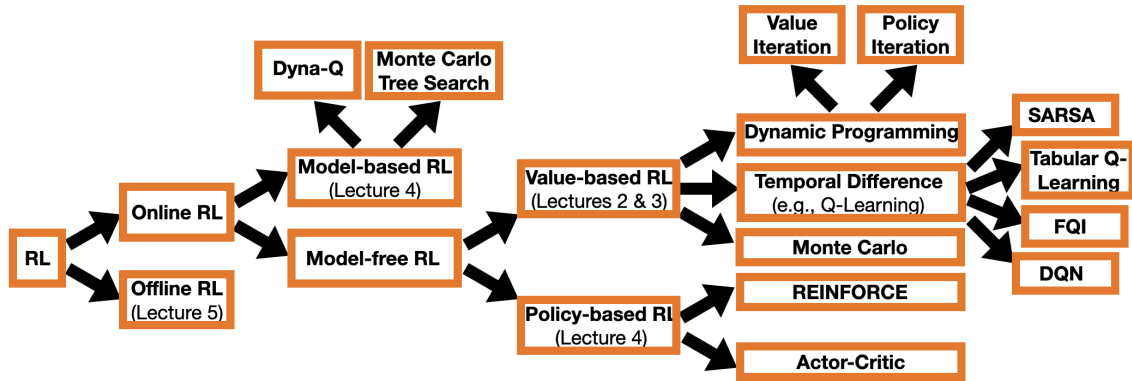


Reinforcement Learning

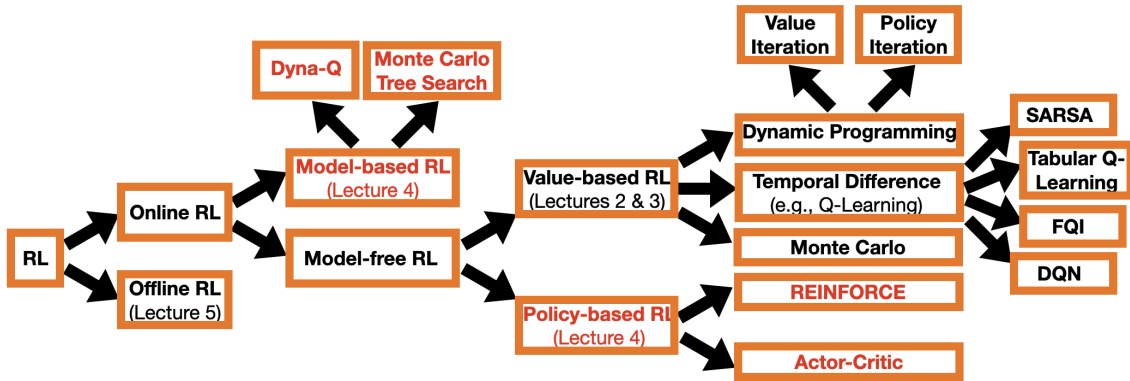
Lecture 4: Policy- and Model-based Learning

Chengchun Shi

Roadmap



Roadmap (Cont'd)



Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

Policy We Studied So Far

- Greedy policy:

$$\pi^{\text{opt}}(\textcolor{blue}{s}) = \arg \max_{\textcolor{red}{a}} Q^{\pi^{\text{opt}}}(\textcolor{blue}{s}, \textcolor{red}{a})$$

- ϵ -Greedy policy:

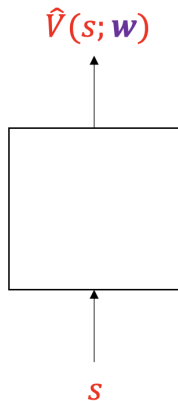
$$\begin{cases} \pi^{\text{opt}}(\textcolor{blue}{s}), & \text{with probability } \mathbf{1} - \epsilon \\ \text{random action}, & \text{with probability } \epsilon. \end{cases}$$

- **Value-based methods:** Policy Iteration, Value Iteration, Monte Carlo, SARSA, Q-Learning, etc.

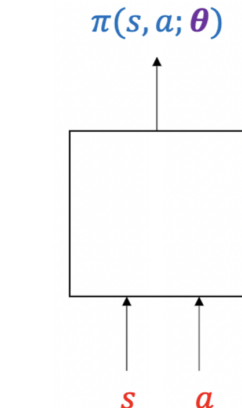
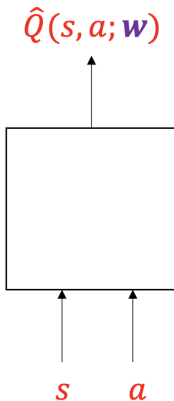
Value-based v.s. Policy-based Methods

- **Value-based methods:** derive π^{opt} by learning an optimal Q-function (or value function)
- **Policy-based methods:** search π^{opt} within a restricted function class (e.g., linear, neural networks) that maximizes the value

Value-based v.s. Policy-based Methods (Cont'd)



Value-based Methods



Policy-based Methods

Example: Linear Function Approximation

- Linear approximation of features $\phi(\mathbf{s}, \mathbf{a})$
- State-action value function approximation

$$Q(\mathbf{s}, \mathbf{a}; \theta) = \phi^\top(\mathbf{s}, \mathbf{a})\theta$$

- Policy function approximation

$$\pi(\mathbf{s}, \mathbf{a}; \theta) = \frac{\exp(\phi^\top(\mathbf{s}, \mathbf{a})\theta)}{\sum_{\mathbf{a}'} \exp(\phi^\top(\mathbf{s}, \mathbf{a}')\theta)}$$

Value-based v.s. Policy-based Methods (Cont'd)

- **Pros** of policy-based methods:
 1. Suitable for learning general **stochastic** policies (value-based methods mainly designed for deterministic policies)
 2. More **robust** to model misspecification
 3. Scalable for **high-dimensional** or **continuous** action spaces (SARSA, Q-learning mainly designed for discrete action space)
- **Cons** of policy-based methods:
 1. Convergence to local minima
 2. May have large variance

Example I: Advantage of Stochastic Policy



- Two-player game of rock-paper-scissors
 - Scissors beats paper
 - Rock beats scissors
 - Paper beats rock
- Consider iterated rock-paper-scissors
 - A deterministic policy is easily exploited
 - A uniform random policy is optimal (Nash equilibrium)

