Lecture 20
Reinforcement Learning
(Part II)

Lei Li, **Yu-Xiang Wang**

Announcements

1. **Poster printing:** See the poster printing instructions from edstem.

2. **Project Presentation day:** Small prizes will be given to the best poster presentations. You can invite your friends to come！

3. **Course evaluation:** Please complete you ESCI surveys if you haven’t yet. It takes only a few minutes.

Optional HW4

- For practice problems in convex optimization
  - For a simple coding problem with cvx: Q4 of here with data here
  - For practices on convex analysis: Q1,2,3 here

- Theory / concept practices for RL:
  - Problem 3 and 4 here
  - Problem 1 and 2 here
- Coding practice for MDP / RL: Here

- For more advanced problems in RL:
  - see HW1,2,3 from my RL theory course.
  - These are only useful if you are hoping to do RL research.

Recap: Markov Decision processes (infinite horizon / discounted)

- Infinite horizon / discounted setting

$$\mathcal{M}(\mathcal{S}, \mathcal{A}, P, r, \gamma, \mu)$$

Transition kernel: $P: S \times A \to \Delta(S)$ i.e. $P(s'|s,a)$

(Expected) reward function: $r: S \times A \to \mathbb{R} / [0, R_{max}]$ $\mathbb{E}[R_t \mid S_t = s, A_t = a] =: r(s,a)$

Initial state distribution $\mu \in \Delta(S)$

Discounting factor: γ

Stationary Policy π: mapping from state to an action (possibly a random action).

Recap: Value functions

- state value function: $V^{\pi}(s)$
  - expected **long-term** return when starting in *s* and following π

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_1 + \gamma R_2 + ... + \gamma^{t-1} R_t + ...|S_1 = s]$$

- state-action value function: $Q^{\pi}(s,a)$
  - expected **long-term** return when starting in *s*, performing *a,* and following π

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_1 + \gamma R_2 + ... + \gamma^{t-1} R_t + ...|S_1 = s, A_1 = a]$$

Recap: Bellman equations

- Bellman consistency equation

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

$$\underline{V^\pi = r^\pi + \gamma P^\pi V^\pi}$$

- Bellman optimality equation

$$V^*(s) = \underbrace{\max_a}_{} \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma \underbrace{V^*(s')}_{})]$$

$$\boxed{Q^* = r + \gamma P V_{\underset{\uparrow}{Q^*}}}$$ where $$V_Q(s) := \underbrace{\max_{a \in \mathcal{A}} Q(s,a)}_{}.$$

Recap: MDP planning and Value iterations

- MDP planning:

$$\text{Find } \pi^* \text{ such that } V^{\pi}(s) = V^*(s) \quad \forall s$$

$$\pi \text{ is } \epsilon\text{-optimal if } \underline{V^{\pi} \geq V^*(s) - \epsilon \mathbf{1}}$$

- Policy evaluation
  - Solving Bellman consistency equation $\Rightarrow V^{\pi}$

$$\underline{\pi_{next} = \arg\max Q^{\pi}}$$

- Value iteration
  - Solving Bellman optimality equation

Recap: RL agent needs to learn the underlying MDP model

- **Model-based algorithm**
  - Estimates the MDP then do MDP planning

- **Model-free algorithms** Q^* only thing needed
  - Monte Carlo Policy evaluation + Policy improvement
  - Temporal difference learning = MC + Bellman equations

Recap: TD Learning

- TD-Policy evaluation

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[\underbrace{\overset{\checkmark}{R_{t+1}} + \gamma \overset{\text{if } V = V^{\pi}}{\boxed{V(S_{t+1})}} - V(S_t)} \Big]$$

- TD-Policy optimization
  - SARSA (on-policy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha \underbrace{\big[R + \gamma \overset{\checkmark}{Q}(S', A') - Q(S, A) \big]}$$

Then choose the next A’ using Q, e.g., eps-greedy.

