An Introduction to Reinforcement Learning

Prof. Mingkui Tan

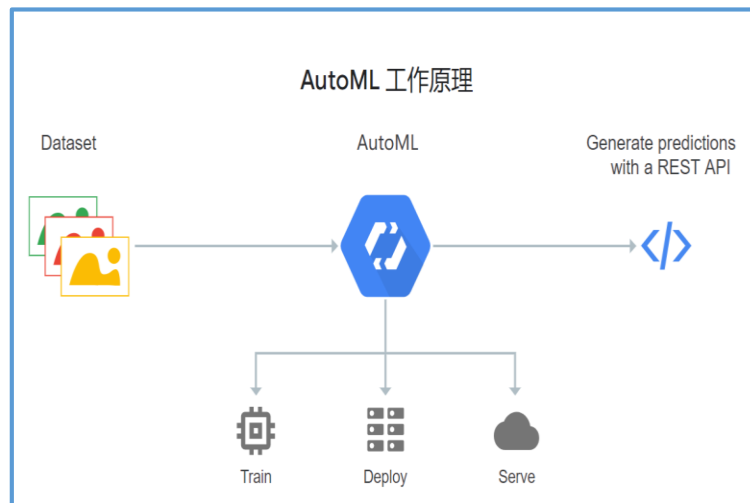
SCUT Machine Intelligence Laboratory (SMIL)

Reinforcement Learning Applications

AlphaGo, DeepMind

Dota 2 AI, OpenAI

王者荣耀觉悟 AI，腾讯



自动化机器学习平台，Google

OpenAI Five vs OG (TI8 Champion)

AI Learns to Park

Multi-Agent Hide and Seek

AI Taught Itself to Walk

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process for Reinforcement Learning

- Markov Process
- Markov Reward Process
- Markov Decision Process

3 Policy Gradient Methods for Reinforcement Learning

4 Reinforcement Learning Example: AlphaGo

5 Summary

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process for Reinforcement Learning

- Markov Process
- Markov Reward Process
- Markov Decision Process

3 Policy Gradient Methods for Reinforcement Learning

4 Reinforcement Learning Example: AlphaGo

5 Summary

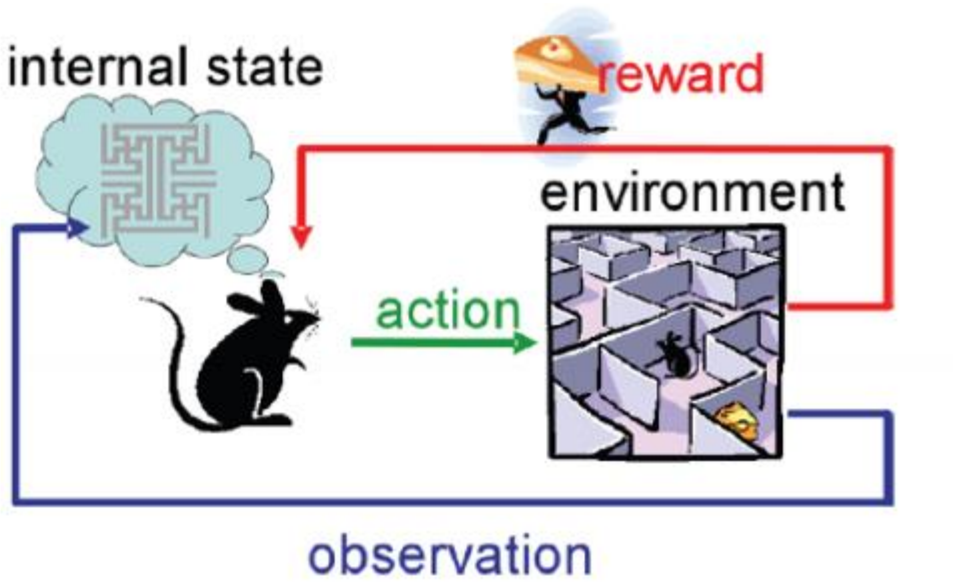
Reinforcement Learning

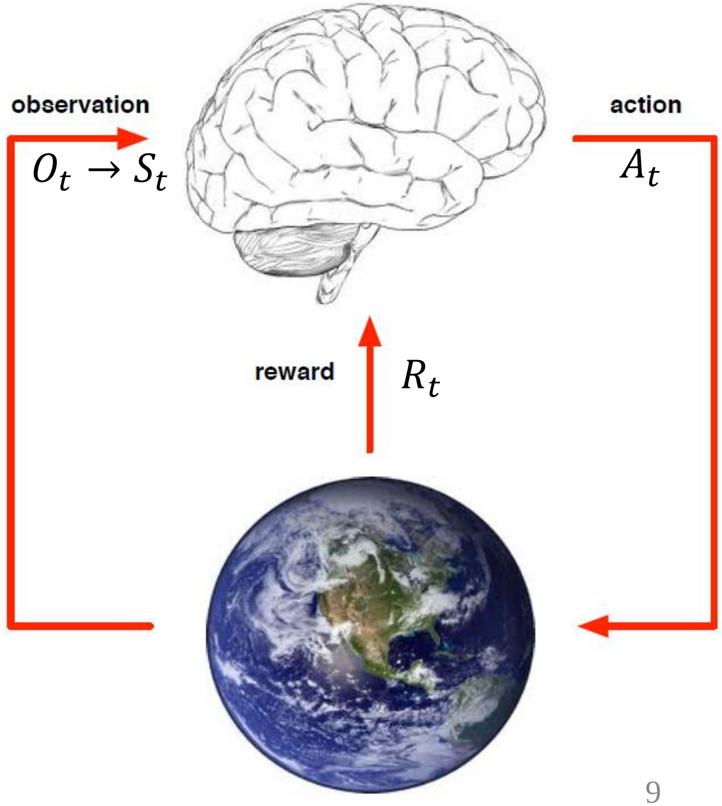
- **What is Reinforcement Learning?**

  Learning to solve sequential decision making problems

- **How it works?**

  Trial and error in a world that provides occasional rewards





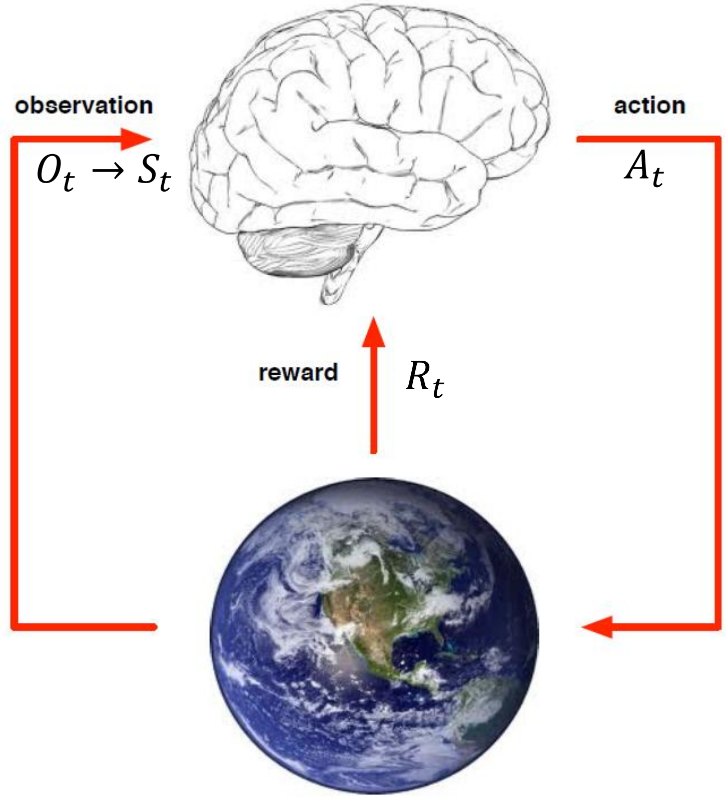
Characteristics of Reinforcement Learning

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

Agent and Environment

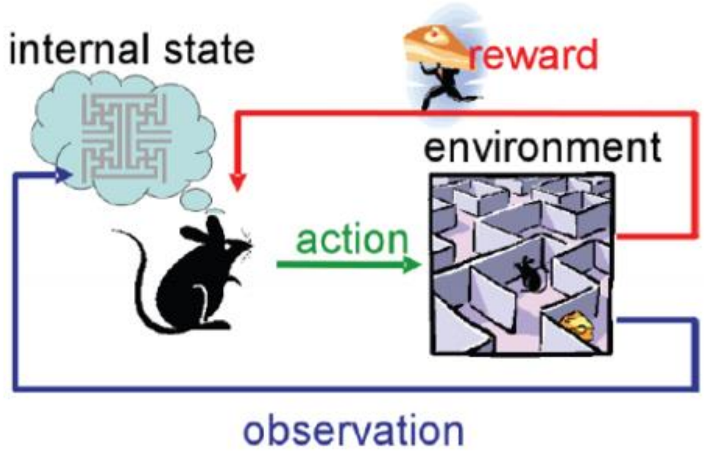
- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t (S_t)
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits scalar reward R_t
  - Emits observation $O_{t+1}(S_{t+1})$



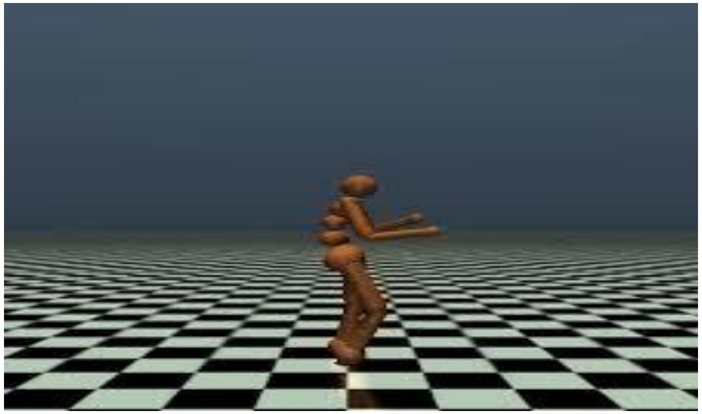
Reward

- A reward R_t is a scalar feedback signal at step t
- The agent's job is to **maximise cumulative reward**

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



Make a humanoid robot walk



- positive R_t (+1) for moving forward
- negative R_t (-1) for falling down

Super Mario

- positive reward R_t (+1) for getting a gold coin

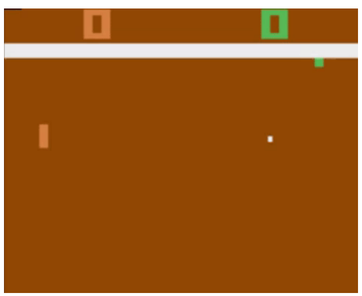
Sequential Decision Making

- Objective: Let an agent select **a series of actions** to **maximise total future rewards via** **some policy:**