Example II: Robustness of Policy-based Method

- Q-function is more **difficult** to model compared to the optimal policy
- Example: optimal Q-function: $Q^{\pi^{\text{opt}}}(\mathbf{s}, \mathbf{a}) = g(\phi^\top(\mathbf{s}, \mathbf{a})\theta^*)$ for some monotonically increasing function $g : \mathbb{R} \rightarrow \mathbb{R}$
- When g is not **linear** function, value-based method misspecifies Q-function model

$$g(\phi^\top(\mathbf{s}, \mathbf{a})\theta^*) \neq \phi^\top(\mathbf{s}, \mathbf{a})\theta$$

- However, since g is a monotonically increasing function

$$\pi^{\text{opt}}(\mathbf{s}) = \arg \max_{\mathbf{a}} g(\phi^\top(\mathbf{s}, \mathbf{a})\theta^*) = \arg \max_{\mathbf{a}} \phi^\top(\mathbf{s}, \mathbf{a})\theta^*$$

- Policy-based methods correctly specify the optimal policy

$$\frac{\exp(\phi^\top(\mathbf{s}, \mathbf{a})\theta)}{\sum_{\mathbf{a}'} \exp(\phi^\top(\mathbf{s}, \mathbf{a}')\theta)} \rightarrow \mathbb{I}(\mathbf{a} = \pi^{\text{opt}}(\mathbf{s}))$$

when $\theta = k\theta^*$ and $k \rightarrow \infty$

Policy Objective Functions

- Average rewards:

$$J(\theta) = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}^{\pi(\bullet; \theta)} \left[\sum_{t=0}^{T-1} R_t \right] = \sum_{s, a} \nu^{\pi(\bullet; \theta)}(s) \pi(s, a; \theta) \mathcal{R}_s^a$$

where $\mathcal{R}_s^a = \mathbb{E}(R_t | A_t = a, S_t = s)$

- For each π , the states $\{S_t\}_t$ forms a time-homogeneous Markov chain
- $\nu^{\pi(\bullet; \theta)}$ the stationary distribution of $\{S_t\}_t$ under $\pi(\bullet; \theta)$

Policy Objective Functions (Cont'd)

- Discounted rewards: given a discounted factor $\gamma \in [0, 1]$ and initial state distribution ν , maximize the expected discounted rewards:

$$J(\theta) = \mathbb{E}^{\pi(\bullet; \theta)} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right],$$

or equivalently,

$$J(\theta) = \sum_{\mathbf{s}} \nu(\mathbf{s}) V^{\pi(\bullet; \theta)}(\mathbf{s})$$

- If $\gamma = 1$, the task is assumed to be episodic

Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

Policy Gradient

- **Objective:** identify the maximizer of $J(\theta)$
- **Method:** apply (stochastic) gradient ascent algorithm to update θ (gradient descent to minimize $-J(\theta)$)

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_{\theta} J(\theta_t)$$

Need to calculate the gradient $\nabla_{\theta} J(\theta)$!

Policy Gradient Theorem

Theorem

For any differentiable policy $\pi(\mathbf{s}, \mathbf{a}; \theta)$ with respect to parameter θ , the policy gradient for average reward and discounted expected rewards objective is

$$\nabla_{\theta} J(\theta) = \sum_{\mathbf{s}, \mathbf{a}} \mu^{\pi(\bullet; \theta)}(\mathbf{s}, \mathbf{a}) \nabla_{\theta} \log(\pi(\mathbf{s}, \mathbf{a}; \theta)) Q^{\pi(\bullet; \theta)}(\mathbf{s}, \mathbf{a})$$

- For average reward objective:
 $\mu^{\pi(\bullet; \theta)}$ is the stationary distribution of $\{(\mathbf{S}_t, \mathbf{A}_t)\}_t$ under $\pi(\bullet; \theta)$
- For discounted expected rewards objective:

$$\mu^{\pi(\bullet; \theta)}(\mathbf{s}, \mathbf{a}) = \sum_{t \geq 0} \gamma^t \Pr^{\pi(\bullet; \theta)}(\mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a})$$

Discounted state-action visitation probability

Policy Gradient Theorem (Cont'd)

Theorem

For any differentiable policy $\pi(\mathbf{s}, \mathbf{a}; \theta)$ with respect to parameter θ , the policy gradient for average reward and discounted expected rewards objective is

$$\nabla_{\theta} J(\theta) = \sum_{\mathbf{s}, \mathbf{a}} \mu^{\pi(\cdot; \theta)}(\mathbf{s}, \mathbf{a}) \nabla_{\theta} \log(\pi(\mathbf{s}, \mathbf{a}; \theta)) Q^{\pi(\cdot; \theta)}(\mathbf{s}, \mathbf{a})$$

- For average reward objective:

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \mathbb{E}^{\pi} \left[\sum_{t \geq 0} (\mathbf{R}_t - J(\theta)) \mid \mathbf{S}_0 = \mathbf{s}, \mathbf{A}_0 = \mathbf{a} \right]$$

- For discounted expected rewards objective: Q-function defined as usual.
- Proof given in the appendix

Policy Score

- For any state-action pair (\mathbf{s}, \mathbf{a}), the term

$$\nabla_{\theta} \log(\pi(\mathbf{s}, \mathbf{a}; \theta))$$

is referred as the **policy score**

Example 1: Softmax Policy Gradient

- State-action pairs weighted by linear combination of features

$$\pi(\mathbf{s}, \mathbf{a}; \theta) = \frac{\exp(\phi^\top(\mathbf{s}, \mathbf{a})\theta)}{\sum_{\mathbf{a}'} \exp(\phi^\top(\mathbf{s}, \mathbf{a}')\theta)}$$

- The score function

$$\nabla_{\theta} \log \pi(\mathbf{s}, \mathbf{a}; \theta) = \phi(\mathbf{s}, \mathbf{a}) - \frac{\sum_{\mathbf{a}'} \phi(\mathbf{s}, \mathbf{a}') \exp(\phi^\top(\mathbf{s}, \mathbf{a}')\theta)}{\sum_{\mathbf{a}'} \exp(\phi^\top(\mathbf{s}, \mathbf{a}')\theta)}$$

or equivalently,

$$\nabla_{\theta} \log \pi(\mathbf{s}, \mathbf{a}; \theta) = \phi(\mathbf{s}, \mathbf{a}) - \mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{s}, \bullet; \theta)} \phi(\mathbf{s}, \mathbf{a}')$$

