Artificial Intelligence

CS 165A

May 19, 2022

Instructor: Prof. Yu-Xiang Wang

Today

→ Reinforcement Learning

Recap: Tabular MDP

- Discrete State, Discrete Action, Reward and Observation

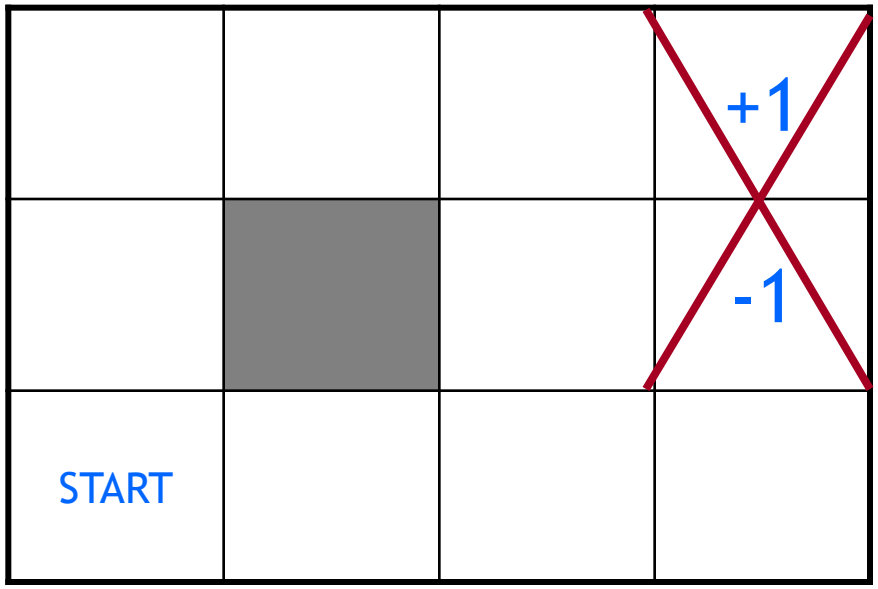
$$S_t \in \mathcal{S} \quad A_t \in \mathcal{A} \quad R_t \in \mathbb{R} \quad \sout{O_t \in \mathcal{O}}$$

- Policy:
  - When the state is observable: $\pi : \mathcal{S} \to \mathcal{A}$
  - ~~Or when the state is not observable~~

$$\sout{\pi_t : (\mathcal{O} \times \mathcal{A} \times \mathbb{R})^{t-1} \to \mathcal{A}}$$

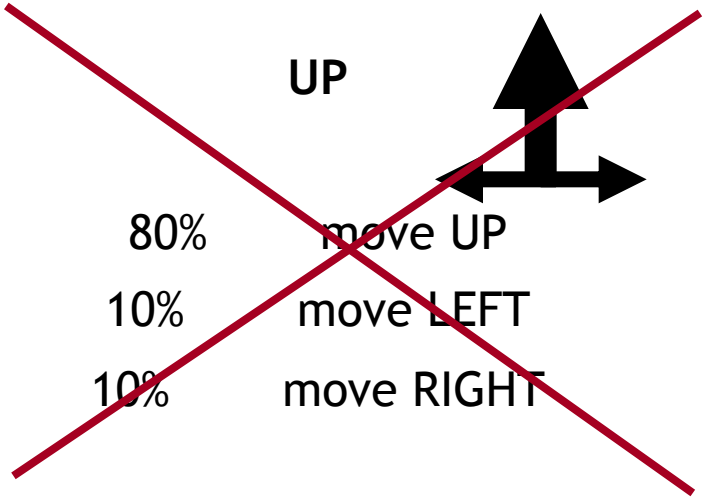
- Learn the best policy that maximizes the expected reward
  - Finite horizon (episodic) RL: $\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}[\sum_{t=1}^{T} R_t]$ T: horizon
  - Infinite horizon RL: $\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} R_t]$ γ: discount factor

Recap: Example: Frozen Lake



Action 1, Action 2, Action 3, Action 4

actions: ~~UP, DOWN, LEFT, RIGHT~~



- ~~reward +1 at [4,3], -1 at [4,2]~~
- ~~reward -0.04 for each step~~
- what's the strategy to achieve max reward?