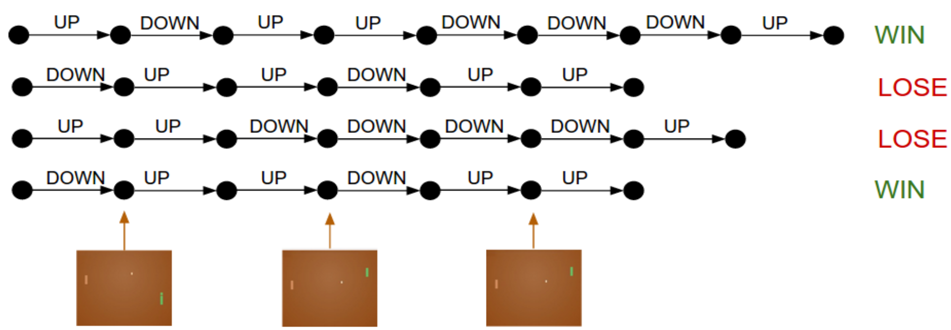
$$\pi(a \mid s) = P\left[a_t = a \mid s_t = s\right]$$

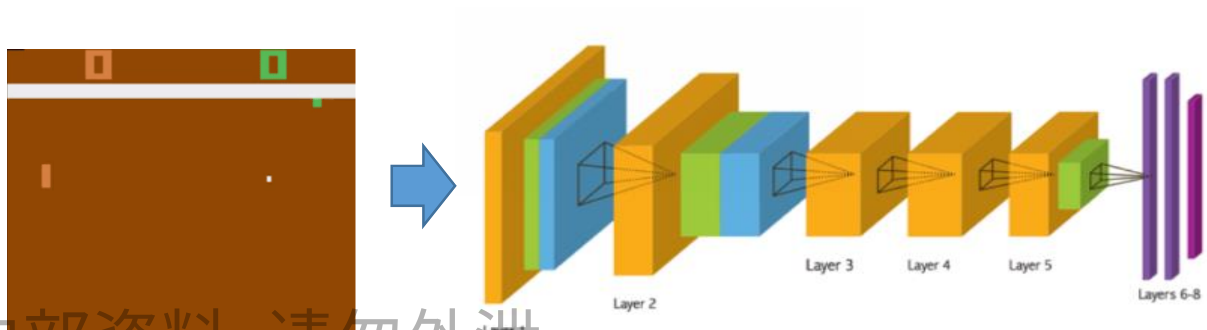
s: the current state

a: possible actions given current state: e.g., move UP or DOWN



Atari: Pong





Action: move UP or DOWN

Sequential Decision Making

- The **trajectory** is the sequence of observations, actions, rewards,

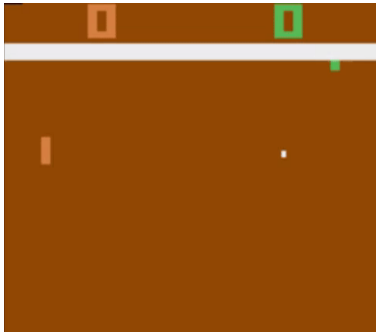
$$\tau =< S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{t-1}, A_{t-1}, R_t >$$

- **State** S_t is determined ***by previous trajectory*** **:**

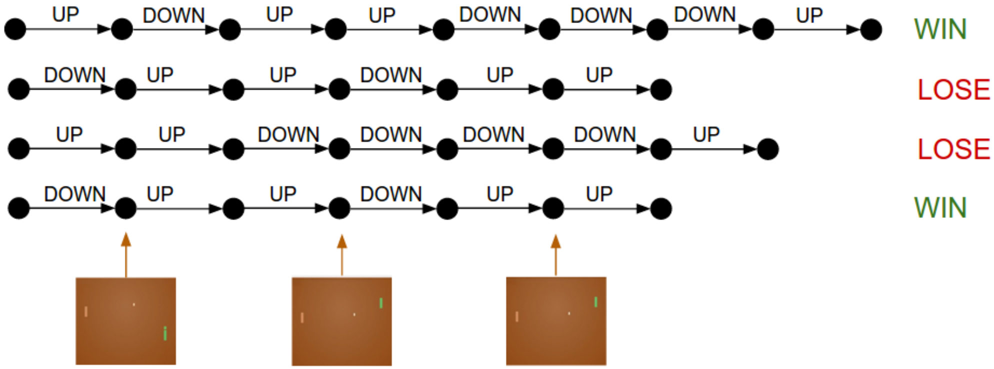
$$P(S_t) = P(S_t | S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{t-1}, A_{t-1}, R_t)$$

- The computation of the probability is much more complex!

Hypothesis: we introduce Markov Property to alleviate this issue!



Atari: Pong



Contents

1 What is Reinforcement Learning?

2 Markov Decision Process for Reinforcement Learning

- Markov Process
- Markov Reward Process
- Markov Decision Process

3 Policy Gradient Methods for Reinforcement Learning

4 Reinforcement Learning Example: AlphaGo

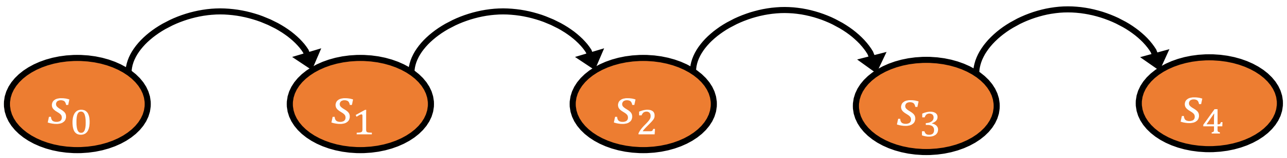
5 Summary

Markov Process

Given a sequence of random states $< \boldsymbol{s_0, s_1, \dots, s_T} >$, it satisfies

Markov Property if and only if :

$$P(s_t | s_{t-1}) = P(s_t | s_0, \dots, s_{t-2}, s_{t-1})$$



- Once the state is known, the history can be thrown away
- The state is a sufficient statistic of the future

Markov Process**：** The future is independent of the past given the present

A ***Markov Process* (*or Markov Chain*)** is a tuple $< \mathcal{S}, \mathcal{P} >$

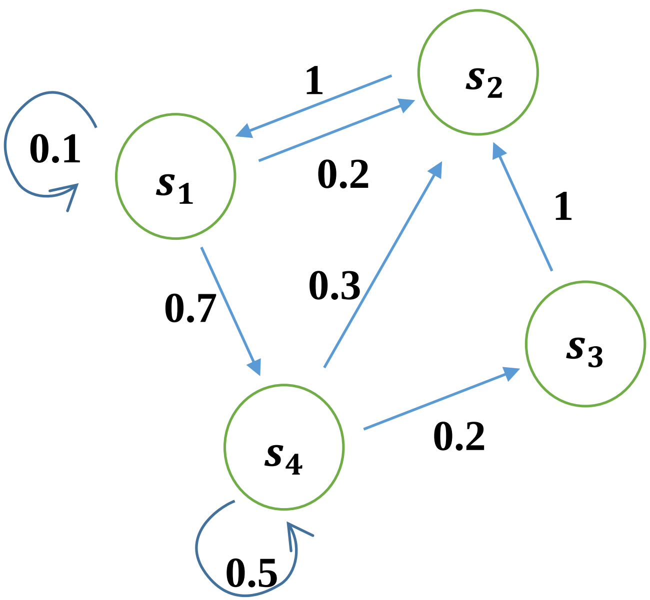
- $\mathcal{S}$ is a (finite) set of states $\mathcal{S} = \{\boldsymbol{s_0, s_1, \dots, s_T}\}$
- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'} = P[s_{t+1} = s' | s_t = s]$$

State Transition Matrix

- State transition matrix $\mathcal{P}$ specifies $P(s_{t+1} = s' | s_t = s)$

$$\mathcal{P} = \begin{pmatrix} P(s_1|s_1) & \cdots & P(s_N|s_1) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N) & \cdots & P(s_N|s_N) \end{pmatrix}$$

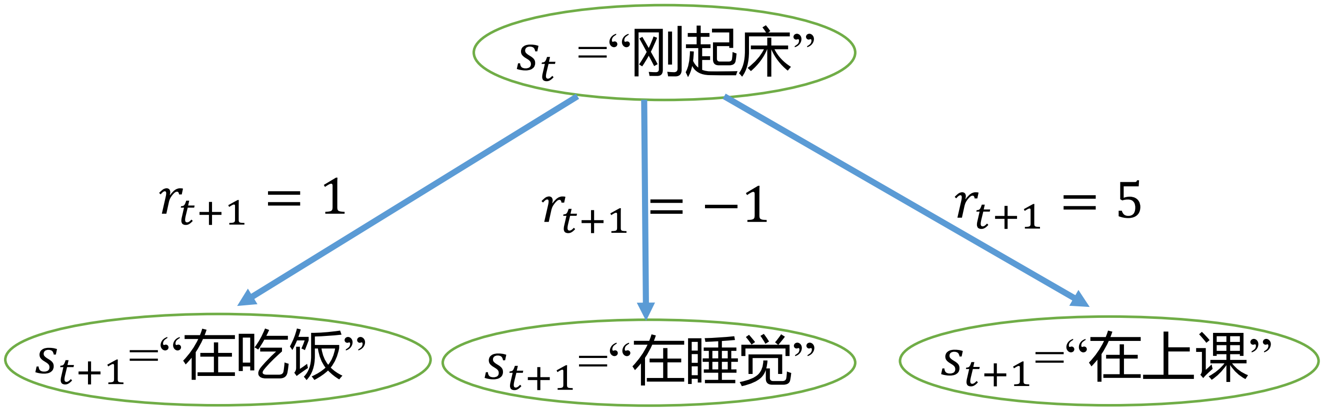


$\mathcal{S} = \{s_1, s_2, s_3, s_4\}$

$\mathcal{P} =$

	s_1	s_2	s_3	s_4
s_1	0.1	0.2	0	0.7
s_2	1	0	0	0
s_3	0	1	0	0
s_4	0	0.3	0.2	0.5

Markov Reward Process



- **Markov Reward Process** is a **Markov Chain** + **rewards**

Definition of ***Markov Reward Process***

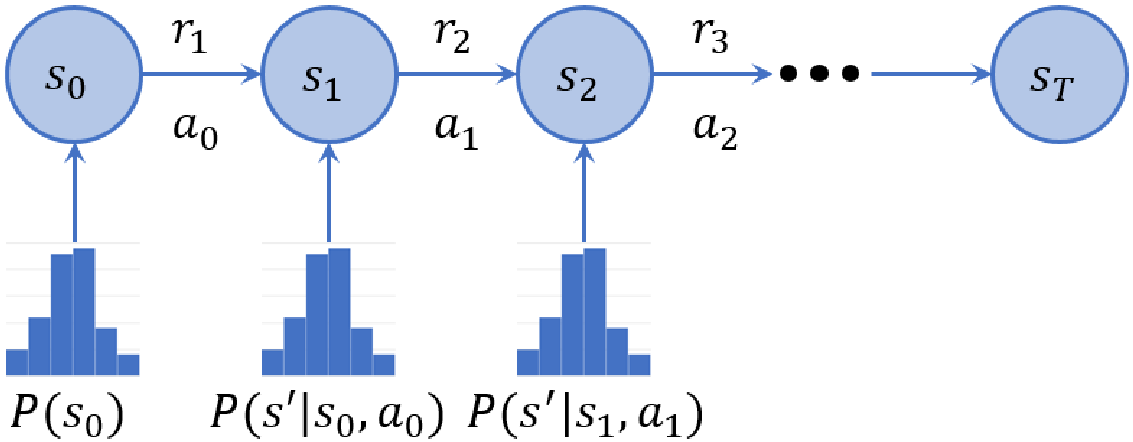
A ***Markov Reward Process*** is a tuple $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'} = P[s_{t+1} = s' | s_t = s]$$