Example 2: Continuous Action Space

- Action space: set of real numbers $\mathcal{A} = \mathbb{R}$
- Policy approximator:

$$\pi(\mathbf{s}, \mathbf{a}, \theta) = \frac{1}{\sqrt{2\pi}\sigma(\mathbf{s}; \theta)} \exp\left(-\frac{(\mathbf{a} - \mu(\mathbf{s}; \theta))^2}{2\sigma^2(\mathbf{s}; \theta)}\right),$$

where μ and σ are mean and deviation function approximators

- Linear function approximator with feature vectors $\phi_\mu(\mathbf{s})$ and $\phi_\sigma(\mathbf{s})$
 - $\mu(\mathbf{s}; \theta) = \phi_\mu^\top(\mathbf{s})\theta_\mu$ and $\sigma(\mathbf{s}; \theta) = \phi_\sigma^\top(\mathbf{s})\theta_\sigma$
 - $\nabla_{\theta_\mu} \log \pi(\mathbf{s}, \mathbf{a}, \theta) = \frac{\mathbf{a} - \mu(\mathbf{s}; \theta)}{\sigma^2(\mathbf{s}; \theta)} \phi_\mu(\mathbf{s})$
 - $\nabla_{\theta_\sigma} \log \pi(\mathbf{s}, \mathbf{a}, \theta) = \frac{(\mathbf{a} - \mu(\mathbf{s}; \theta))^2 - \sigma^2(\mathbf{s}; \theta)}{\sigma^2(\mathbf{s}; \theta)} \phi_\sigma(\mathbf{s})$

Example 3: Bernoulli, Logistic Example

- Actions space: binary, $\{0, 1\}$
- Policy approximator:

$$\pi(1, \mathbf{s}; \theta) = 1 - \pi(0, \mathbf{s}; \theta) = \mathbf{p}(\mathbf{s}; \theta)$$

where $\mathbf{p}(\mathbf{s}; \theta)$ is a function approximator

- Linear function approximator with feature vectors $\phi(\mathbf{s})$
 - Logistic function $\sigma(\mathbf{x}) = [\mathbf{1} + \exp(-\mathbf{x})]^{-1}$
 - For exponential soft-max policy $\mathbf{p}(\mathbf{s}; \theta) = \sigma(\phi^\top(\mathbf{s})\theta)$
 - $\nabla_\theta \log(\pi(\mathbf{s}, \mathbf{a}; \theta)) = (\mathbf{a} - \sigma(\phi^\top(\mathbf{s})\theta))\phi(\mathbf{s})$

Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

REINFORCE: MC Policy Gradient Algorithm

- To maximize $J(\theta)$, we apply (stochastic) gradient ascent algorithm

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_{\theta} J(\theta_t)$$

- According to the policy gradient theorem,

$$\nabla_{\theta} J(\theta) = \sum_{\mathbf{s}, \mathbf{a}} \mu^{\pi(\cdot; \theta)}(\mathbf{s}, \mathbf{a}) \nabla_{\theta} \log(\pi(\mathbf{s}, \mathbf{a}; \theta)) Q^{\pi(\cdot; \theta)}(\mathbf{s}, \mathbf{a})$$

- Focus on the average reward setting
- μ^{π} (stationary state-action distribution) is unknown: use empirical state-action distribution $\{(\mathbf{S}_t, \mathbf{A}_t)\}_t$ as an approx
- Q^{π} is unknown: use empirical return $\mathbf{G}_t = \sum_{j=t}^T \mathbf{R}_j$ as an approx

REINFORCE: Pseudocode

- **Initialization:** θ arbitrary
- **For each** episode $(\mathbf{S}_0, \mathbf{A}_0, \mathbf{R}_0, \dots, \mathbf{S}_T, \mathbf{A}_T, \mathbf{R}_T)$ generated using policy $\pi(\bullet; \theta)$

For $t = 0, 1, 2, \dots, T$ **do:**

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t; \theta)) \mathbf{G}_t$$

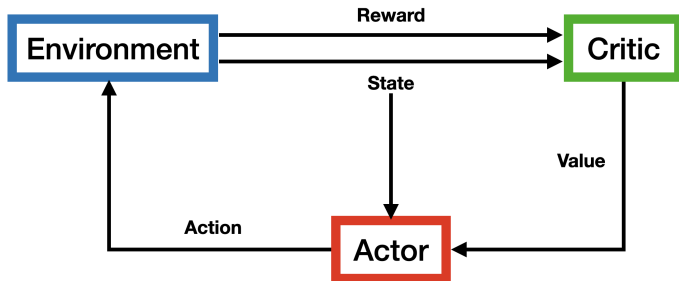
end for

return θ

Actor-Critic Algorithm

- MC policy gradient algorithm may have a large **variance**
 - Return involves many state transitions, many actions and many rewards
- Solution sought by using **actor-critic algorithms**
- Actor-critic algorithms combine **policy gradient** with **value function estimation**

Actor-Critic Algorithm (Cont'd)



- **Critic** uses function approximator to learn value function
- **Actor** uses policy approximator to learn optimal policy

Actor-Critic Control

- **Critic:** estimates $Q^{\pi(\cdot;\theta)}(\mathbf{s}, \mathbf{a})$ by a function approximator $\hat{Q}(\mathbf{s}, \mathbf{a}; \omega)$
 - The critic performs **policy evaluation**
 - Standard methods can be applied: MC, TD
- **Actor:** updates policy parameter θ
 - The actor performs control using approximate policy gradient

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \mu} \nabla_{\theta} \log(\pi(\mathbf{s}, \mathbf{a}; \theta)) Q^{\pi}(\mathbf{s}, \mathbf{a}; \omega)$$

- Parameter update
 - Average reward setting

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t; \theta)) \hat{Q}(\mathbf{S}_t, \mathbf{A}_t; \omega)$$

- Discounted reward setting

$$\theta \leftarrow \theta + \alpha \gamma^t \nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t; \theta)) \hat{Q}(\mathbf{S}_t, \mathbf{A}_t; \omega)$$

Example: Actor-Critic with Linear Value Function

- Linear value function approximator

$$\hat{Q}(\mathbf{s}, \mathbf{a}; \boldsymbol{\omega}) = \boldsymbol{\phi}^\top(\mathbf{s}, \mathbf{a})\boldsymbol{\omega}$$

- Focus on the discounted reward setting
- **Critic:** updates $\boldsymbol{\omega}$ by linear TD