Recap: Plug in the estimated MDP parameters and then do offline planning.

$$V^{\pi}_{k+1}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} \hat{P}(s'|s,a)[\hat{r}(s,a,s') + \gamma V^{\pi}_{k}(s')]$$

$$\pi' \leftarrow \arg\max_{a} \sum_{s'} \hat{P}(s'|s,a)[\hat{r}(s,a,s') + \gamma V^{\pi}_{k}(s')]$$

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} \hat{P}(s'|s,a)[\hat{r}(s,a,s') + \gamma V_{k}(s')]$$

* Note the **"hat".** Usually it indicates empirical estimates.

* These iterations will produce $\hat{V}^*$ and $\hat{Q}^*$ functions, and then $\hat{\pi}^*$

Recap: Three ideas for solving RL

- Idea 1: Model-based approach
  - Estimated the CPTs of MDP by their empirical frequency
  - Plug-in the estimate to Bellman equations for VI / PI

- Idea 2: Model-free approach: Directly estimate V function and Q function
  - Monte Carlo: Run many episodes of the MDP
  - First-visit MC: first time you visit State s, keep all subsequent rewards, then average over many such episodes

- Idea 3: Better model-free approach: Combining MC with VI / PI directly.
  - Temporal difference (TD) learning

This lecture

- Monte Carlo Methods
- Temporal Difference Learning
- Features and linear function approximation
- Policy Gradient method

Idea 2: **Model-free** Reinforcement Learning

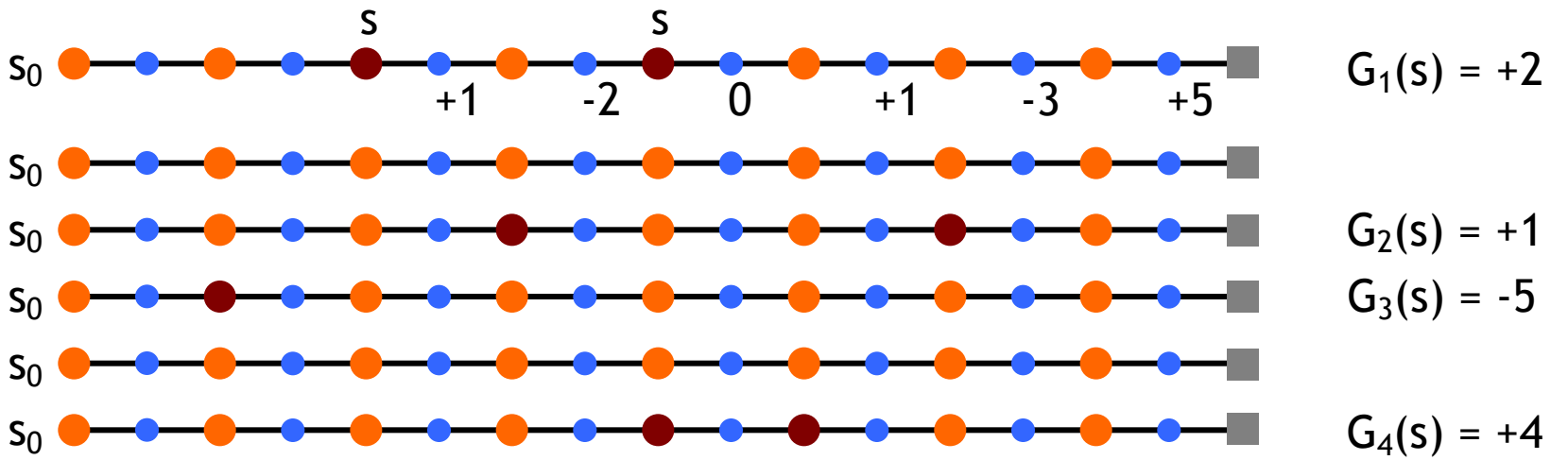
- Do we need the model? Can we learn the Q function directly?
  - **How many free parameters are there to represent the Q-function?**
  - **Ans: SA << O(S^2A)**
- Recall: Policy iterations

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E V^*$$

  - Maybe we can do policy evaluation without estimating the model?