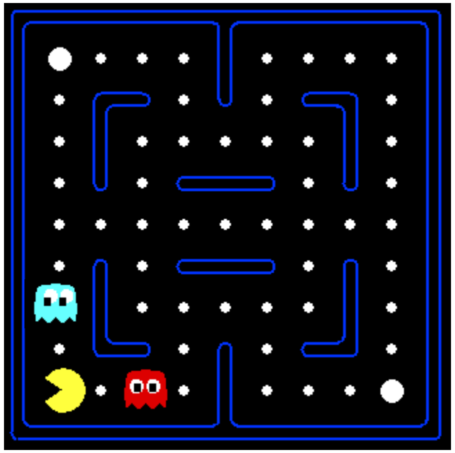
  - Q-Learning (off-policy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha \big[R + \gamma \underbrace{\max_a Q(S', a)} - Q(S, A) \big]$$

Then choose the next action in your favorite way.

Recap: The problem of large-state space

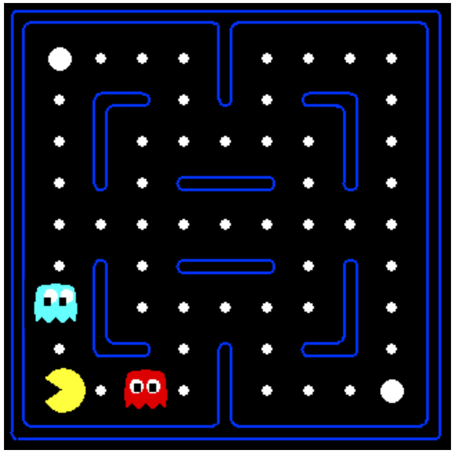
Let's say we discover through experience that this state is bad:



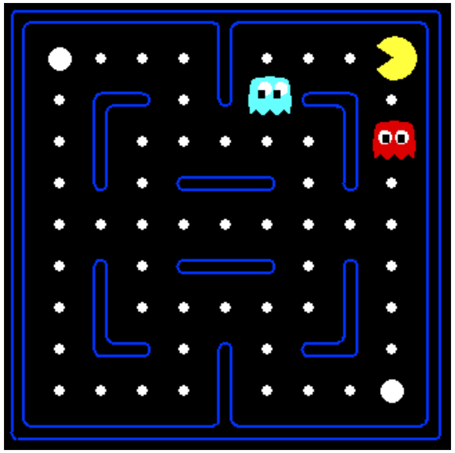
(From Dan Klein and Pieter Abbeel)

Recap: The problem of large-state space

Let’s say we discover through experience that this state is bad:



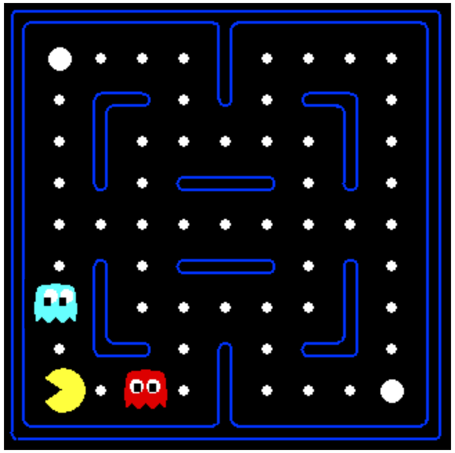
In naïve q-learning, we know nothing about this state:



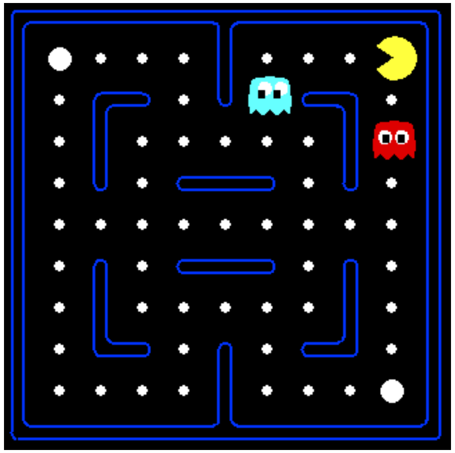
(From Dan Klein and Pieter Abbeel)