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \eta \boldsymbol{\phi}(\mathbf{S}_t, \mathbf{A}_t)(R_t + \gamma \boldsymbol{\phi}^\top(\mathbf{S}_{t+1}, \mathbf{A}_{t+1})\boldsymbol{\omega}_t - \boldsymbol{\phi}^\top(\mathbf{S}_t, \mathbf{A}_t)\boldsymbol{\omega}_t)$$

Pseudocode

- **Initialization:** s , θ , ω

- **For each** episode:

Initialize $t = 0$

Sample action a from $\pi(\bullet, s; \theta)$

Repeat until s is terminal

Receive reward r and next state s'

Sample action a' from $\pi(\bullet, s; \theta)$

$$\theta \leftarrow \theta + \alpha \gamma^t \log(\pi(s, a; \theta)) \phi^\top(s, a) \omega$$

$$\omega \leftarrow \omega + \eta \phi(s, a) [r + \gamma \phi^\top(s', a') \omega - \phi^\top(s, a) \omega]$$

$a \leftarrow a'$ and $s \leftarrow s'$

$$t \leftarrow t + 1$$

Bias-Variance Tradeoff

- **REINFORCE** uses Return G_t , an unbiased estimate of $Q^{\pi(\cdot;\theta)}(s, a)$
- **Actor-critic** uses $\hat{Q}(s, a; \omega)$, a biased estimate of $Q^{\pi(\cdot;\theta)}(s, a)$
- REINFORCE gradient has **high variance** and **zero bias**
- Actor-critic gradient has **low variance** and **some bias**
- Similar to Pros & Cons of MC vs TD (Lecture 2)

Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

Variance Reduction Using a Baseline

- Recall that policy parameter update

$$\theta \leftarrow \theta + \alpha \gamma^t \nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t; \theta)) \hat{Q}(\mathbf{S}_t, \mathbf{A}_t; \omega)$$

- For any θ , when $\mathbf{A}_t \sim \pi(\mathbf{S}_t, \bullet, \theta)$

$$\mathbb{E}[\nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t, \theta)) | \mathbf{S}_t] = \mathbf{0}$$

- For any baseline function $B(\mathbf{s})$, consider the update

$$\theta \leftarrow \theta + \alpha \gamma^t \nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t; \theta)) [\hat{Q}(\mathbf{S}_t, \mathbf{A}_t; \omega) - B(\mathbf{S}_t)]$$

- The **mean** of gradient is the same without baseline
- However, the **variance** of the gradient would be smaller with a properly chosen B

Variance Reduction Using a Baseline (Cont'd)

- Consider the baseline that minimizes the variance of the gradient
- For any random variable \mathbf{Z} , the mean $\mathbb{E}\mathbf{Z}$ minimizes $\arg \min_{\mathbf{z}} \mathbb{E}(\mathbf{Z} - \mathbf{z})^2$
- To minimize variance of the gradient $\nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t; \theta))[\hat{\mathbf{Q}}(\mathbf{S}_t, \mathbf{A}_t; \omega) - \mathbf{B}(\mathbf{S}_t)]$, the baseline is set to the conditional mean of Q-function given the state
- i.e., $\mathbf{B}(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{s}, \mathbf{a}; \theta) \hat{\mathbf{Q}}(\mathbf{s}, \mathbf{a}; \omega)$, e.g., the estimated state-value

Policy Gradient Using Advantage Function

- Advantage function: $A^{\pi(\cdot;\theta)}(\mathbf{s}, \mathbf{a}) = Q^{\pi(\cdot;\theta)}(\mathbf{s}, \mathbf{a}) - V^{\pi(\cdot;\theta)}(\mathbf{s})$
- Policy gradient based on advantage function

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \mu^{\pi(\cdot;\theta)}} \nabla_{\theta} \log(\pi(\mathbf{s}, \mathbf{a}; \theta)) A^{\pi(\cdot;\theta)}(\mathbf{s}, \mathbf{a})$$

- The advantage function reduces the variance of policy gradient

An Approach for Estimating Advantage Function

- The critic may compute estimators of both value functions

$$\hat{Q}(\mathbf{s}, \mathbf{a}; \omega) \text{ for } Q^{\pi(\cdot; \theta)}(\mathbf{s}, \mathbf{a})$$

and

$$\hat{V}(\mathbf{s}; \omega) \text{ for } V^{\pi(\cdot; \theta)}(\mathbf{s})$$

which can be done by standard methods such as TD learning

- The estimator of the advantage function

$$\hat{A}(\mathbf{s}, \mathbf{a}; \omega) = \hat{Q}(\mathbf{s}, \mathbf{a}; \omega) - \hat{V}(\mathbf{s}; \omega)$$

Another Approach

- $r + \gamma V^{\pi(\cdot;\theta)}(s') - V^{\pi(\cdot;\theta)}(s)$ is unbiased to $A^{\pi(\cdot;\theta)}(s, a)$

$$\begin{aligned} & \mathbb{E}[r + \gamma V^{\pi(\cdot;\theta)}(s') - V^{\pi(\cdot;\theta)}(s) | a, s] \\ &= \mathbb{E}[r + \gamma V^{\pi(\cdot;\theta)}(s') - Q^{\pi(\cdot;\theta)}(s, a) + Q^{\pi(\cdot;\theta)}(s, a) - V^{\pi(\cdot;\theta)}(s) | a, s] \\ &= Q^{\pi(\cdot;\theta)}(s, a) - V^{\pi(\cdot;\theta)}(s) = A^{\pi(\cdot;\theta)}(s, a) \end{aligned}$$

- As such,

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{(s,a) \sim \mu^{\pi(\cdot;\theta)}} \nabla_{\theta} \log(\pi(s, a; \theta)) [r + \gamma V^{\pi(\cdot;\theta)}(s') - V^{\pi(\cdot;\theta)}(s)]$$

- No need to estimate the advantage. It suffices to estimate the state-value and use the estimator to compute the policy gradient

Advantage Policy Gradient Methods

- The policy gradient

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \mu^{\pi(\cdot; \theta)}} \nabla_{\theta} \log(\pi(\mathbf{s}, \mathbf{a}; \theta)) \mathbf{A}^{\pi(\cdot; \theta)}(\mathbf{s}, \mathbf{a})$$