Monte Carlo Policy Evaluation (Prediction)

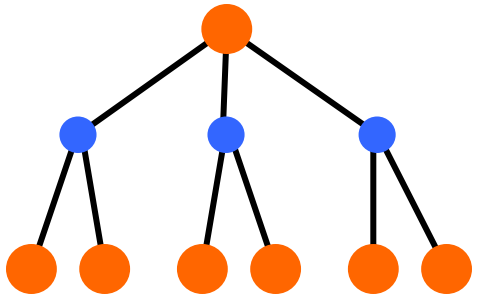
- want to estimate $V^{\pi}(s)$
  = expected return starting from s and following π
  – estimate as average of observed returns in state s
- We can execute the policy π
- first-visit MC
  – average returns following the first visit to state s



$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$

Monte Carlo Policy Optimization (Control)

- V^π not enough for policy improvement
  - need exact model of environment



- estimate $Q^\pi(s,a)$

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$

- MC control

$$\pi_0 \rightarrow^E Q^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E Q^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E Q^*$$

  - update after each episode

- Two problems
  - greedy policy won't explore all actions eps-greedy!
  - Requires many independent episodes for the estimated value function to be accurate.

Improved Monte-Carlo Q-function estimate using Bellman equations

- Recall:

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

$$Q^{\pi}(s,a) = r^{\pi}(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)}[V^{\pi}(s')]$$

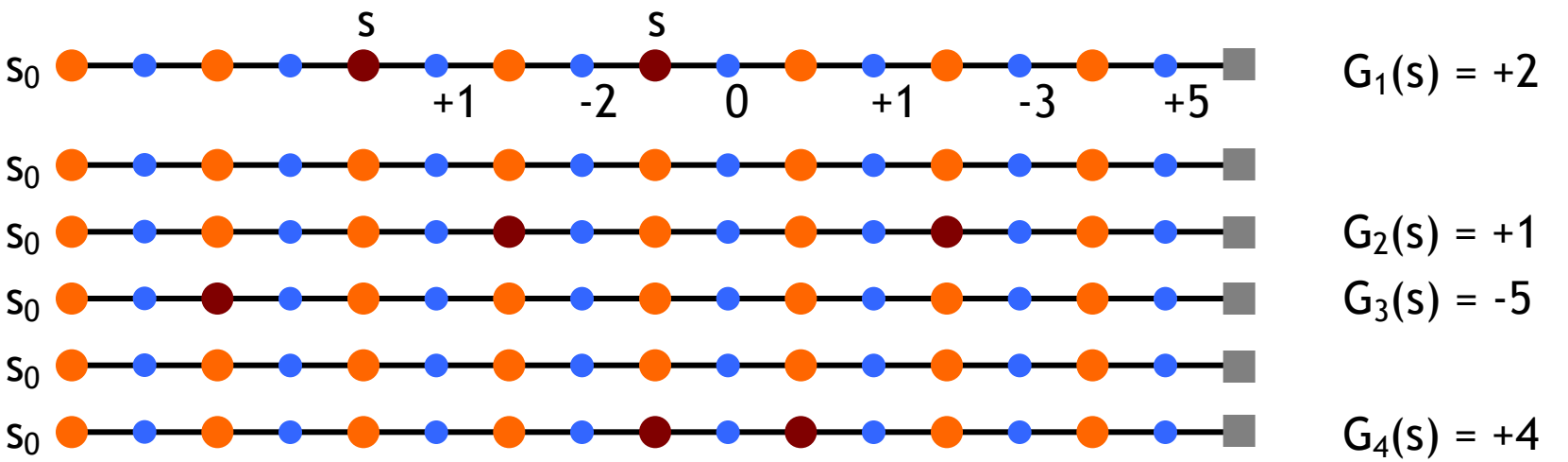
- We can use the empirical (Monte Carlo) estimate.

$$\widehat{Q^{\pi}}(s,a) = \widehat{r^{\pi}}(s,a) + \gamma \widehat{\mathbb{E}}_{s' \sim P(s'|s,a)}[\widehat{V}^{\pi}(s')]$$

*No need to estimate P(s' | s,a) or r(s,a,s') as intermediate steps.