Recap: The problem of large-state space

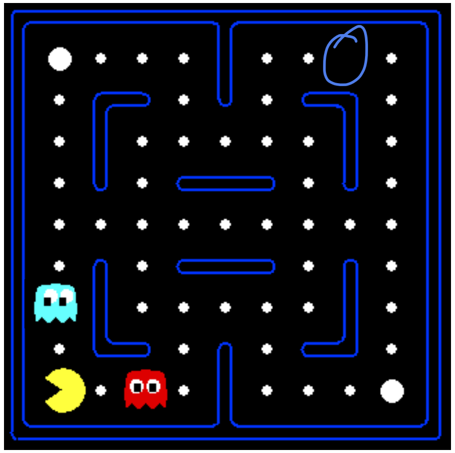
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



Or even this one!



(From Dan Klein and Pieter Abbeel)

This lecture

- Solve the problem of large state space with function approximation

- Other RL algorithms: Policy gradient

- Exploration in RL

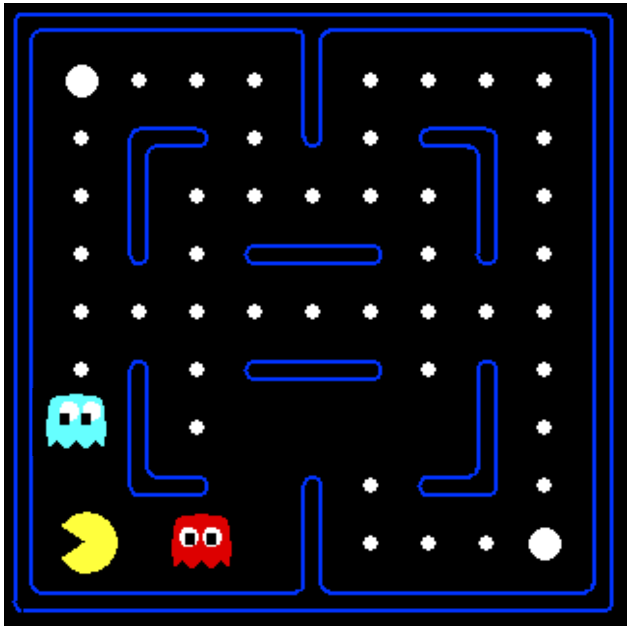
Video of Demo Q-Learning Pacman – Tiny – Watch All

Video of Demo Q-Learning Pacman – Tiny – Silent Train

Video of Demo Q-Learning Pacman – Tricky – Watch All

Why not use an evaluation function? A Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:
  - $V_{\mathbf{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$ $= \langle w, \vec{f}(s) \rangle$
  - $Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$ $= \langle w, \vec{g}(s,a) \rangle$

$g(s,a) = E_{s' \sim P(s,a)} f(s')$

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

Updating a linear value function

Updating a linear value function

- Original Q learning rule tries to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} \underline{Q(s',a')} - Q(s,a)]$$

Updating a linear value function

- Original Q learning rule tries to reduce prediction error at s, a:

$$Q(s,a) \text{ ⍰ } Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

- Instead, we update the weights to try to reduce the error at s, a:

Updating a linear value function

- Original Q learning rule tries to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$\langle w, f(s,a) \rangle \leftarrow \langle w, f(s,a) \rangle + \alpha \left[R + \gamma \max_{a'} \langle w, f(s',a') \rangle - \langle w, f(s,a) \rangle \right)$$

- Instead, we update the weights to try to reduce the error at s, a:

$$w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$$

$$= w_i + \alpha \cdot \underbrace{[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]}_{\uparrow} f_i(s,a)$$

Updating a linear value function

- Original Q learning rule tries to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

- Instead, we update the weights to try to reduce the error at s, a:

$$w_i \longleftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]\, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$$

$$= w_i + \alpha \cdot \underbrace{[R(s,a,s') + \gamma \max_{a'} Q(s',a')}_{} - \underset{\uparrow}{\underline{Q(s,a)}}]\, f_i(s,a)$$

- Qualitative justification:
  - Pleasant surprise: increase weights on positive features, decrease on negative ones
  - Unpleasant surprise: decrease weights on positive features, increase on negative ones

PACMAN Q-Learning (Linear function approx.)