- Gradient-based method

$$\theta \leftarrow \theta + \alpha \gamma^t \nabla_{\theta} \log(\pi(\mathbf{S}_t, \mathbf{A}_t; \theta)) \hat{\mathbf{A}}(\mathbf{S}_t, \mathbf{A}_t; \omega)$$

- Examples:

- MC: $\hat{\mathbf{A}}(\mathbf{S}_t, \mathbf{A}_t; \omega) = \mathbf{G}_t - \hat{\mathbf{V}}(\mathbf{S}_t; \omega)$
- TD: $\hat{\mathbf{A}}(\mathbf{S}_t, \mathbf{A}_t; \omega) = \mathbf{R}_t + \gamma \hat{\mathbf{V}}(\mathbf{S}_{t+1}; \omega) - \hat{\mathbf{V}}(\mathbf{S}_t; \omega)$

Summary

Policy Function Approximation

		No	Yes
Value Function Approximation	No	Value-based (tabular)	REINFORCE
	Yes	Value-based	Actor-Critic

- **Value-based**

- SARSA
- Tabular Q-learning
- Fitted Q-iteration
- Deep Q-network

- **REINFORCE**

- No value function
- Learn policy

- **Actor-critic**

- Learn value
- Learn policy

- **Advantage actor-critic**

- Variance reduction

Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

Recap: Planning vs Learning

Two fundamental problems in sequential decision making

- **Planning**

- A model of the environment (e.g., state transition, reward function) is **known**
- The agent performs computations with its model, **without** any external interaction
- a.k.a. deliberation, reasoning, introspection, pondering, thought, search

- **Learning**

- The environment is initially **unknown**
- The agent **interacts** with the environment
- The agent **learns** the optimal policy from experience

RL Algorithms We Have Covered So Far

- **Dynamic Programming** (Lecture 2): learn **value** from **model** (planning)
- **MC, TD** (Lectures 2 & 3): learn **value** from **experience** (learning)
- **Policy-based** (Lecture 4): learn **policy** from **experience** (learning)
- Today's lecture: **Model-based** RL
 - learn **model** from experience
 - use both learned model and experience to construct a **value** function or **policy**
 - combine learning with planning

What is a Model?

- A model \mathcal{M} is a **representation** of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
- The state space \mathcal{S} and action space \mathcal{A} are usually known to us
- The discounted factor γ is **user-specified**
- Only need to learn the state transition \mathcal{P}

$$\mathcal{P}_{ss'}^a = \Pr(\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s, \mathcal{A}_t = a)$$

and reward function \mathcal{R}

$$\mathcal{R}_s^a = \mathbb{E}(\mathcal{R}_t | \mathcal{S}_t = s, \mathcal{A}_t = a)$$

Model-Free v.s. Model-Based RL

- Model-based RL
 - Learn the model (e.g., reward \mathcal{R}_s^a and transition $\mathcal{P}_{ss'}^a$) from experience
 - **Plan** value or policy from model or **integrate** planning with learning
- Model-free RL
 - **Learn** value or policy **without** learning the reward and transition function
 - Rely on Bellman optimality equation
 - Examples: MC, TD, Policy gradient

Model-Free v.s. Model-Based RL (Cont'd)

- **Pros** of model-based RL
- In some applications, we have a **perfect** model (e.g., Go, chess)
- Can handle **offline** data (more in the next lecture)

- **Pros** of model-free RL
- **Dimensional reduction**
- Easier to learn value than model
- # of parameters of $Q^{\pi^{\text{opt}}}$: $|\mathcal{S}||\mathcal{A}|$
- # of parameters of \mathcal{R}_s^a : $|\mathcal{S}||\mathcal{A}|$
- # of parameters of $\mathcal{P}_{ss'}^a$: $|\mathcal{S}|^2|\mathcal{A}|$

Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Model Learning

- **Goal:** estimate \mathcal{R}_s^a and $\mathcal{P}_{ss'}^a$ from experience $\{S_0, A_0, R_0, \dots, S_T\}$
- Using supervised learning

$$\begin{aligned} S_0, A_0 &\rightarrow R_0, S_1 \\ S_1, A_1 &\rightarrow R_1, S_2 \\ &\vdots \\ S_{T-1}, A_{T-1} &\rightarrow R_{T-1}, S_T \end{aligned}$$

- Learning $s, a \rightarrow r$ is a **regression** problem
- Learning $s, a \rightarrow s'$ is a **conditional density estimation** problem
- Loss function: least square/Huber loss, KL divergence
- Compute parameter that minimizes empirical loss

Models for Conditional Density Estimation

- Table lookup model
- Conditional kernel density estimation
- Gaussian process model [Williams and Rasmussen, 2006]
- Deep conditional generative learning¹
 - mixture density network [Rothfuss et al., 2019]
 - normalising flows [Trippe and Turner, 2018]

¹<https://deepgenerativemodels.github.io/notes/index.html>

Table Lookup Model

- Finite MDP model
- Count visits $N(\mathbf{s}, \mathbf{a}) = \sum_{t=0}^{T-1} \mathbb{I}(\mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a})$ to each state-action pair

$$\hat{\mathcal{P}}_{ss'}^{\mathbf{a}} = \frac{1}{N(\mathbf{s}, \mathbf{a})} \sum_{t=0}^{T-1} \mathbb{I}(\mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a}, \mathbf{S}_{t+1} = \mathbf{s}')$$
$$\hat{\mathcal{R}}_{\mathbf{s}}^{\mathbf{a}} = \frac{1}{N(\mathbf{s}, \mathbf{a})} \sum_{t=0}^{T-1} \mathbb{I}(\mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a}) R_t$$

- Alternatively
 - At each time step t , record experience tuple $\langle \mathbf{S}_t, \mathbf{A}_t, R_t, \mathbf{S}_{t+1} \rangle$
 - To sample model, based on a state-action pair (\mathbf{s}, \mathbf{a}) , randomly pick tuple matching $\langle \mathbf{s}, \mathbf{a}, \bullet, \bullet \rangle$

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Planning with Dynamic Programming

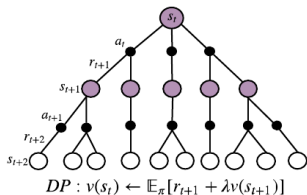
- Give a model $\langle \hat{\mathcal{R}}, \hat{\mathcal{P}} \rangle$
- Use dynamic programming algorithm
 - Policy iteration

$$\pi_0 \xrightarrow{\text{red}} V^{\pi_0} \xrightarrow{\text{blue}} \pi_1 \xrightarrow{\text{red}} V^{\pi_1} \xrightarrow{\text{blue}} \dots \xrightarrow{\text{blue}} \pi^{\text{opt}} \xrightarrow{\text{red}} V^{\pi^{\text{opt}}}$$