*Require only O(SA) space, rather than O(S^2A)

Online averaging representation of MC



$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$

- Alternative, *online averaging* update

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big], \quad \text{where } \alpha = 1/N_{S_t}$$

DP + MC = Temporal Difference Learning

- Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big],$$

Issue: G_t can only be obtained after the entire episode!

- The idea of TD learning:

$$\mathbb{E}_\pi[G_t] = \mathbb{E}_\pi[R_t|S_t] + \gamma V^\pi(S_{t+1})$$

We only need one step before we can plug-in and estimate the RHS!

- TD-Policy evaluation

Bootstrapping!

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

Bootstrap's origin

- "The Surprising Adventures of Baron Münchausen"
  - Rudolf Erich Raspe, 1785

PULL YOURSELF UP BY THE BOOT STRAPS!!!

- In statistics: Brad Efron's resampling methods
- In computing: Booting…
- In RL: It simply means TD learning

TD policy optimization (TD-control)

- SARSA (On-Policy TD-control)
  - Update the Q function by bootstrapping Bellman Equation

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$$

  - Choose the next A' using Q, e.g., eps-greedy.

- Q-Learning (Off-policy TD-control)
  - Update the Q function by bootstrapping Bellman Optimality Eq.

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

  - Choose the next A' using Q, e.g., eps-greedy, or any other policy.

Remarks:

- These are **proven to converge** asymptotically.
- Much more data-efficient in practice, than MC.
- Regret analysis is still active area of research.

Advantage of TD over Monte Carlo

- Given a trajectory, a roll-out, of T steps.
  - MC updates the Q function only once
  - TD updates the Q function (and the policy) T times!

Remark: This is the same kind of improvement from Gradient Descent to Stochastic Gradient Descent (SGD).

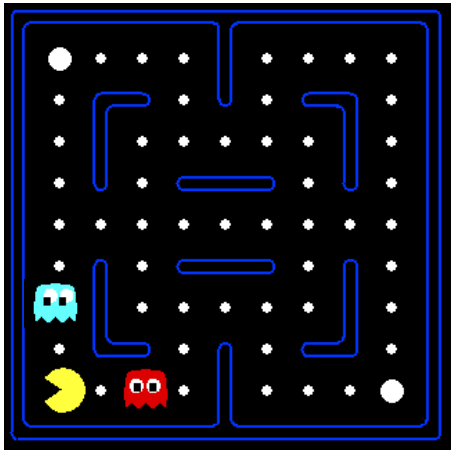
(Optional reading: Sutton and Barton 9.3 Semi-gradient methods)

The problem of large state-space is still there

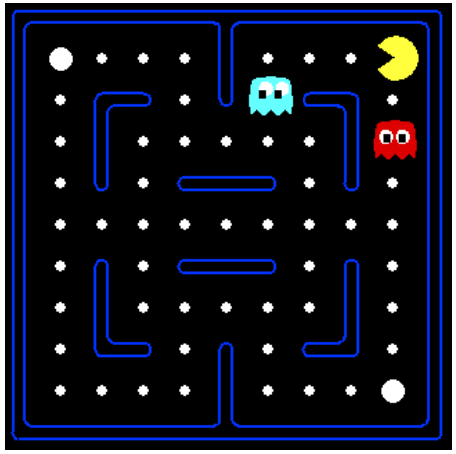
- We need to represent and learn SA parameters in Q-learning and SARSA.
- S is often large
  - 9-puzzle, Tic-Tac-Toe: 9! = 362,800, S^2 = 1.3*10^11
  - PACMAN with 20 by 20 grid. S = O(2^400), S^2 = O(2^800)
- O(S) is not acceptable in some cases.
- Need to think of ways to "generalize"/share information across states.