- $\mathcal{R}$ is a reward function, $R(s_t = s) = \mathbb{E}[r_{t+1} | s_t = s]$
- γ is a discount factor, $\gamma \in [0,1]$

Markov Decision Process



- **Markov Decision Process** is a **Markov Reward Process** + **actions**

Definition of *Markov Decision Process*

A ***Markov Decision Process*** is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'}^{a} = P[s_{t+1} = s' | s_t = s, a_t = a]$$

- $\mathcal{R}$ is a reward function, $R(s_t = s, a_t = a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$
- γ is a discount factor, $\gamma \in [0,1]$

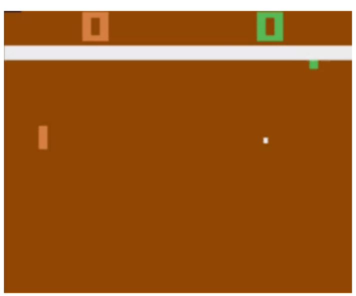
Policy

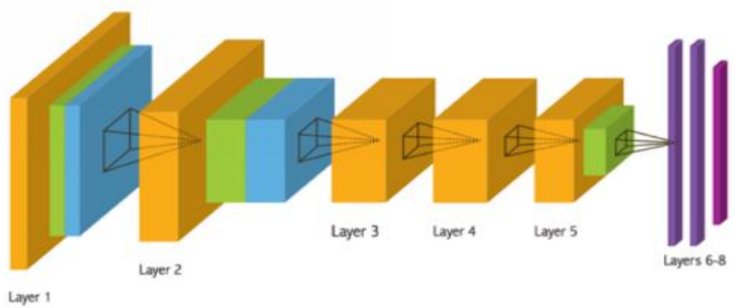
Definition of ***Policy***

A policy π is a distribution over actions given states：

$$\pi(a \mid s) = P\left[a_t = a \mid s_t = s\right]$$

- A policy $\pi(a \mid s)$: the agent's behavior model
- An MDP policy depend on the current state (not the history)
- $\pi(a \mid s)$ can be represented by neural networks





Action: move UP or DOWN

How to learn $\pi(a \mid s)$: **maximize total future rewards**!

Return

Definition of *Return* (*the* total future rewards)

The return $\boldsymbol{G_t}$ is the total discounted reward from time-step $\boldsymbol{t}$

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

- $\gamma \in [0,1]$： discount factor for weighting the future rewards
- γ is used to trade off immediate reward and delayed reward
  - γ close to 0 leads to "short-term" evaluation
  - γ close to 1 leads to "long-term" evaluation

$\boxed{\text{Why use discount factor?}}$

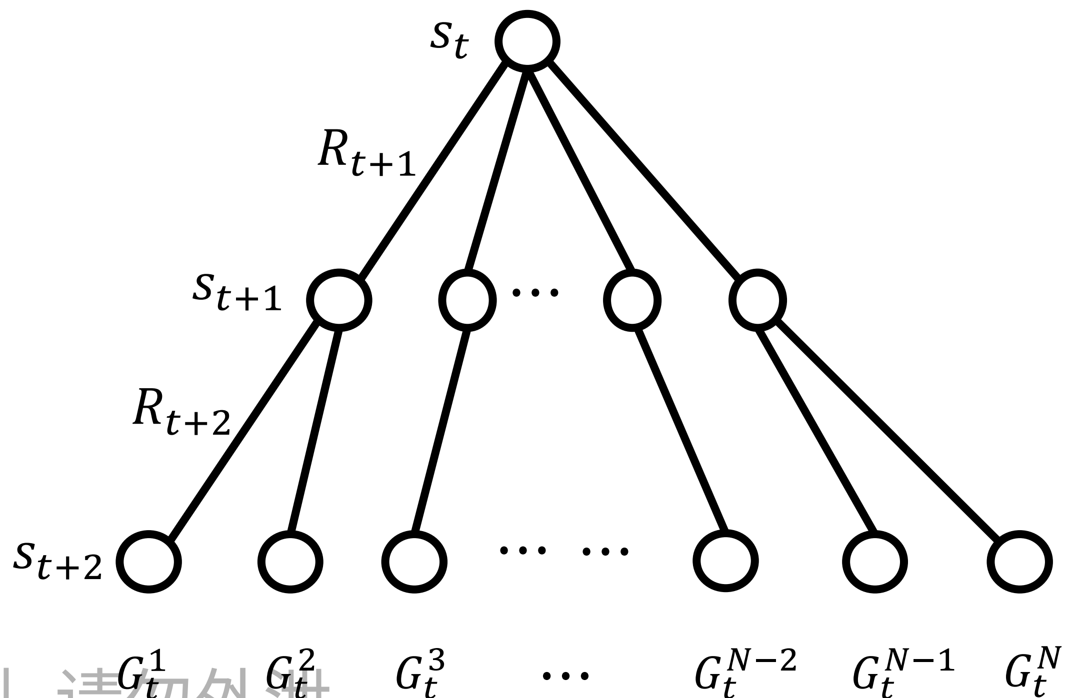
- Avoids infinite returns in cyclic Markov processes
- Ensures the convergence when solving an MDP by dynamic programming

State Value Function for MRP

Definition (State-value function)

The state-value function $V(s)$ is the expected return starting from state s:

$$V(s) = \mathbb{E}[\, G_t \mid s_t = s \,]$$



Bellman Equation for MRP

- The **state-value** function can be decomposed into immediate reward R_{t+1} and discounted value of successor state $\gamma V(s_{t+1})$

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t \mid s_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid s_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) \mid s_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid s_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) \mid s_t = s]
\end{aligned}
$$

- Bellman equation describes the iterative relations of states

$$V(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s) V(s')$$

The Bellman Equation indicates the value function of the current state can be evaluated by the next state

Bellman Equation in Matrix Form

$$\boxed{V(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s) V(s')}$$

- The Bellman equation can be expressed concisely using matrices

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}(1) \\ \vdots \\ \mathcal{R}(n) \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & \vdots & \vdots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

$$(I - \gamma \mathcal{P}) v = \mathcal{R}$$

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

Solution to MRP

- Solving MRP when the model $\mathcal{P}$ is known:

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

  - Matrix inverse takes the **complexity $O(N^3)$** for N states
  - Only possible for small MRPs
  - The model $\mathcal{P}$ must be known

- **Iterative** methods for large MRPs:
  - Dynamic Programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process for Reinforcement Learning

- Markov Process
- Markov Reward Process
- Markov Decision Process

3 Policy Gradient Methods for Reinforcement Learning

4 Reinforcement Learning Example: AlphaGo

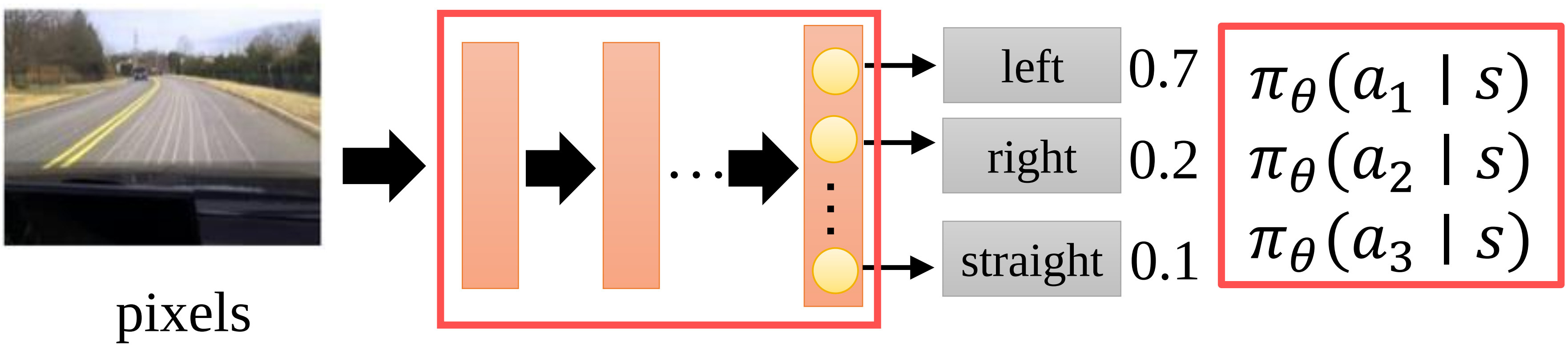
5 Summary

Policy of Agent

- Policy π can be represented by a network with parameter θ:

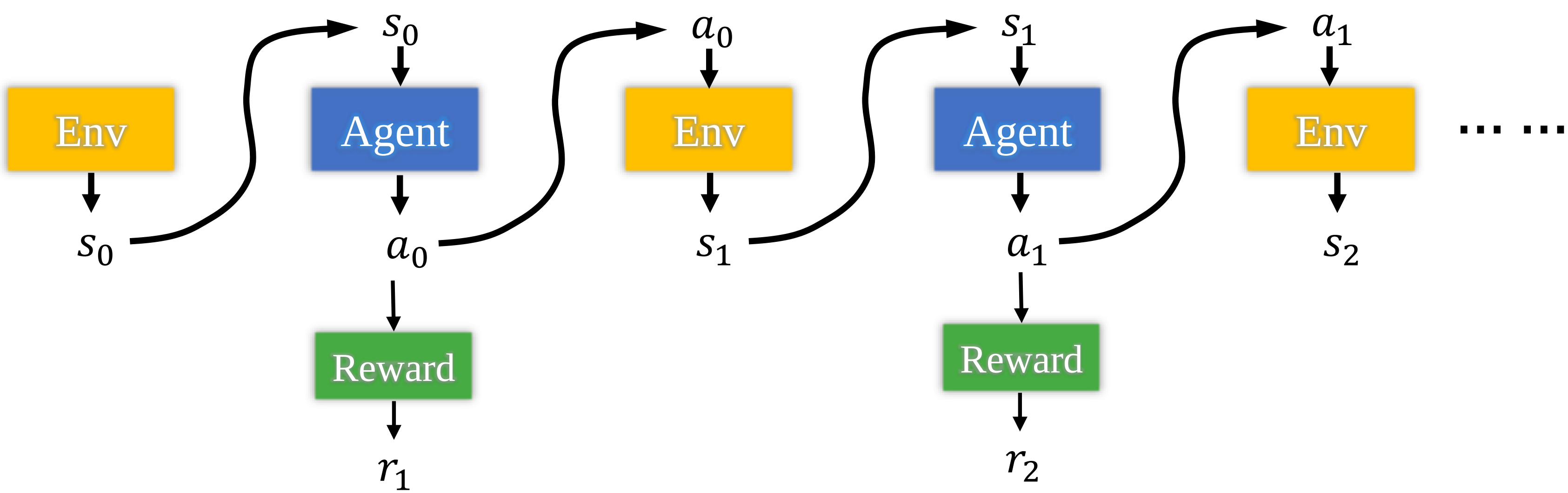
$$\pi_\theta(a \mid s) = P(a \mid s; \theta)$$

  where $\pi_\theta(a_t \mid s)$ denotes the probability of taking an action a_t given state s
- **Input**: state s
- **Output**: each action a corresponds to a neuron in output layer
- Take the action based on the output probability



$\boxed{\text{How to learn } \pi_\theta\text{?}}$

Objective Function



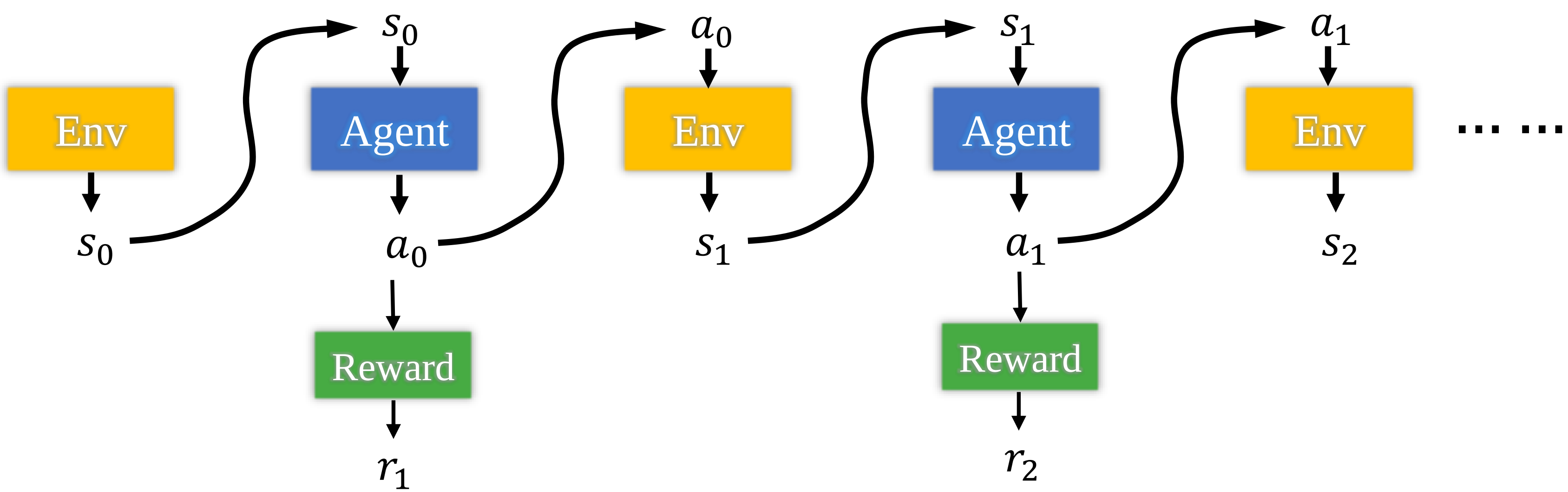
- To measure the quality of a policy π_θ, we define the objective function

$$J(\theta) = E_{\tau \sim P(\tau;\theta)}[R(\tau)] = \sum_{\tau} P(\tau;\theta)R(\tau)$$

where $R(\tau)$ is a reward of τ: $R(\tau) = \sum_{t=0}^{T} \gamma^t r_{t+1}$

What is a trajectory τ ?

What is a Trajectory τ ?



- A trajectory τ is the sequence of state, action, and reward

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2 \dots)$$

- Given θ, we can compute the probability of τ for each trajectory

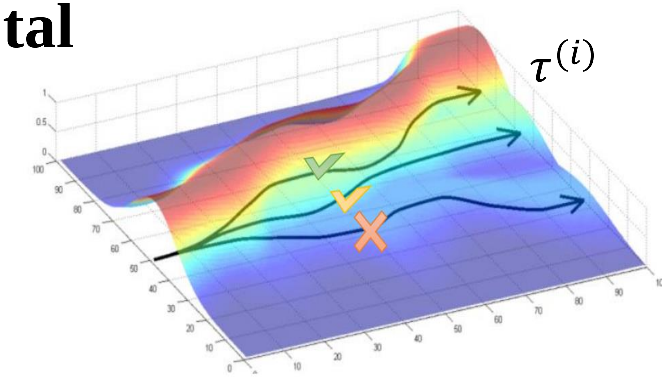
$$P(\tau\,;\theta) = p(s_0)\pi_\theta(a_0 \mid s_0)p(s_1|s_0, a_0)\ \pi_\theta(a_1 \mid s_1)p(s_2|s_1, a_1) \dots$$

$$= p(s_0)\prod_{t=0}^{T}\pi_\theta(a_t \mid s_t)p(s_{t+1}|s_t, a_t)$$

Policy Gradient

Learn the policy $\pi_\theta(a_t|s_t)$ by **maximizing total future rewards:**

$$\max_\theta \; J(\theta) = \sum_\tau P(\tau;\theta)R(\tau)$$



Policy Gradient algorithm:

Input: random initialized policy π_θ, max number of episodes N,
learning rate η

Output: π_θ

for $i = 1$ **to** N **do**

Sample a trajectory $\tau^{(i)} = \{s_0^{(i)}, a_0^{(i)}, r_1^{(i)}, \dots, s_{T_i}^{(i)}, a_{T_i}^{(i)}, r_{T_i+1}^{(i)}\}$

for $t = 0$ **to** T_i **do**

$G = \nabla_\theta J(\theta)$ *// calculate the gradient*

$\theta \leftarrow \theta + \eta G$ *// maximize the objective function by ascending the gradient*

end for

end for

How to comput $\nabla_\theta J(\theta)$?

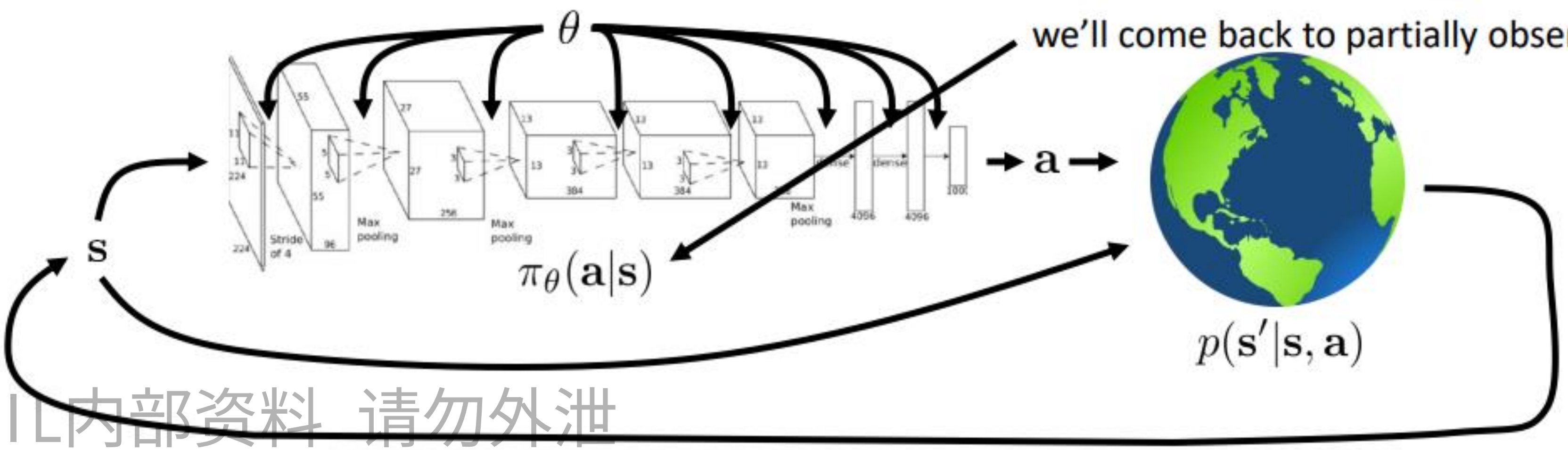
How to learn π_θ?

How to obtain $\theta^\star$?

$$\theta^\star = \mathrm{argmax}_\theta\, J(\theta) = \mathrm{argmax}_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

$$R(\tau) = \sum_t \gamma^t r\,(s_t, a_t)$$

$$P(\tau\,;\theta) = p(s_0)\prod_{t=0}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$$



Policy Gradient

How to compute $\nabla_\theta J(\theta)$?