Deriving the TD via incremental optimization that minimizes Bellman errors

$\hat{V}^{\pi}(s,w)$ w: weights

- Mean Square Error and Mean Square Bellman error

$$\min_w \mathbb{E}_{s\sim\mu}\left[\left(V^{\pi}(s) - \hat{V}^{\pi}(s;w)\right)^2\right]$$

Monte Carlo G_i s.t. $\mathbb{E}[G_i] = V^{\pi}(s)$

$$\min_w \sum_{i=1}^{N}\left(G^{s_i} - \hat{V}^{\pi}(s_i,w)\right)^2$$

$$w^{+} = w - \alpha \nabla_w \left(G_i^{s_i} - \hat{V}^{\pi}(s_i,w)\right)^2$$

$$= w - \alpha\left(\hat{V}^{\pi}(s_i,w) - G_i^{s_i}\right)\cdot \nabla_w \hat{V}^{\pi}(s_i,w)$$

$$= w + \alpha\left(G_i^{s_i} - \hat{V}^{\pi}(s_i,w)\right)\nabla_w \hat{V}^{\pi}(s_i,w)$$

SGD

$$\min_w \mathbb{E}_{s\sim\mu}\left[\left(r^{\pi}(s) + \gamma\, \mathbb{E}_{\substack{s'\sim P(s,a)\\ a\sim\pi(s)}} \hat{V}^{\pi}(s';w) - \hat{V}^{\pi}(s,w)\right)^2\right]$$

Sample $s_i \sim \mu$, $s_i' \sim P(s_i, a_i)$
R_i $a_i \sim \pi(s_i)$.

$$\min \frac{1}{n}\sum_{i=1}^{n}\left(R_i + \gamma \underbrace{\hat{V}^{\pi}(s_i';w)}_{\text{fix,}} - \underbrace{\hat{V}(s_i,w)}_{\nabla \text{ only}}\right)^2$$

$$w^{+} = w - \alpha \nabla_w \left(\underbrace{R_i + \gamma V}\, - \hat{V}(s_i,w)\right)^2$$

$$= w + \alpha\left(R_i + \gamma V - \hat{V}(s_i,w)\right)\cdot \nabla \hat{V}(s_i,w)$$

"Semigradient update"

So far, in RL algorithms

- Model-based approaches
  - Estimate the MDP parameters.
  - Then use policy-iterations, value iterations.

- Monte Carlo methods:
  - estimating the rewards by empirical averages

- Temporal Difference methods:
  - Combine Monte Carlo methods with Dynamic Programming

- Linear function approximation in Q-learning
  - Similar to SGD
  - Learning heuristic function

Policy class and policy gradient methods

$$\boxed{\pi_w(s,a) = \frac{e^{f(s,a)^T w}}{\sum_{a'} e^{f(s,a')^T w}}}$$

- Policy $\pi \in \Pi$
- Parametric policy class:

$$\underline{\Pi = \{\pi_\theta | \theta \in \mathbb{R}^d\}}$$

- Goal: optimize the value

$$V^{\pi} = V^{\pi_\theta} \qquad \underline{\theta^* = \arg\max_{\theta \in \mathbb{R}^d} V^{\pi_\theta}}$$

- Policy gradient methods
  - aim at learning the policy parameter by SGD.

Policy gradient

- Objective function to maximize: $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0),$

*Note how this theorem is non-trivial… The first two terms depends on π, but we did not take the gradient w.r.t. them.

Policy gradient

- Objective function to maximize: $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0),$
- Do SGD: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)},$

*Note how this theorem is non-trivial… The first two terms depends on π, but we did not take the gradient w.r.t. them.

Policy gradient

- Objective function to maximize: $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0),$
- Do SGD: $$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \underline{\widehat{\nabla J(\boldsymbol{\theta}_t)}},$$
- Policy gradient theorem:

*Note how this theorem is non-trivial… The first two terms depends on π, but we did not take the gradient w.r.t. them.