Example: Pacman

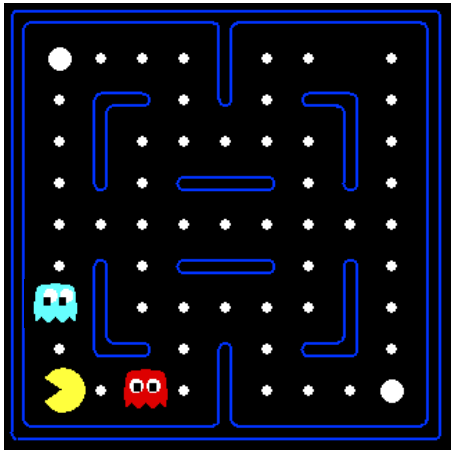
Let's say we discover through experience that this state is bad:



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



Or even this one!



(From Dan Klein and Pieter Abbeel)

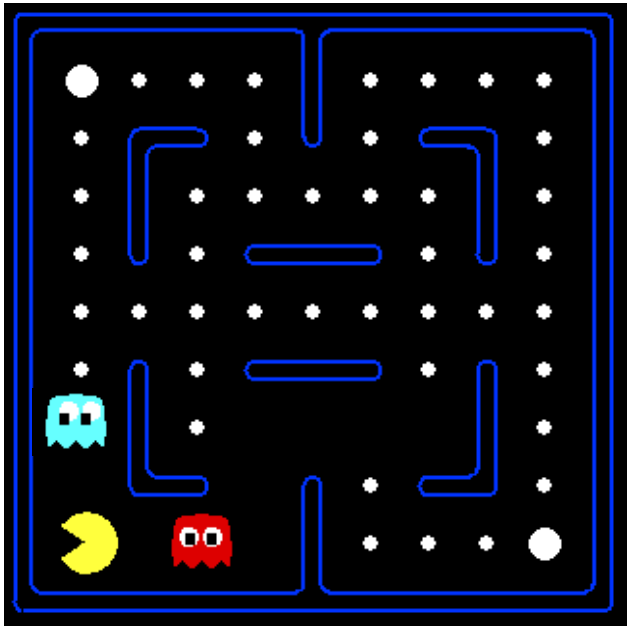
Video of Demo Q-Learning Pacman – Tiny – Watch All

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:
  - $V_{\mathbf{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - $Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$
- 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

- 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:

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

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:

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

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

Q-Learning with function approximation

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

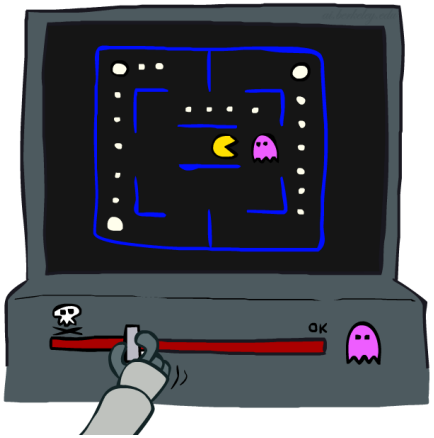
- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}] \qquad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \qquad \text{Approximate Q's}$$



- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares (Read the textbook!)

PACMAN Q-Learning (Linear function approx.)

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

*Question: What is the policy class Π of interest in these methods?

Policy gradient

- Let's not worry about states, dynamics, Q function.
  - We might not even observe the true state.
  - Let's specify a class of parametrized policy and hope to compare to the best within this class

- 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:

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

*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
  - $(S_1, A_1, R_1), \ldots, (S_T, A_T, R_T)$

- Idea: Sample s, then the following is an unbiased estimator (finite horizon episodic case)

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

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

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

The REINFORCE algorithm (Williams, 1987)

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

- From Sutton and Barto Ch .13.
- Note the γ^t term. This is for the discounted (episodic) case
- Updating the parameter T times for each episode!
- Easy to implement easy to understand from SGD theory.

