD learns from incomplete episodes(guess towards a guess)
- Goal: learn v_π from episodes of experience under policy π
- Incremental Implementation(vs MC)
  - Monte-Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha(\underline{G_t} - V(S_t))$$

unbiased estimation(high variance, zero bias)

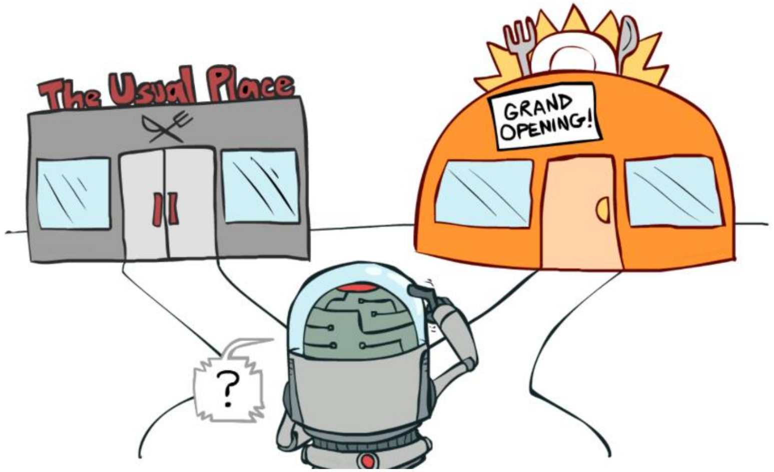
  - Temporal-Difference: TD(0), Bootstrapping

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma \underline{V(S_{t+1})} - V(S_t))$$

biased estimation(low variance, some bias)
but can be online learned

Model-Free Control: Exploration & Exploitation

- ϵ-Greedy Exploration
  - Simplest idea for ensuring continual exploration
  - All actions are tried with non-zero probability
  - With probability $1-\epsilon$ choose the greedy action
  - With probability ϵ choose an action at random



$$\pi(a|s) = \begin{cases} \dfrac{\epsilon}{|A(S_t)|} + 1 - \epsilon & \text{if } a^* = \text{argmax}_{a\in A}\, Q(s,a) \\ \dfrac{\epsilon}{|A(S_t)|} & \text{otherwise} \end{cases}$$

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:
        $$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Model-Free Control

- For more details…(Not cover in this presentation)
  - On policy / Off policy Control(Sarsa vs Q-learning)
  - Function Approximation(Deep Reinforcement Learning, e.g. Deep Q Networks)
  - Policy-based RL / Value-based RL
  - Various exploration method(e.g. Maximum Entropy RL)
  - etc.

절대 바빴거나 몰라서 안 다루는 것이 아닙니…
오늘 스터디 발표 시간을 지키기 위해…ㅎ

IV. POMDPs(Partially Observable Markov Decision Processes)

Definition of POMDPs

- A POMDP is an MDP with hidden states(hidden Markov model with actions)
- A POMDP is a tuple $(S, A, O, P, R, Z, \gamma)$
  - S is a finite set of states
  - A is a finite set of actions
  - O is a finite set of observations
  - P is a state transition probability matrix : $P_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
  - R is a reward function : $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - Z is an observation function : $Z_{s'o}^{a} = \mathbb{P}[O_{t+1} = o | S_{t+1} = s', A_t = a]$
  - γ is discount factor $\gamma \in [0,1]$

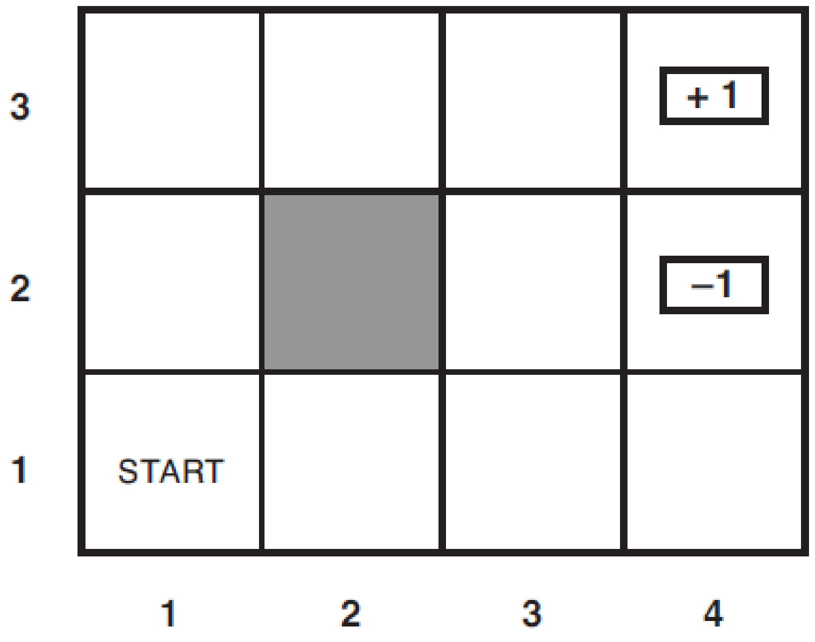
- A History H_t is a sequence of actions, observations and rewards