Update Model

Given policy π_θ

$$
\begin{array}{lll}
\tau^1: & (s_0^1, a_0^1) & R(\tau^1) \\
 & (s_1^1, a_1^1) & R(\tau^1) \\
 & \vdots & \vdots \\
\tau^2: & (s_0^2, a_0^2) & R(\tau^2) \\
 & (s_1^2, a_1^2) & R(\tau^2) \\
 & \vdots & \vdots
\end{array}
$$

$$\theta \leftarrow \theta + \eta \nabla_\theta J$$

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \sum_{t=0}^{T_i} R\left(\tau^{(i)}\right) \nabla_\theta log \pi_\theta \left(a_t^{(i)} | s_t^{(i)}\right)$$

Data Collection

How to compute the Gradient $\nabla_\theta J(\theta)$?

The objective function is: $\max_\theta J(\theta) = \sum_\tau P(\tau;\theta)R(\tau)$

Taking the gradient w.r.t. θ gives

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau)$$

$$= \sum_\tau P(\tau;\theta)\boxed{\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}}R(\tau)$$

$$= \sum_\tau P(\tau;\theta)\boxed{\nabla_\theta log P(\tau;\theta)}R(\tau)$$

$$\boxed{\nabla f(x) = f(x)\nabla log f(x)}$$

$$\boxed{R(\tau) = \sum_t \gamma^t r(s_t, a_t)}$$

$$\boxed{P(\tau\,;\theta) = p(s_0)\prod_{t=0}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)}$$

Approximate the gradient,

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\boxed{\nabla_\theta log P\left(\tau^{(i)};\theta\right)}R(\tau^{(i)})$$

$\boxed{\text{How to compute } \nabla_\theta log P\left(\tau^{(i)};\theta\right)?}$

How to compute $\nabla_\theta logP\left(\tau^{(i)};\theta\right)$?

$$\boxed{P(\tau\,;\theta) = p(s_0)\prod_{t=0}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)}$$

$$\nabla_\theta logP\left(\tau^{(i)};\theta\right) = \nabla_\theta \log[\prod_{t=0}^{T}\underbrace{p\left(s_{t+1}^{(i)}\,\middle|s_t^{(i)},a_t^{(i)}\right)}_{\textbf{\textit{dynamics model}}}\cdot\underbrace{\pi_\theta\left(a_t^{(i)}|s_t^{(i)}\right)}_{\text{policy}}]$$

$$= \nabla_\theta[\sum_{t=0}^{T}\log p\left(s_{t+1}^{(i)}\,\middle|s_t^{(i)},a_t^{(i)}\right) + \sum_{t=0}^{T} log\pi_\theta\left(a_t^{(i)}|s_t^{(i)}\right)]$$

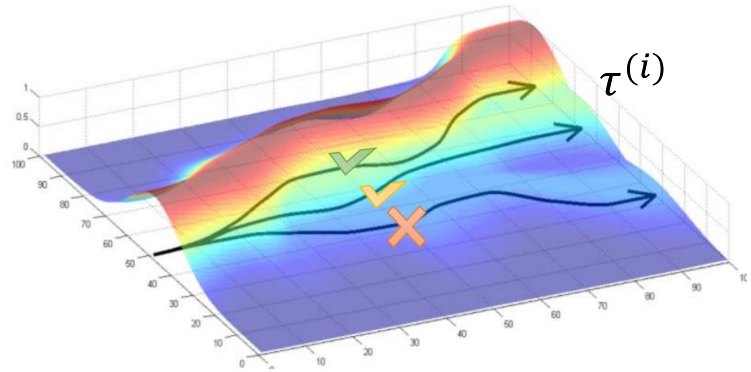
$$= \nabla_\theta \sum_{t=0}^{T} log\pi_\theta\left(a_t^{(i)}|s_t^{(i)}\right)$$

$$= \sum_{t=0}^{T}\underbrace{\nabla_\theta log\pi_\theta\left(a_t^{(i)}|s_t^{(i)}\right)}_{\textbf{\textit{no dynamics model required}}}$$

Finally, we can obtain $\nabla_\theta J(\theta)$: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} R\left(\tau^{(i)}\right)\sum_{t=0}^{T_i}\nabla_\theta log\pi_\theta\left(a_t^{(i)}|s_t^{(i)}\right)$

Policy Gradient

$$\max_{\theta} \; J(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau)$$



Policy Gradient algorithm:

Input: random initialized policy π_θ, max number of episodes N,
learning rate η

Output: π_θ

for $i = 1$ **to** N **do**

Sample a trajectory$\tau^{(i)} = \{s_0^{(i)}, a_0^{(i)}, r_1^{(i)}, \ldots, s_{T_i}^{(i)}, a_{T_i}^{(i)}, r_{T_i+1}^{(i)}\}$

for $t = 0$ **to** T_i **do**

$\nabla_\theta J(\theta) = R\left(s_t^{(i)}, a_t^{(i)}, r_{t+1}^{(i)}, \ldots, r_{T_i+1}^{(i)}\right) \nabla_\theta log\pi_\theta\left(a_t^{(i)} | s_t^{(i)}\right)$

$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$

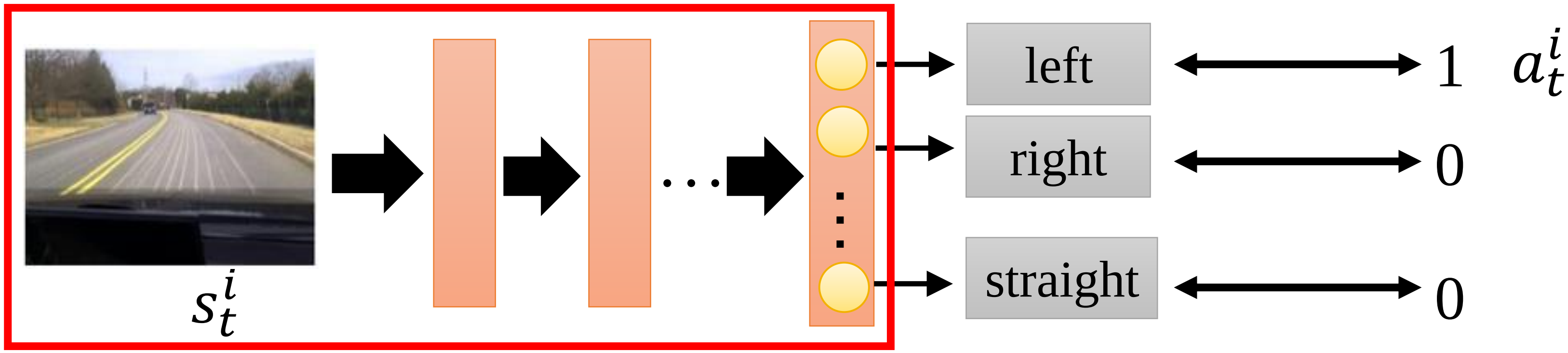
end for

end for

Differences from Gradient Descent

Consider as a classification problem

$s_t^i \quad a_t^i \quad R(\tau^i)$



Implementation for classification:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i} log p_\theta(a_t^i|s_t^i) \xrightarrow[\text{TF, PyTorch …}]{} \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i} \nabla log p_\theta(a_t^i|s_t^i)$$

Implementation for RL:

Only difference

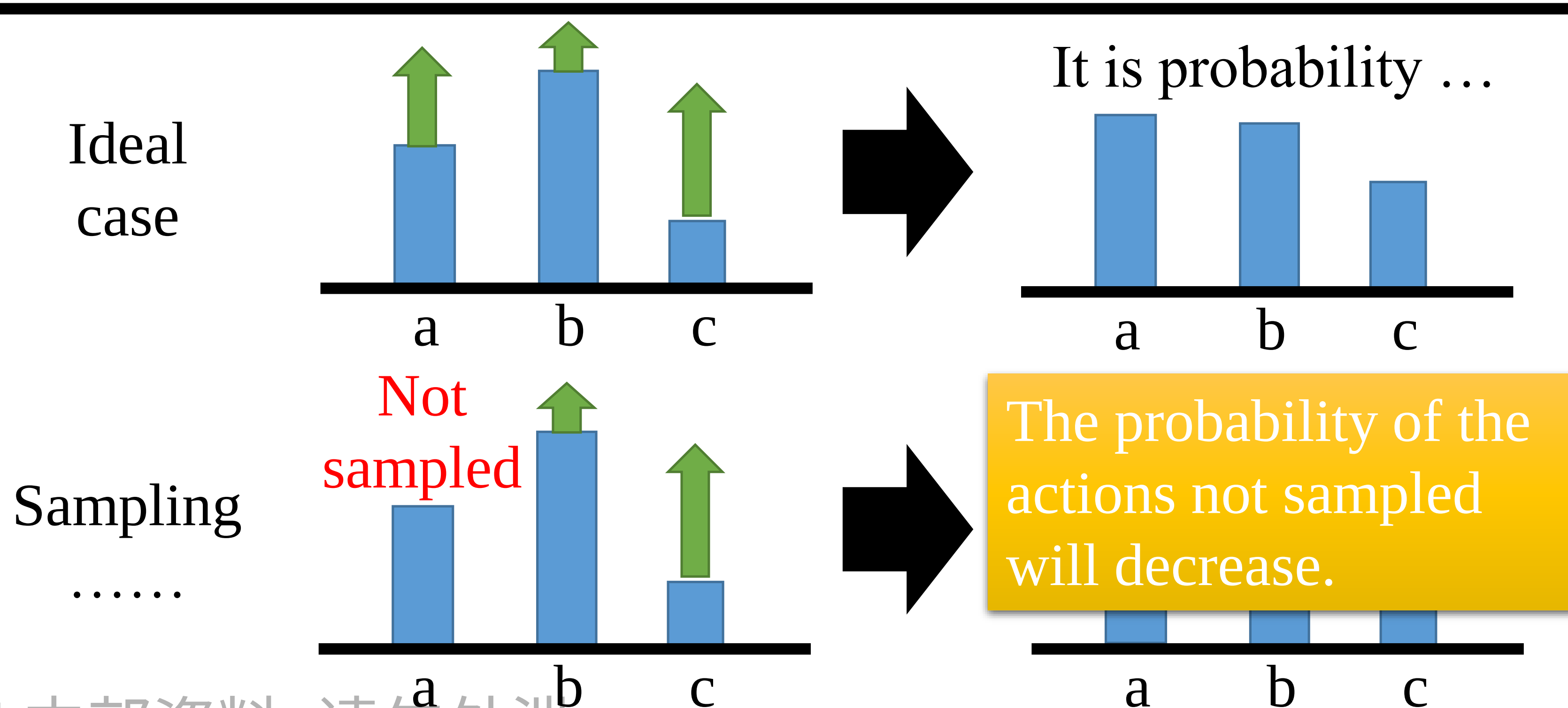
$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i} R(\tau^i) log p_\theta(a_t^i|s_t^i) \xrightarrow[\text{TF, PyTorch …}]{} \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i} R(\tau^i) \nabla log p_\theta(a_t^i|s_t^i)$$