Policy gradient

- Objective function to maximize: $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$,
- Do SGD: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$,
- Policy gradient theorem:

$(ab)' = a'b + ab'$

$J(\theta) = \sum_s d^\pi(s) \sum_a \pi(a|s;\theta) \, Q^\pi(s,a)$

$$\nabla J(\boldsymbol{\theta}) = \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

$\mathbb{E}$ distribution of state induced by π

*Note how this theorem is non-trivial... The first two terms depends on π, but we did not take the gradient w.r.t. them.

Stochastic approximation in policy gradients

$$\nabla J(\boldsymbol{\theta}) = \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

- Sample from running policy π
  - $\underline{(S_1, A_1, R_1), \dots, (S_T, A_T, R_T)}$

Stochastic approximation in policy gradients

$$\nabla J(\boldsymbol{\theta}) = \overset{\mathbb{E}_{d^\pi(s)}}{\sum_s} d^\pi(s) \overset{\mathbb{E}_{a\sim\pi(\cdot|s)}}{\sum_a} Q^\pi(s,a)\nabla_{\boldsymbol{\theta}}\pi(a|s,\boldsymbol{\theta})$$

$$= \pi \cdot \nabla \log \pi$$

$$\mathbb{E}_{a\sim\pi(\cdot|s)}$$

- Sample from running policy π
  - $(S_1, A_1, R_1), \ldots, (S_T, A_T, R_T)$
- Idea: Sample s, then the following is an unbiased estimator (finite horizon episodic case)

$$\sum_{t=1}^{T} \Big(\underbrace{\sum_{\ell=t}^{T} R_\ell}_{\text{MC approximation}} \Big) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

$$\boxed{\nabla_\theta \log \pi = \frac{1}{\pi}\nabla \pi}$$

$$\nabla \pi$$

Stochastic approximation in policy gradients

$$\nabla J(\boldsymbol{\theta}) = \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

- Sample from running policy π
  - $(S_1, A_1, R_1), \ldots, (S_T, A_T, R_T)$
- Idea: Sample s, then the following is an unbiased estimator (finite horizon episodic case)

$$\sum_{t=1}^{T} \Big(\sum_{\ell=t}^{T} R_\ell \Big) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

$$\underline{= \sum_{t=1}^{T} G_t \nabla_\theta \log(\pi(A_t|S_t,\theta))}$$

Stochastic approximation in policy gradients

$$\nabla J(\boldsymbol{\theta}) = \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

- Sample from running policy π
  - $(S_1, A_1, R_1), \ldots, (S_T, A_T, R_T)$
- Idea: Sample s, then the following is an unbiased estimator (finite horizon episodic case)

$$\sum_{t=1}^{T} \Big(\sum_{\ell=t}^{T} R_\ell \Big) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