$$H_t = A_o, O_1, R_1, \ldots, A_{t-1}, O_t, R_t$$

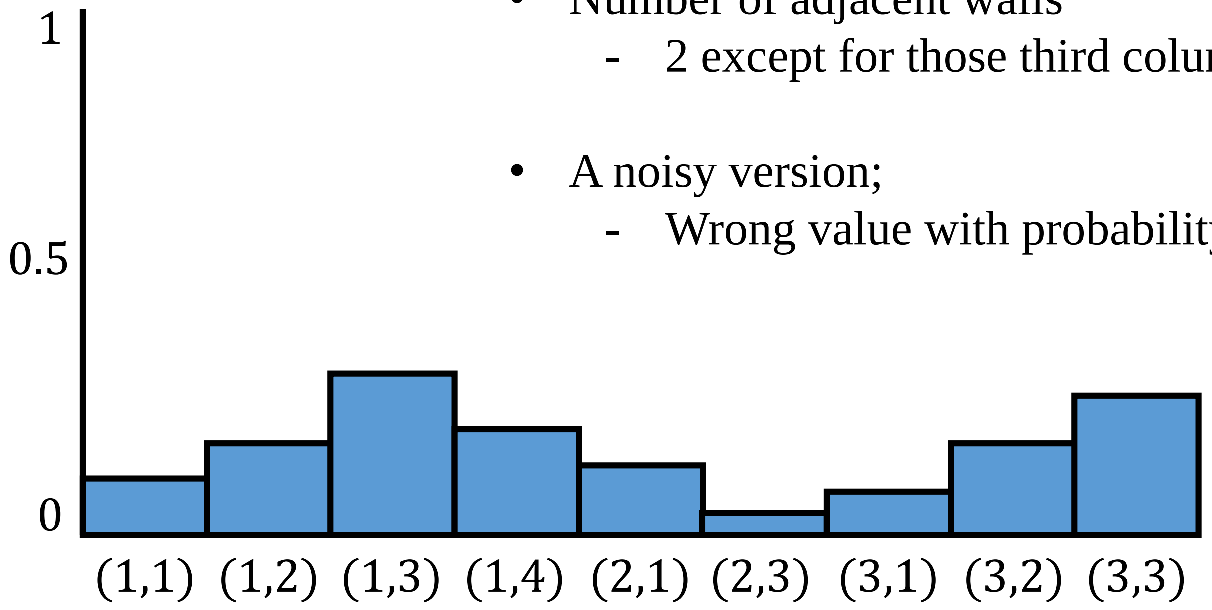
Belief States

- A belief state $b(h)$ is a probability distribution over states, conditioned on the history h

$$b(h) = (\mathbb{P}[S_t = s^1 | H_t = h], \dots, \mathbb{P}[S_t = s^n | H_t = h])$$