Tip : Add a Baseline

$$\theta \leftarrow \theta + \eta \nabla J_\theta$$

It is possible that $R(\tau^i)$ is always positive

$$\nabla_\theta J \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i}\left(R(\tau^i) - b\right)\nabla \log p_\theta(a_t^i|s_t^i) \qquad b \approx E[R(\tau)]$$



Contents

1 What is Reinforcement Learning?

2 Markov Decision Process for Reinforcement Learning

- Markov Process
- Markov Reward Process
- Markov Decision Process

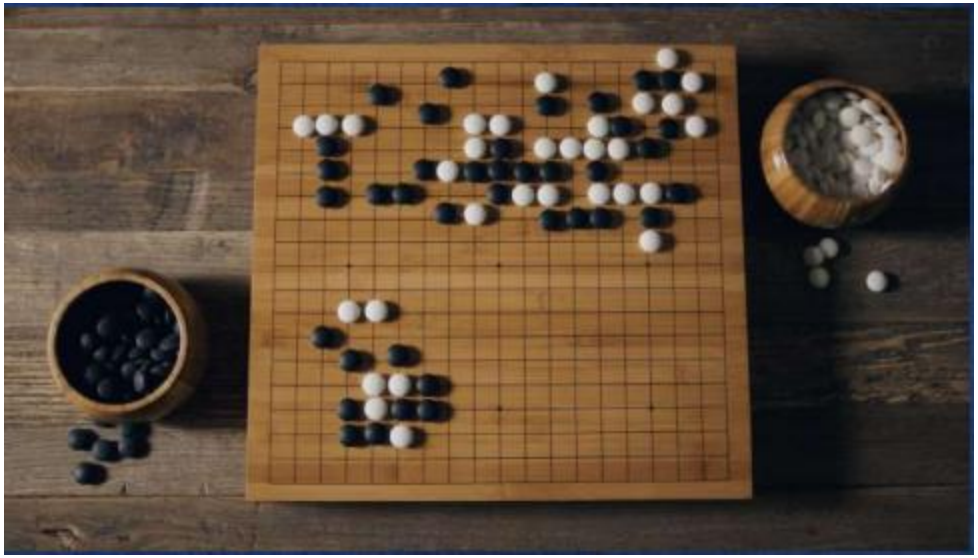
3 Policy Gradient Methods for Reinforcement Learning

4 Reinforcement Learning Example: AlphaGo

5 Summary

What is Go?

- An abstract board game for two players
- Played on 19 x 19 board
- Playing pieces are black and white stones
- Stones placed on vacant intersections of board
- Player which surrounds more territory wins



Challenges for AI in Cracking Go

- Impossible to calculate every possible move on board
- Brute-force method used by most AIs clearly fails
  - Search space is **huge**
  - Impossible for computers to evaluate who is winning
- Go requires **more intuition and experience** than just logic
- Becomes necessary to mimic human mind

AlphaGo

- A computer program that plays Go game
- Developed by Google DeepMind in 2016
- Not a pre-programmed algorithm
- Can actually learn from itself
- Reduce search space
  - Reducing action candidates
  - Board Evaluation

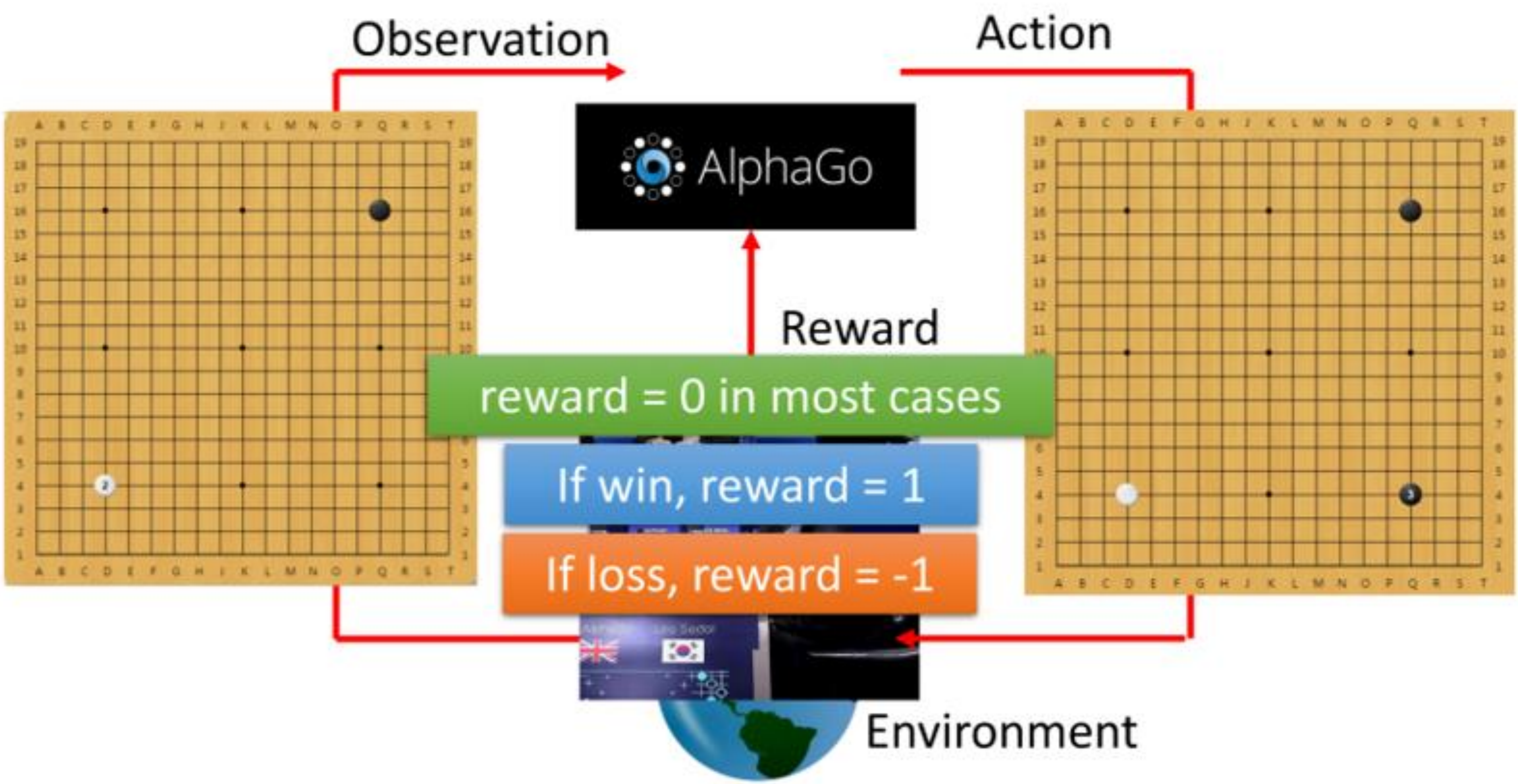
Reducing Action Candidates

- Imitating expert's moves (supervised learning)

Training: $\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$

Reducing Action Candidates

- Improving through self-plays (reinforcement learning)



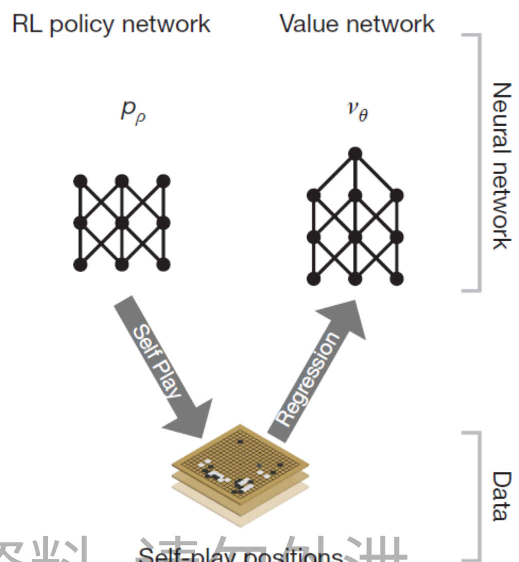
Reward

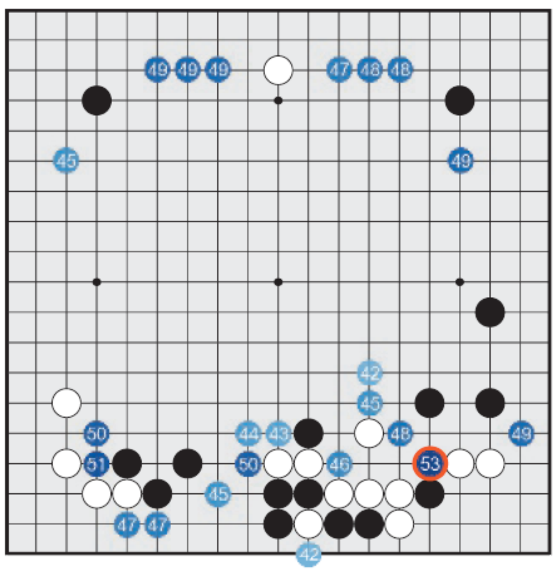
- The reward function $r(s)$ that is zero for all non-terminal state
- The reward signal is delayed
- The outcome $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time step t
  - $+1$ for winning
  - -1 for losing
- Rollout is inefficient