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

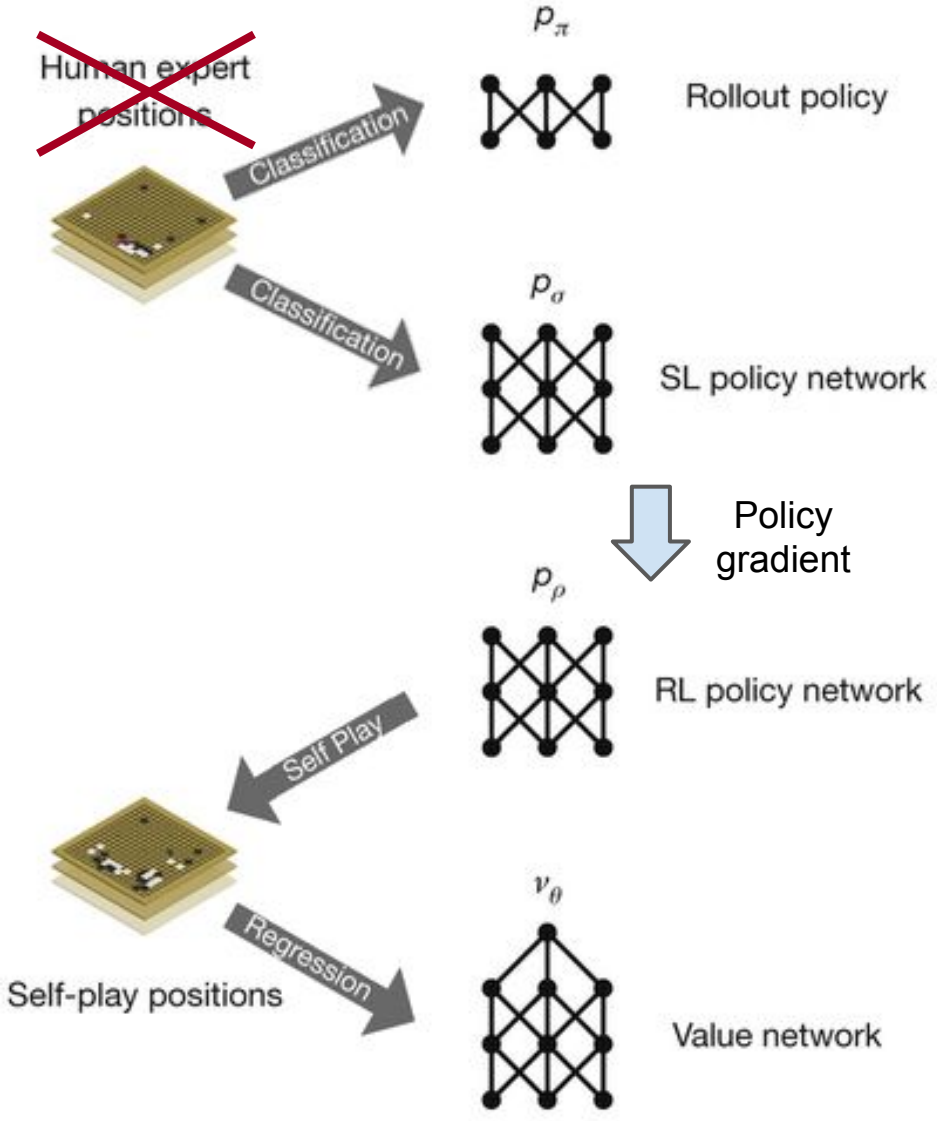
Example of State-of-the-Art RL for Hide-n-Seek

Lowe, Wu et al. (2017)
https://proceedings.neurips.cc/paper/2017/hash/68a9750337a418a86fe06c1991a1d64c-Abstract.html

Alpha-Go and Alpha-Zero

- Parameterize the policy networks with CNN
- ~~Supervised learning initialization~~
- RL using Policy gradient
- Fit Value Network (This is a heuristic function!)
- 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.

Summary of RL algorithms

- Model-based:
  - Policy iteration / Value iteration
  - Need to estimate the dynamics (MDP parameters)

- Model-free: (no need to "explicitly" estimate dynamics)
  - TD learning: SARSA, Q-learning
  - Function approximation (Share information across states)

- Absolutely model-free (do not even need an MDP model)
  - Policy gradient

Next lecture

- Start Logical Agent
- Logical inference for propositional logic
- What you should do:
  - Read Chapter 7 of AIMA textbook.
  - Start working on Project 3 if you haven't yet.