Environment



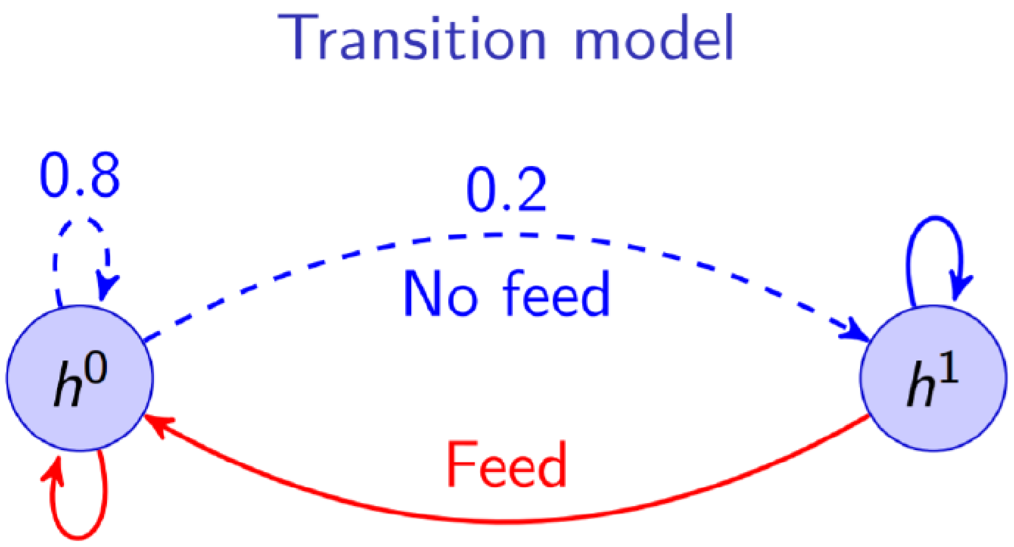
Belief state $b(h)$

Example of observation;

- Number of adjacent walls
  - 2 except for those third column(1)
- A noisy version;
  - Wrong value with probability 0.1

Example of POMDPs

- Crying Baby Problem
  - Need to decide whether to feed baby given whether baby is crying
  - Crying is a noisy indication that the baby is hungry



$P(c^1|h^0) = 0.2$ (cry when not hungry)
$P(c^1|h^1) = 0.8$ (cry when hungry)

Example of POMDPs

- Computing Belief States
  - Begin with some initial belief state b prior to any observations
  - Compute new belief state b' based on current belief state b, action a, observation o

$$
\begin{aligned}
b'(s') &= P(s'|o,a,b) \\
&= P(o|s',a,b)P(s'|a,b) \\
&= O(o|s',a)P(s'|a,b) \\
&= O(o|s',a)\sum_{s\in S} P(s'|a,b,s)P(s|a,b) \\
&= O(o|s',a)\sum_{s\in S} T(s'|s,a)\color{red}{b(s)}
\end{aligned}
$$

Example of POMDPs

- Computing Belief States
  - Begin with some initial belief state b prior to any observations
  - Compute new belief state b' based on current belief state b, action a, observation o

$$
\begin{aligned}
b'(s') &= P(s'|o,a,b) \\
&= P(o|s',a,b)P(s'|a,b) \\
&= O(o|s',a)P(s'|a,b) \\
&= O(o|s',a)\sum_{s\in S} P(s'|a,b,s)P(s|a,b) \\
&= O(o|s',a)\sum_{s\in S} T(s'|s,a)\color{red}{b(s)}
\end{aligned}
$$

$b = (h^0, h^1) = (0.5, 0.5)$

No feed, cry

$b = (0.0928, 0.9072)$

Feed, no cry

$b = (1, 0)$

No feed, no cry

$b = (0.9759, 0.0241)$

No feed, no cry

$b = (0.9701, 0.0299)$

No feed, cry

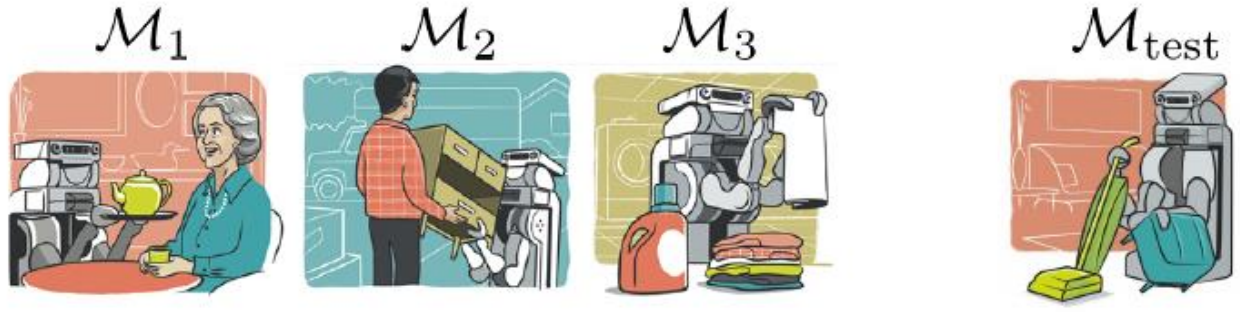
$b = (0.4624, 0.5376)$

V. Meta Reinforcement Learning as POMDPs

Meta Reinforcement Learning as POMDPs

- What can we do for AGI?
- I think RL is not enough, we need RL+sth
- Can we meta-learn reinforcement learning algorithm that are much more efficient?
- Reinforcement Learning