Board Evaluation

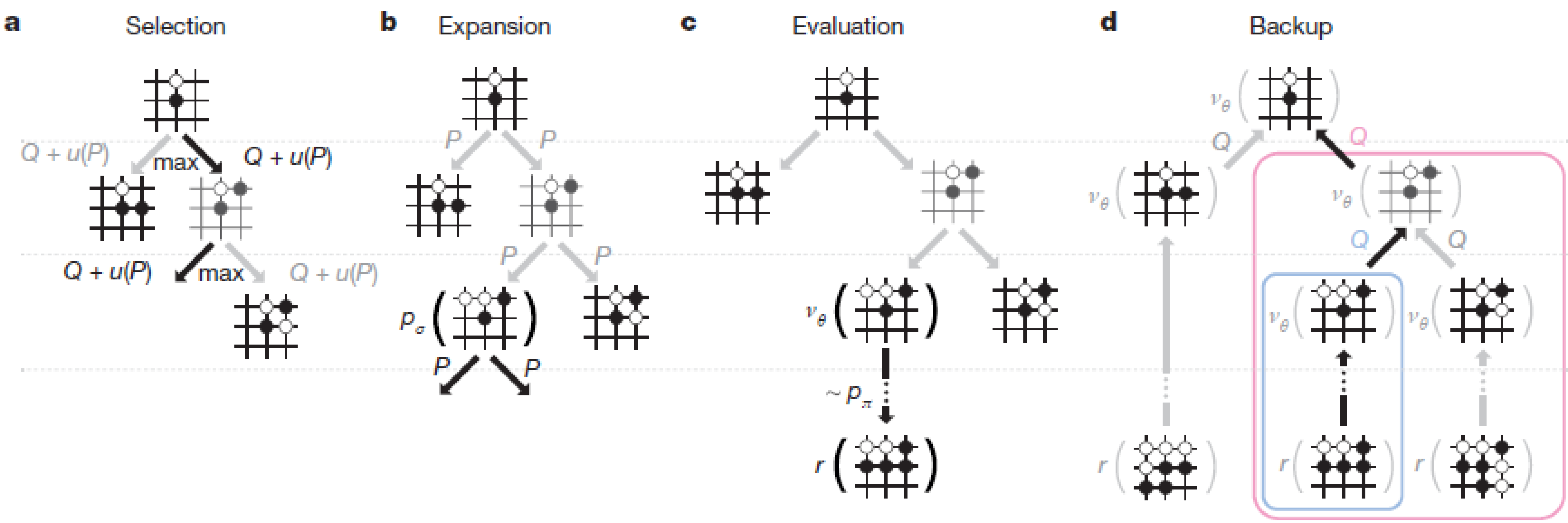
- Estimate a value function that predicts the outcome from the position
- Train the value network by regression on state-outcome pairs (s, z)
- Minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$, and the corresponding outcome z

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$





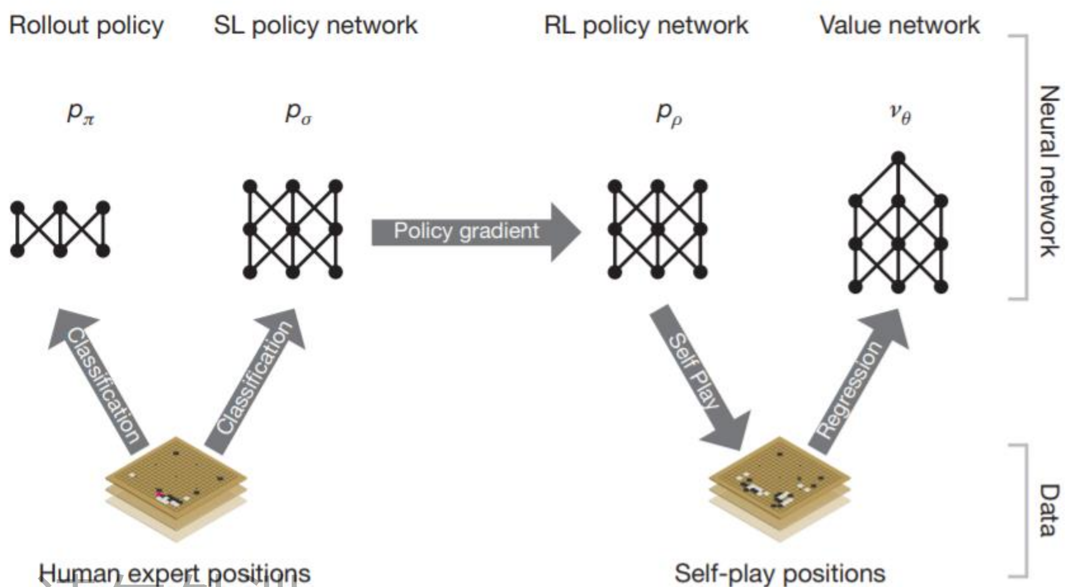
Monte Carlo Tree Search in AlphaGo



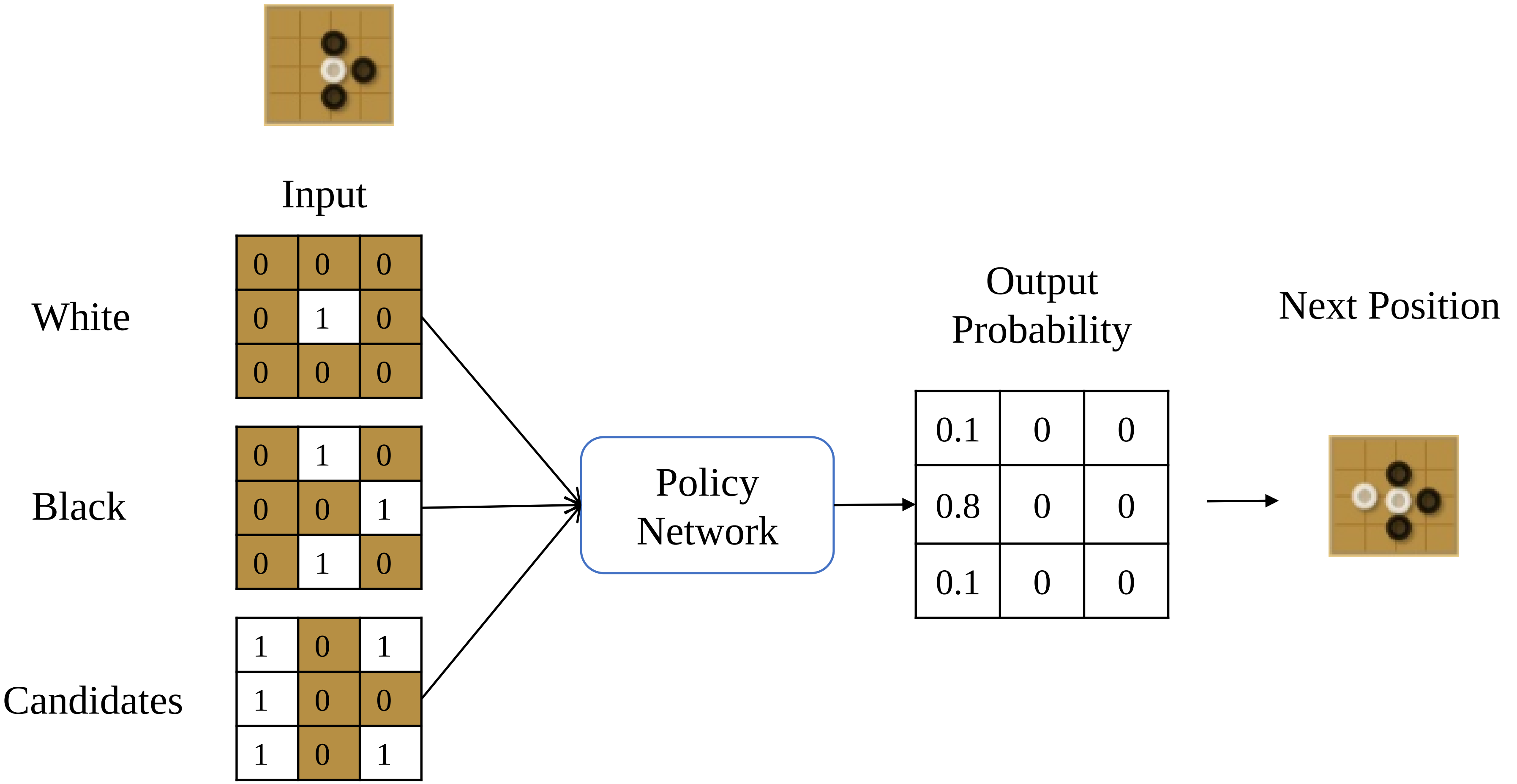
a. Selecting the edge with maximum action value Q
b. The leaf node may be expanded
c. At the end of a simulation, the leaf node is evaluated by value network
d. Action value Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action

Learning Pipeline of AlphaGo

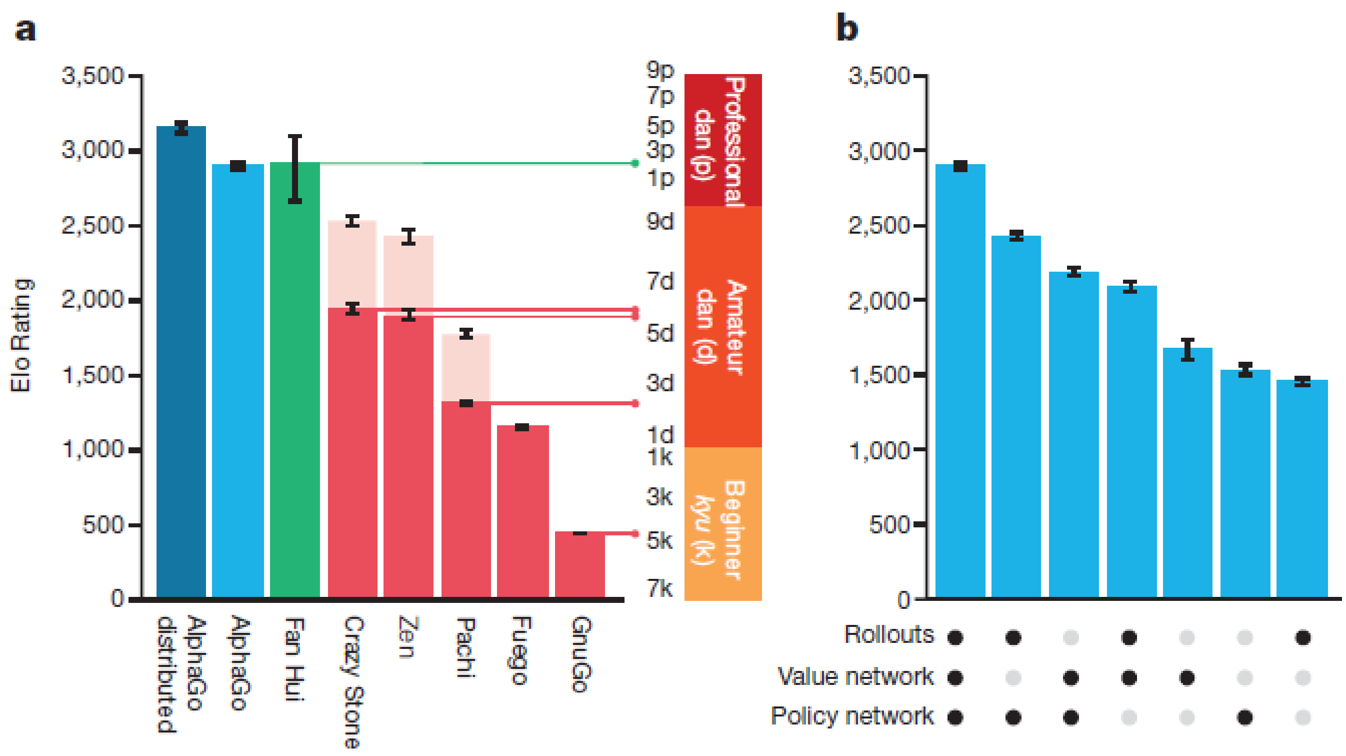
- A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions.
- A reinforcement learning (RL) policy p_σ is initialized to the SL policy
- Then p_σ is improved by policy gradient learning to maximize the outcome (i.e., winning more game) against previous versions of the policy network.
- A new dataset is generated by playing games of self-play with the RL policy network.
- A value network v_θ is trained by regression to predict the expected outcome



Playing Go Using Learnt Policy



Performance



a. Results of a tournament between different Go programs. Each program used approximately 5s computation time per move.
b. Performance of AlphaGo for different combinations of components.