$$= \sum_{t=1}^{T} \underline{G_t} \nabla_\theta \log(\pi(A_t|S_t,\theta))$$

← REINFORCE

↑ Replace with TD approx. / Bootstrapping

***Show that this is an unbiased estimator of the gradient.**

Checkpoint for RL

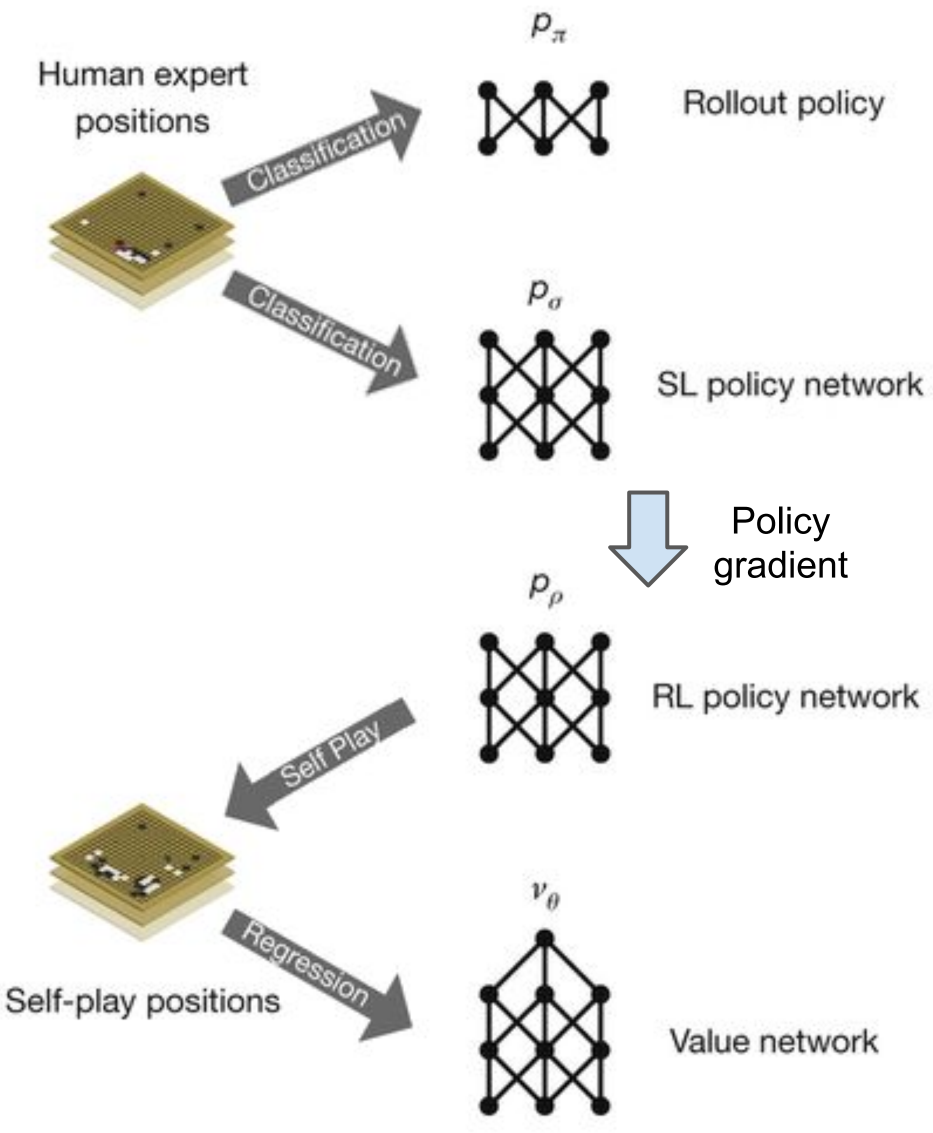
- Model-based methods
- Model-free methods
    - Monte Carlo methods
    - TD-learning: Q-Learning and Sarsa
- Function approximation in RL
    - Approximate the MDP: Model-based
    - Approximate the value function
- Policy gradients
    - Parametrize the policy and run SGD

Elements of State-of-the-Art Reinforcement Learning

- Use a deep neural network to parameterize Q-function
- Use a deep neural network to parameterize the policy \pi
- Run a combination of Q-learning and Policy Gradient.
  - Actor-Critics, A3C, etc…
- Heuristic-based exploration: curiosity, reward shaping, etc..
- Experience replay to generate more data from existing data.
- Multi-agent RL: modeling your opponents

Alpha-Go and Alpha-Zero

- Parameterize the policy networks with CNN
- Supervised learning initialization
- RL using Policy gradient
- Fit Value Network (This is a value function approximation)
- Monte-Carlo Tree Search