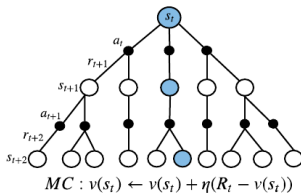
- Value iteration

$$V^{\pi_0} \xrightarrow{\text{red}} V^{\pi_1} \xrightarrow{\text{red}} V^{\pi_2} \xrightarrow{\text{red}} \dots \xrightarrow{\text{red}} V^{\pi^{\text{opt}}} \xrightarrow{\text{red}} \pi^{\text{opt}}$$

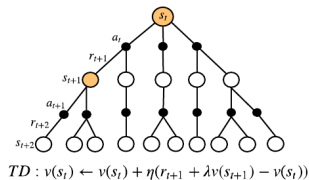
Difference From Model-Free Methods



Dynamic Programming (DP)



Monte Carlo (MC)



Temporal Difference (TD)

Planning with Model-Free RL

- A simple but powerful approach to planning
- Use the model only to **generate samples**
- **Sample** experience from model:

$$S' \sim \hat{\mathcal{P}}_{S,\bullet}^A \quad \text{and} \quad R = \hat{\mathcal{R}}_S^A$$

- Apply **model-free** RL to samples
 - deep Q-network
 - fitted Q-iteration
 - actor-critic
- This is often more **efficient** than dynamic programming-based method

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Real and Simulated Experience

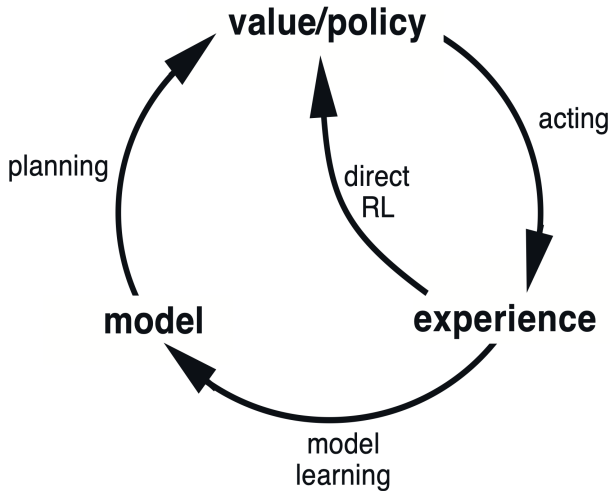
- We consider two sources of experience
- **Real experience:** Sampled from environment (true MDP)

$$\{S_0, A_0, R_0, \dots, S_T\}$$

- **Simulated experience:** Sampled from model (estimated MDP)

$$S' \sim \hat{\mathcal{P}}_{S,\bullet}^A \quad \text{and} \quad R = \hat{\mathcal{R}}_S^A$$

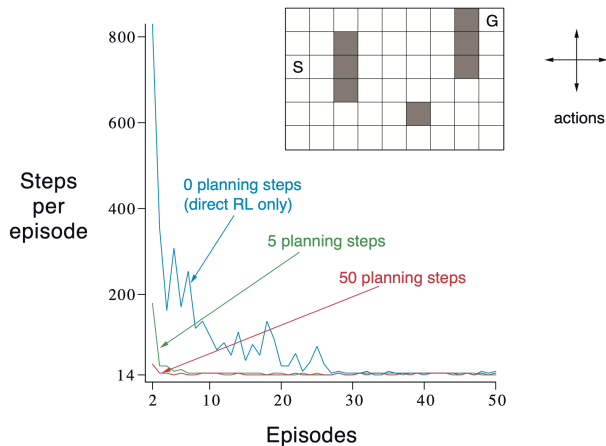
Dyna



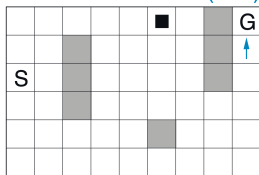
Dyna-Q Algorithm

- **Initialize** $Q(s, a)$ and **model**(s, a) for all s and a
- **do** forever:
 - (a) $s \leftarrow$ current (non-terminal) state
 - (b) $a \leftarrow \epsilon$ -greedy(s, Q)
 - (c) Execute action a ; observe reward r and next state s'
 - (d) $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
 - (e) **model**(s, a) $\leftarrow (r, s')$
 - (f) Repeat n times:
 - $s \leftarrow$ random previously observed state
 - $a \leftarrow$ random action previously taking in s
 - $(r, s') \sim \mathbf{model}(s, a)$
 - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

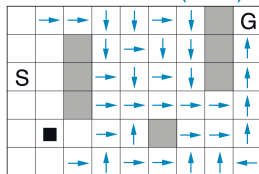
Dyna-Q on a Simple Maze



WITHOUT PLANNING ($n=0$)



WITH PLANNING ($n=50$)



- similar to “experience replay” in DQN
- use historical data more efficiently

Figure: Policies found through 2nd episode. The arrows indicate greedy action; if no arrow is shown for a state, then all of its action values were equal.

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Two Ways of Planning

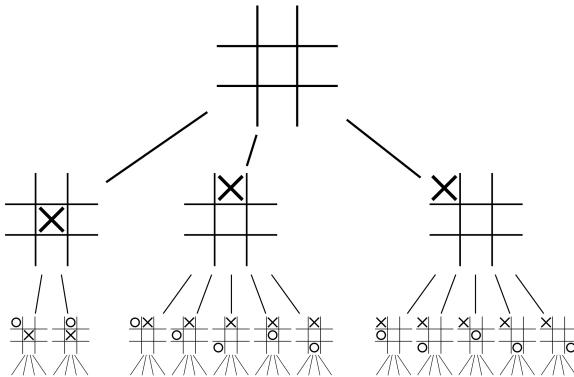
- **Background planning**

- Planning is used well **before** an action is selected
- Need to select actions for each state, not **current** state
- Examples: policy iteration and value iteration in Lecture 2

- **Decision-time planning**

- Planning is started and completed **after** encountering each new state S_t
- As a computation to **determine** A_t
- On the next step planning begins anew with S_{t+1} to produce A_{t+1} , and so on