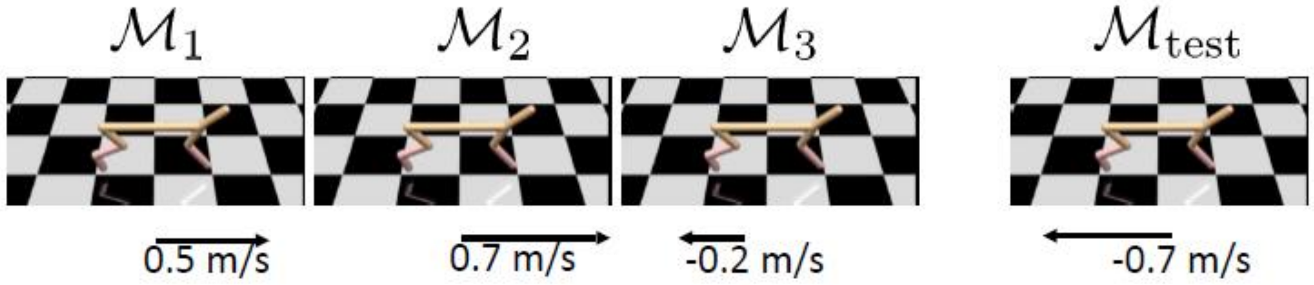
$$\theta^* = \underset{\theta}{\text{argmax}}\, \mathbb{E}_{\pi_\theta(\tau)}[R(\tau)]$$
$$= f_{\text{RL}}(\mathcal{M})$$



- Meta Reinforcement Learning

$$\theta^* = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \mathbb{E}_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_\theta(\mathcal{M}_i)$



Problem Definition

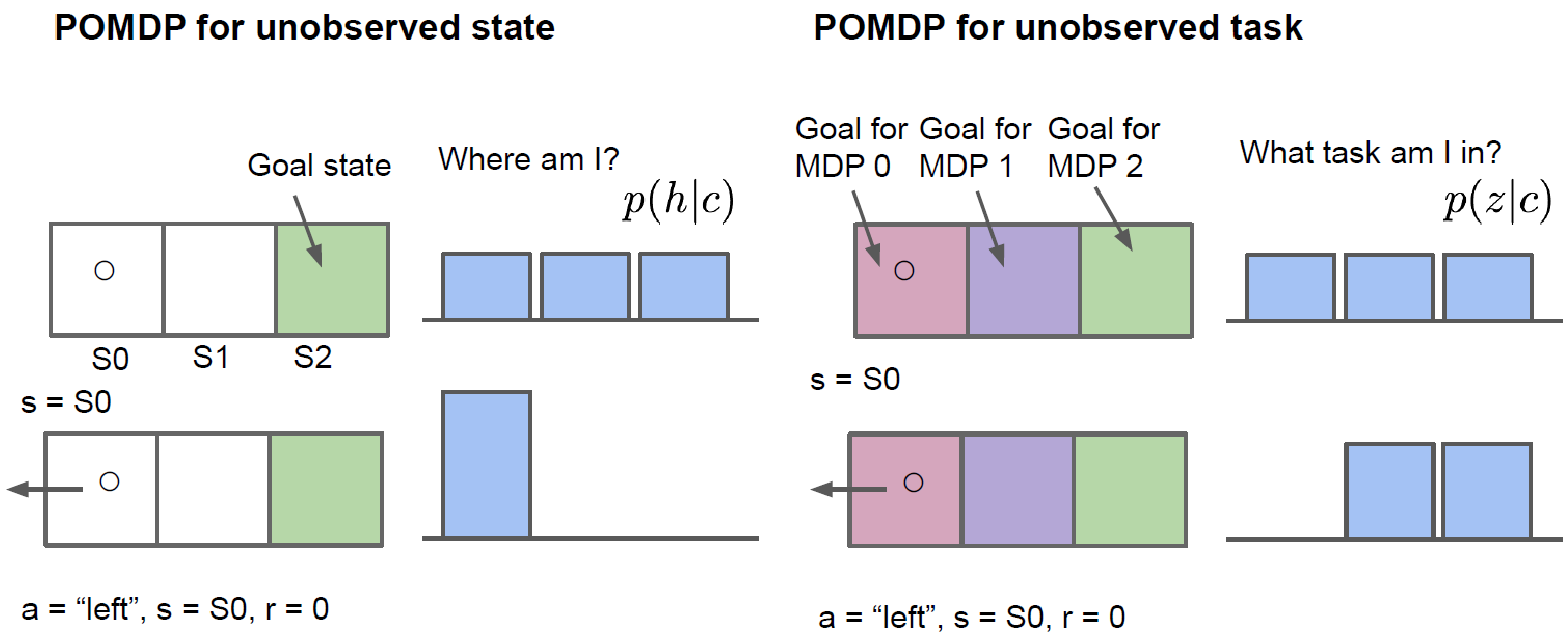
- We can do this Black-box adaptation approach(with recurrent nets), or as optimization problem(e.g. MAML), but my research focus on…