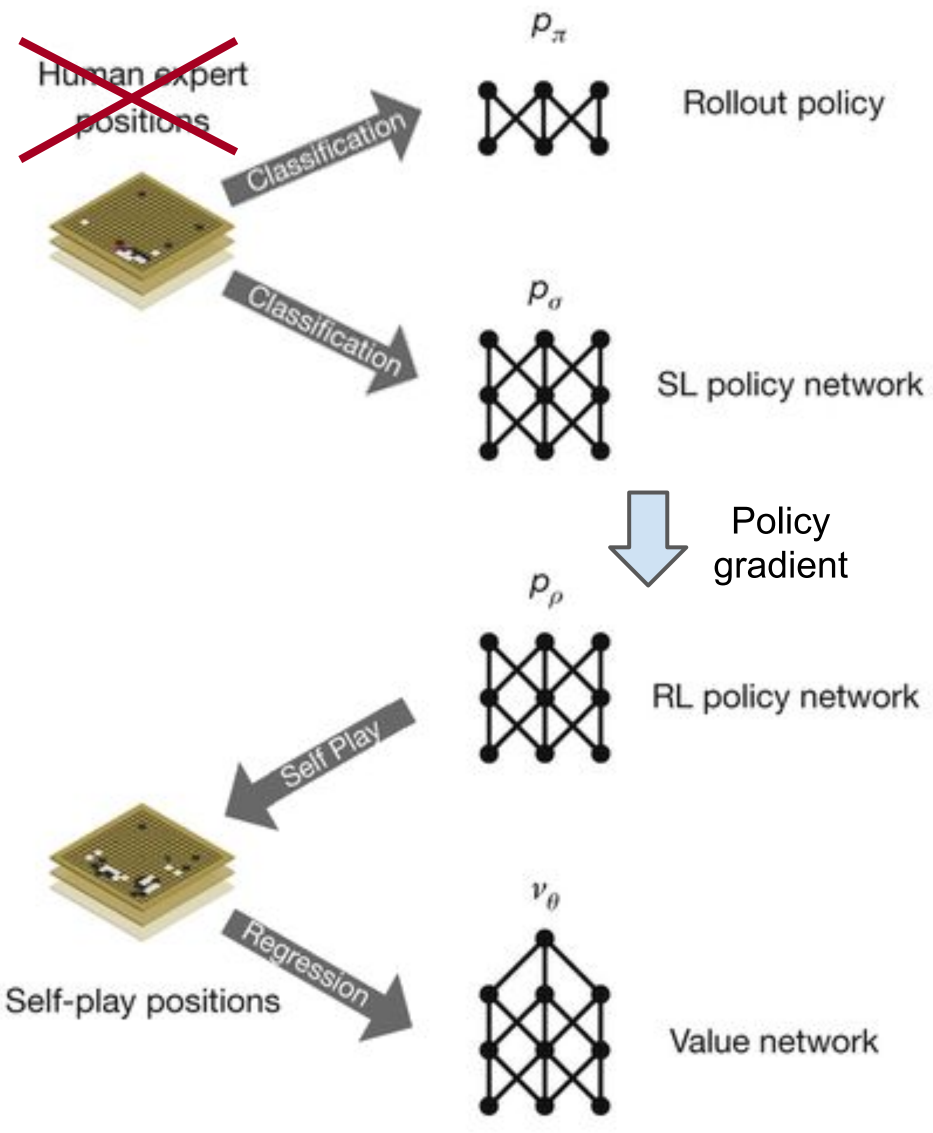


https://www.youtube.com/watch?v=4D5yGiYe8p4

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

Alpha-Go and Alpha-Zero

- Parameterize the policy networks with CNN
- ~~Supervised learning initialization~~
- RL using Policy gradient
- Fit Value Network (This is a value function approximation)
- Monte-Carlo Tree Search