Game Trees



- Game trees: data structures to represent a game
- Exhaustive search can be **computationally intensive**
- Solutions sought by **Monte Carlo tree search**

Monte-Carlo Tree Search (Evaluation)

- Given a model \mathcal{M}
- Simulate K episodes from current states S_t using current policy π

$$\left\{ S_t, A_t^k, R_t^k, S_{t+1}^k, A_{t+1}^k, R_{t+1}^k, \dots, S_T^k \right\}_{k=1}^K \sim \mathcal{M}, \pi$$

- Build a search tree containing visited states and actions
- Evaluate states $Q(s, a)$ by mean return of episodes from s, a

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^K \sum_{u=t}^T \mathbb{I}(S_u = s, A_u = a) G_u \rightarrow Q^\pi(s, a)$$

- After search is finished, select current action with maximum value in search tree

$$A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$$

Monte-Carlo Tree Search (Simulation)

- In MCTS, the simulation policy (rollout policy) π that simulates data **improves**
- Repeat (each simulation)
 - **Evaluate** states $Q(\mathbf{s}, \mathbf{a})$ by Monte-Carlo evaluation
 - **Improve** simulation policy, e.g., by ϵ -greedy(Q)
 - **Monte-Carlo control** applied to **simulated experience**
- Converges to the optimal search tree, $Q(\mathbf{s}, \mathbf{a}) \rightarrow Q^{\pi^{\text{opt}}}(\mathbf{s}, \mathbf{a})$

Lecture Outline

1. Policy-based Learning

- 1.1 Introduction to Policy-based Learning
- 1.2 Policy Gradient Theorem
- 1.3 REINFORCE and Actor Critic Algorithms
- 1.4 Advantage Actor-Critic (A2C)

2. Model-based Learning

- 2.1 Introduction to Model-based Learning
- 2.2 Model-based Methods
- 2.3 Mastering the Game of Go

Case Study: the Game of Go



- Invented in China over 2500 years ago
- The **hardest** classic board game
- Much harder than chess:
 - Go has larger number of legal moves than chess (≈ 250 v.s. ≈ 35)
 - Go involve more moves than chess (≈ 150 v.s. ≈ 80)
 - Traditional game-tree search fails in Go

Rules of Go

- Two players place down white and black stones **alternately**
- Stones are **captured** according to simple rules

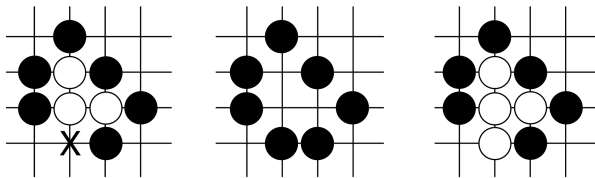
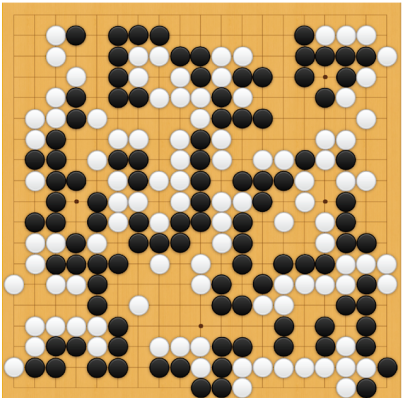


Figure: Left: the three white stones are not surrounded because point X is unoccupied. Middle: if black places a stone on X, the three white stones are captured and removed from the board. Right: if white places a stone on point X first, the capture is blocked.

- The game ends when neither player wishes to place another stone
- The player with more **territory** wins the game


AlphaGo




THE ULTIMATE GO CHALLENGE
GAME 3 OF 3

27 MAY 2017

● vs ●

 **AlphaGo**
Winner of Match 3

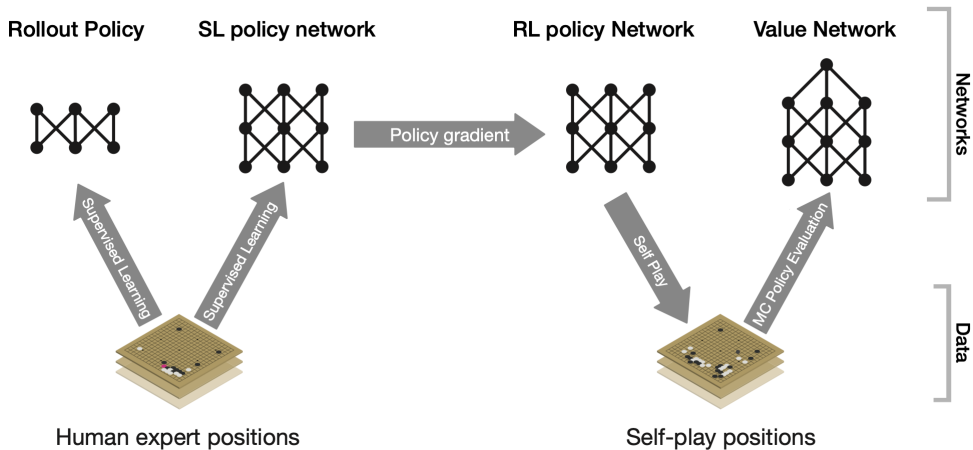
 **Ke Jie**

RESULT B + Res

AlphaGo Pipeline

- Based on a novel version of **Monte-Carlo tree search** (MCTS)
- Combined with a **policy** and a **value function** learned by RL with function approximation provided by deep CNN
- Simulate trajectories and generate the search tree using the **rollout** policy
- **Expand** search tree by selecting unexplored actions according to a **policy network**
- Policy network trained previously via supervised learning to predict moves contained in a database of nearly 30 million human expert moves, and updated via **self-play**
- Evaluate state-action value based on simulated returns (MC) and a **value** network
- Value network trained previously via RL

AlphaGo Pipeline (Cont'd)



Input of Neural Networks

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Policy Network

- Training the **SL policy network** took approximately 3 weeks using distributed implementation of SGD on 50 processors
- The SL policy network achieved **57%** accuracy; best accuracy achieved by other methods **44%**
- The **RL policy network** is trained on a million games in a single day
- The final RL policy won more than **80%** of games played against the SL policy
- It won **85%** of games played against a Go program using MCTS that simulated 100,000 games per move