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process for Reinforcement Learning

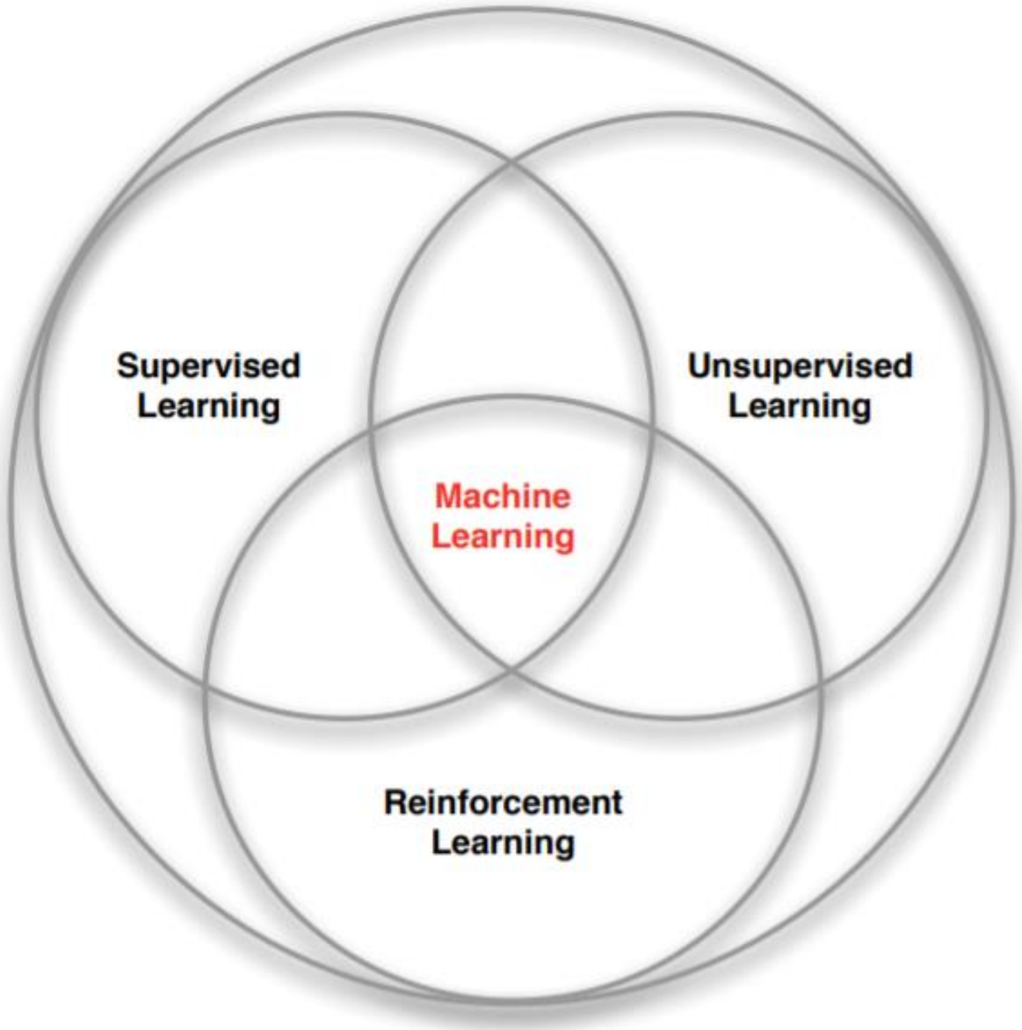
- Markov Process
- Markov Reward Process
- Markov Decision Process

3 Policy Gradient Methods for Reinforcement Learning

4 Reinforcement Learning Example: AlphaGo

5 Summary

Branches of Machine Learning



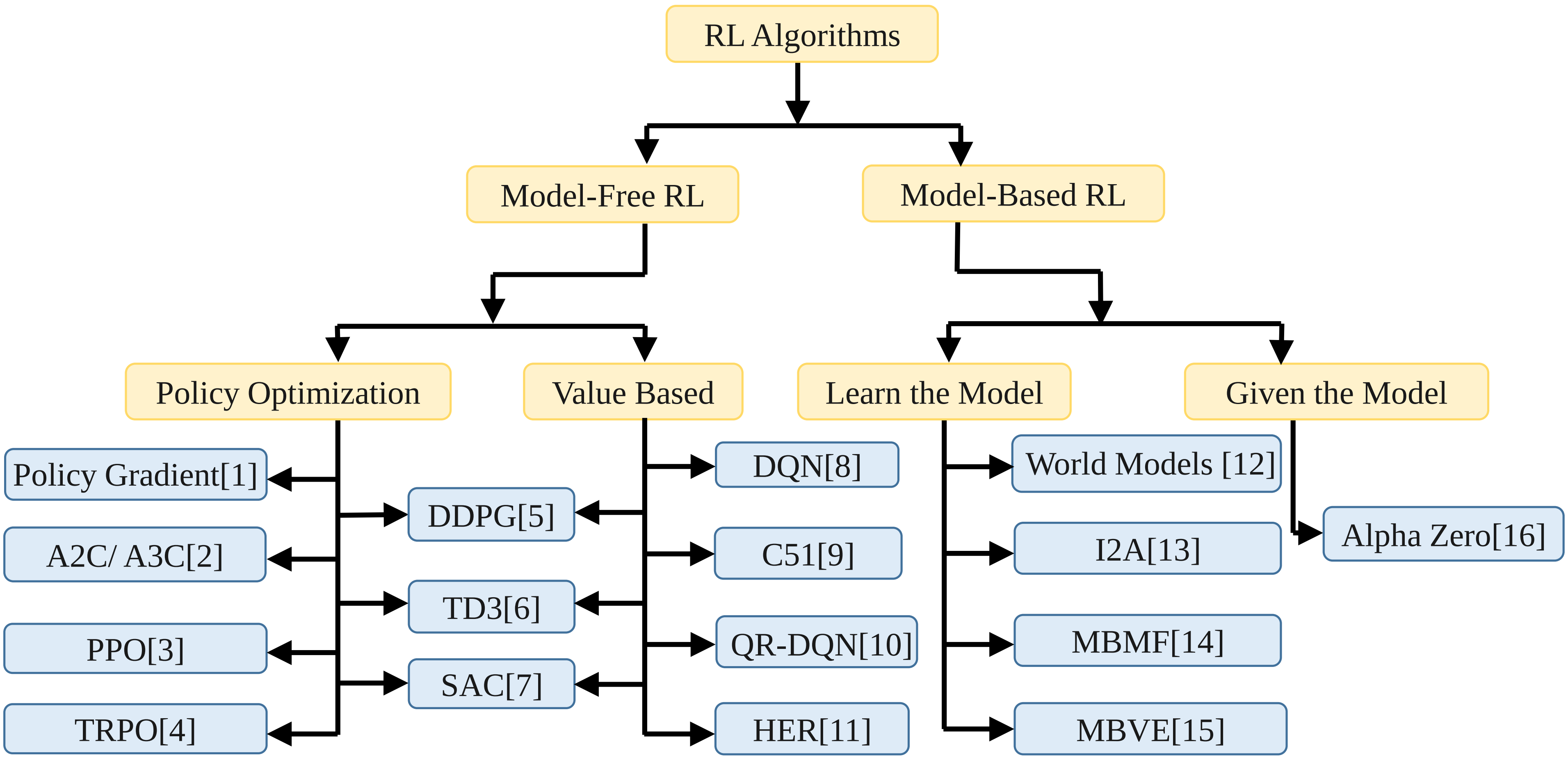
Categorizing of RL Agents (1)

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

Categorizing of RL Agents (2)

- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Policy and/or Value Function
  - Model

Taxonomy of RL Algorithms



Types of RL algorithms

Better Sample Efficient ⟷ Less Sample Efficient

Model-based (100 time steps)	Off-policy Q-learning (1 M time steps)	Actor-critic	On-policy Policy Gradient (10 M time steps)	Evolutionary/ gradient-free (100 M time steps)

Model-based

- Learn the model of the world, then plan using the model
- Update model often
- Re-plan often

Value-based

- Learn the state or state-action value
- Act by choosing best action in state
- Exploration is a necessary add-on

Policy-based

- Learn the stochastic policy function that maps state to action
- Act by sampling policy
- Exploration is baked in

Reference

[1] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." NeurIPS. 2000.

[2] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." ICML. 2016.

[3] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv. 2017.

[4] Schulman, John, et al. "Trust region policy optimization." ICML. 2015.

[5] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." ICLR.2016.

[6] Fujimoto, Scott, Herke Van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." ICLR. 2018.

[7] Haarnoja, Tuomas, et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." ICML. 2018.

[8] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv. 2013.

Reference

[9] Bellemare, Marc G., et al. "A Distributional Perspective on Reinforcement Learning.“ ICML. 2017.

[10] Dabney, Will, et al. "Distributional Reinforcement Learning With Quantile Regression." AAAI. 2018.

[11] Andrychowicz, Marcin, et al. "Hindsight experience replay." NeurIPS. 2017.

[12] Ha, David, et al. "World models." NeurIPS. 2018.

[13] Racanière, Sébastien, et al. "Imagination-augmented agents for deep reinforcement learning." NeurIPS. 2017.

[14] Bansal, Somil, et al. "Mbmf: Model-based priors for model-free reinforcement learning." arXiv. 2017.

[15] Feinberg, Vladimir, et al. "Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning." arXiv. 2018.

[16] Silver, David, et al. "Mastering the game of go without human knowledge." nature. 2017.

Reference

- Silver, D.(2015). UCL Course on RL.
- Levine, S.(2017). CS 294: Deep Reinforcement Learning.
- Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 1998.
- Hung-yi Le. Deep Reinforcement Learning: Scratching the surface, 2016

Q&A