- Meta Reinforcement Learning as partially observable RL;
  - Control-Inference duality problem
  - $\pi_\theta(a|s,z), \quad z_t \sim p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$
  - $z \rightarrow$ everything needed to solve the task

- That is,
  - Learning a task $=$ inferring z
  - Encapsulated information policy $\pi_\theta(a|s,z)$ must solve current task

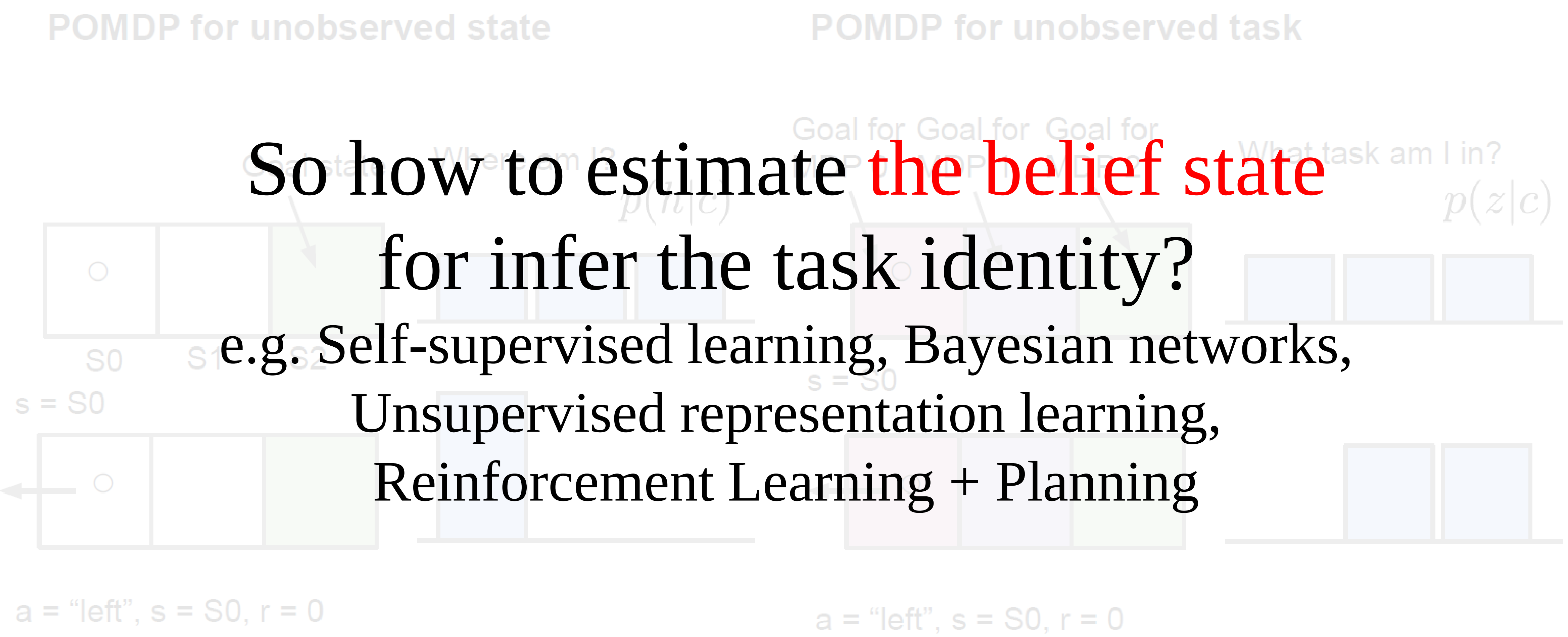
Meta Reinforcement Learning as POMDPs

- Model belief over latent task variables



Meta Reinforcement Learning as POMDPs

- Model belief over latent task variables



So how to estimate the belief state
for infer the task identity?

e.g. Self-supervised learning, Bayesian networks,
Unsupervised representation learning,
Reinforcement Learning + Planning

Thank you for your attention!