Value Network

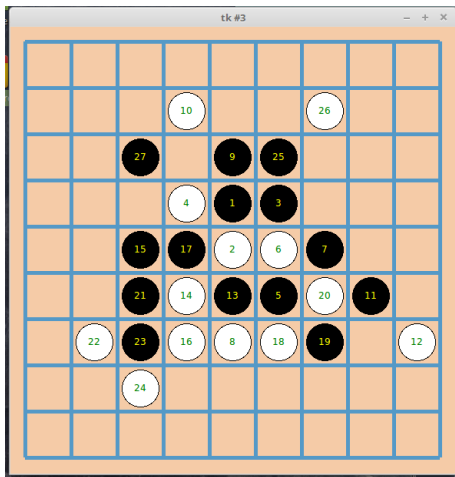
- The **value network** used **Monte Carlo policy evaluation** based on data obtained from a large number of self-play games played using the RL policy
- To avoid overfitting and instability, and to reduce the strong correlations between positions encountered in self-play, the dataset consists of **30** million positions, each chosen randomly from a unique self-play game
- Training was done using **50** million mini-batches each of **32** positions drawn from this data set
- Training took one week on **50** GPUs

Rollout Policy

- The **rollout policy** was learned prior to play by a simple linear network trained by supervised learning from a corpus of **8** million human moves
- In principle, the SL or RL policy networks could have been used in the rollouts, but the forward propagation through these deep networks took **too much time** for either of them to be used in rollout simulations
- The rollout policy network allowed approximately 1,000 complete game simulations per second to be run on each of the processing threads

AlphaGo Zero on Gomoku

https://github.com/initial-h/AlphaZero_Gomoku_MPI



Summary

- Model-based/Model free learning
- Integrating planning and learning
- Dyna-Q
- Background/Decision-time planning
- Monte Carlo Tree Search
- AlphaGo

References I

- Jonas Rothfuss, Fabio Ferreira, Simon Walther, and Maxim Ulrich. Conditional density estimation with neural networks: Best practices and benchmarks. *arXiv preprint arXiv:1903.00954*, 2019.
- Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- Brian L Trippe and Richard E Turner. Conditional density estimation with bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018.
- Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Appendix: Proof of Policy Gradient Theorem

- We focus on the discounted reward setting. Proofs in the average reward setting can be found in Sutton et al. [1999]
- Basic identities

$$(A) \quad V^\pi(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) Q^\pi(\mathbf{s}, \mathbf{a})$$

$$(B) \quad Q^\pi(\mathbf{s}, \mathbf{a}) = R_{\mathbf{s}}^{\mathbf{a}} + \gamma \sum_{\mathbf{s}'} P_{\mathbf{s}, \mathbf{s}'}^{\mathbf{a}} V^\pi(\mathbf{s}')$$

$$(C) \quad \nabla_{\theta} V^\pi(\mathbf{s}) = \sum_{\mathbf{a}} [\nabla_{\theta} \pi(\mathbf{a}|\mathbf{s})] Q^\pi(\mathbf{s}, \mathbf{a}) + \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) [\nabla_{\theta} Q^\pi(\mathbf{s}, \mathbf{a})]$$

$$(D) \quad \nabla_{\theta} Q^\pi(\mathbf{s}, \mathbf{a}) = \gamma \sum_{\mathbf{s}'} P_{\mathbf{s}, \mathbf{s}'}^{\mathbf{a}} \nabla_{\theta} V^\pi(\mathbf{s}')$$

Appendix: Proof (Cont'd)

$$\begin{aligned}
 \nabla_{\theta} \mathbf{V}^{\pi}(\mathbf{s}) &\stackrel{(C)}{=} \sum_{\mathbf{a}} [\nabla_{\theta} \pi(\mathbf{a}|\mathbf{s})] \mathbf{Q}^{\pi}(\mathbf{s}, \mathbf{a}) + \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) [\nabla_{\theta} \mathbf{Q}^{\pi}(\mathbf{s}, \mathbf{a})] \\
 &\stackrel{(D)}{=} \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) [\nabla_{\theta} \log(\pi(\mathbf{a}|\mathbf{s}))] \mathbf{Q}^{\pi}(\mathbf{s}, \mathbf{a}) + \underbrace{\gamma \sum_{\mathbf{a}, \mathbf{s}'} \pi(\mathbf{a}|\mathbf{s}) \mathbf{P}_{\mathbf{s}, \mathbf{s}'}^{\mathbf{a}} \nabla_{\theta} \mathbf{V}^{\pi}(\mathbf{s}')}_{I}
 \end{aligned}$$

Now, consider I . Similarly, we have

$$\begin{aligned}
 I &= \sum_{\mathbf{a}, \mathbf{s}', \mathbf{a}'} \pi(\mathbf{a}|\mathbf{s}) \mathbf{P}_{\mathbf{s}, \mathbf{s}'}^{\mathbf{a}} \pi(\mathbf{a}'|\mathbf{s}') [\nabla_{\theta} \log(\pi(\mathbf{a}'|\mathbf{s}'))] \mathbf{Q}^{\pi}(\mathbf{s}', \mathbf{a}') \\
 &\quad + \gamma \sum_{\mathbf{a}, \mathbf{s}', \mathbf{a}', \mathbf{s}''} \pi(\mathbf{a}|\mathbf{s}) \mathbf{P}_{\mathbf{s}, \mathbf{s}'}^{\mathbf{a}} \pi(\mathbf{a}'|\mathbf{s}') \mathbf{P}_{\mathbf{s}', \mathbf{s}''}^{\mathbf{a}'} \nabla_{\theta} \mathbf{V}^{\pi}(\mathbf{s}'')
 \end{aligned}$$

Appendix: Proof (Cont'd)

Recursively applying the first identity, we obtain

$$\nabla_{\theta} \mathbf{V}^{\pi}(\mathbf{s}) = \mu^{\pi(\bullet; \theta)}(\mathbf{s}', \mathbf{a}'; \mathbf{s}) \nabla_{\theta} \log(\pi(\mathbf{s}', \mathbf{a}')) \mathbf{Q}^{\pi}(\mathbf{s}', \mathbf{a}')$$

where

$$\mu^{\pi(\bullet; \theta)}(\mathbf{s}', \mathbf{a}'; \mathbf{s}) = \sum_{t \geq 0} \gamma^t \pi(\mathbf{s}', \mathbf{a}') \Pr^{\pi(\bullet; \theta)}(\mathbf{S}_t = \mathbf{s}' | \mathbf{S}_0 = \mathbf{s})$$

Questions