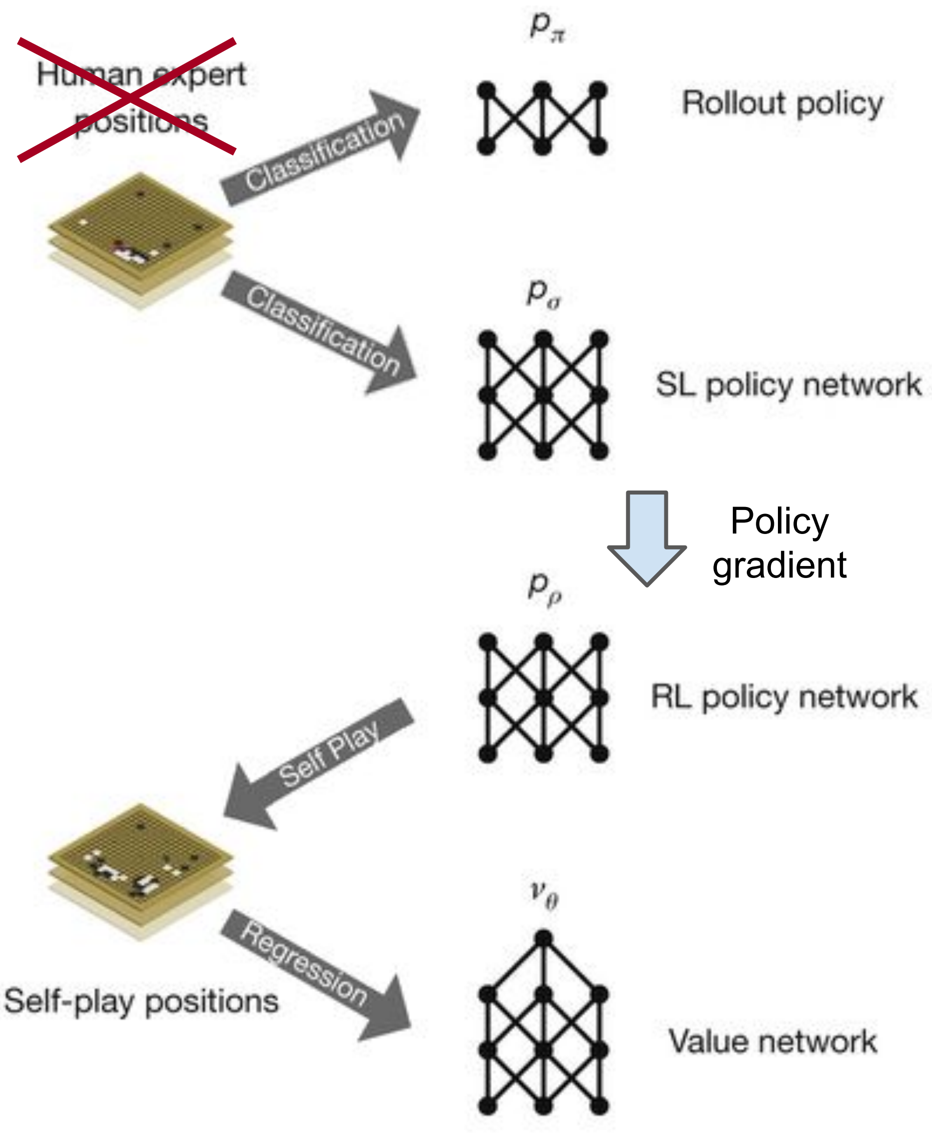


https://www.youtube.com/watch?v=4D5yGiYe8p4

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

Alpha-Go and Alpha-Zero

- Parameterize the policy networks with CNN
- ~~Supervised learning initialization~~
- RL using Policy gradient
- Fit Value Network (This is a value function approximation)
- Monte-Carlo Tree Search



https://www.youtube.com/watch?v=4D5yGiYe8p4

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

D. Silver, et al. "Mastering the game of go without human knowledge." *Nature* 550.7676 (2017): 354-359.

What I did not cover

- Useful results in RL for both theory and alg design
  - Simulation lemma
  - Advantage function and performance difference lemma
- Exploration
  - “Optimism in the face of uncertainty”
- Offline RL
  - “Pessimism in the face of uncertainty”
- How to start research in RL ?
  - Take my RL course (email me to ask for the videos)
  - Solve homework problems, implement RL algorithms from scratch.

Final words to students

- If you are doing theoretical research
  - It’s useful have an empirical mind set
  - implement your algorithm, try it on examples (even toy examples would work)
  - These help you to challenge your assumptions and define theoretical problems that are useful

- If you are doing empirical research
  - Don’t just chase SOTA in benchmarks
  - Think deeply about the problems you are working on
  - ML theory helps you to avoid pitfalls and design better algorithms.

Thank you! Looking forward to your project presentations!