CS292F StatRL Lecture 5
RL Algorithms

Instructor: Yu-Xiang Wang

Spring 2021

UC Santa Barbara

Homeworks and Project

- Homework 0 “due” tonight
  - Sticking to the schedule is highly recommended

- You should start forming project team / ideas now
  - Did you check out the list of papers that I sent?

Recap: Lecture 4

- Miscellaneous on MDPs
  - Advantage function and performance difference lemma
  - Other types of MDPs: finite horizon, undiscounted, variable horizon…
- We started to talk about RL
  - Model-based algorithms
  - Model-free algorithms: Monte Carlo method, temporal difference methods.

Recap: MDP planning with access to generative models

- Motivation:
  1. Solving MDP faster / approximately with randomized algs that sample
  2. Study sample complexity of RL with unknown transitions (without worrying about exploration)

- Algorithm of interest: Model-based plug-in estimator.
  - Sample all state-action pairs uniformly. Estimate the transition kernel.
  - Do VI / PI on the approximate MDP.

Recap: “Monte Carlo” prediction / “Monte Carlo” control

- Sutton and Barto notations / terminologies:
  - “Prediction” ⇔ “Policy evaluation”
  - “Control” ⇔ “Policy optimization”
  - Q and V are used to denotes estimates / while q, v denotes the true value functions.
  - 0 based indexing: S_0, A_0, R_1, S_1, A_{sd}, R_2, …
- Idea: Roll out trajectories and average them.

Recap: 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!

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

Recap: TD policy optimization (TD-control)

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

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

  - 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.

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

  - 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.

Recap: Model-free vs Model-based RL algorithms

- Different function approximations

- Different space efficiency

- Which one is more statistically efficient?
  - More or less equivalent in the tabular case.
  - Different challenges in their analysis.

This lecture

- Variants / improvements of Sarsa and Q-learning
- TD-learning with function approximation
- Policy gradient methods

Expected SARSA

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big] \\ &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big], \end{aligned}$$

Expected SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big]$$
$$\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big],$$

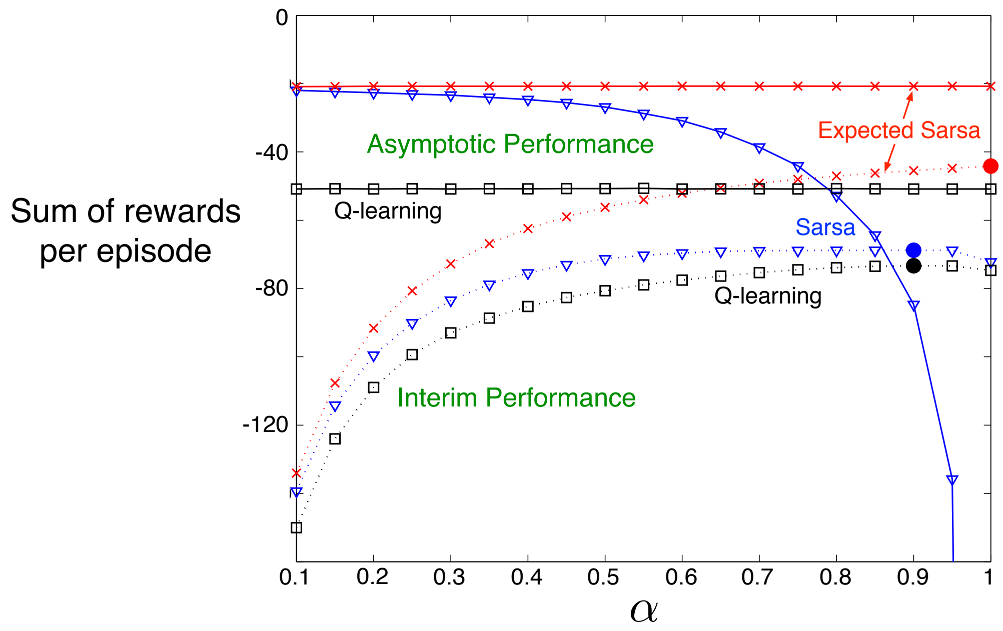


Figure 6.6: Interim and asymptotic performance of TD control methods on the cliff-walking task as a function of α. All algorithms used an ε-greedy policy with $\varepsilon = 0.1$. Asymptotic performance is an average over 100,000 episodes whereas interim performance is an average over the first 100 episodes. These data are averages of over 50,000 and 10 runs for the interim and asymptotic cases respectively. The solid circles mark the best interim performance of each method. Adapted from van Seijen et al. (2009).

Bias in Q-learning and double Q-learning

- Q-learning updates

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big].$$

- Double-Q-learning updates
    - Keep track of two Q function estimates
    - Toss a coin, if "head":

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big].$$

    - If "tail": update Q_2

The problem of large state-space is still there

- We need to represent and learn SA parameters in Q-learning and SARSA.

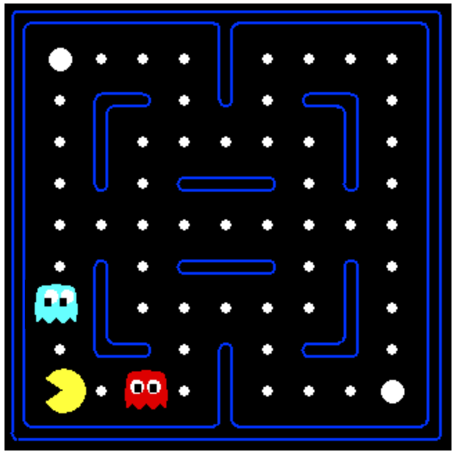
- S is often large
  - 9-puzzle, Tic-Tac-Toe: 9! = 362,800, S^2 = 1.3*10^11
  - PACMAN with 20 by 20 grid. S = O(2^400), S^2 = O(2^800)

- O(S) is not acceptable in some cases.

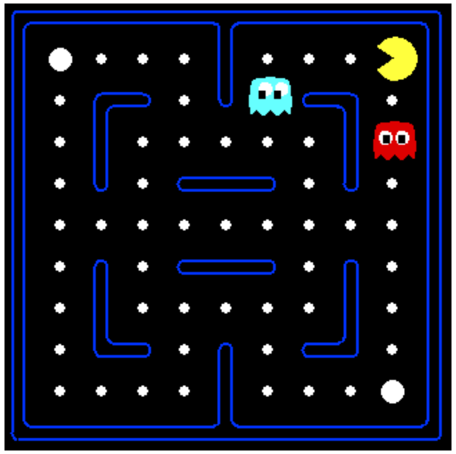
- Need to think of ways to “generalize”/share information across states.

Example: Pacman

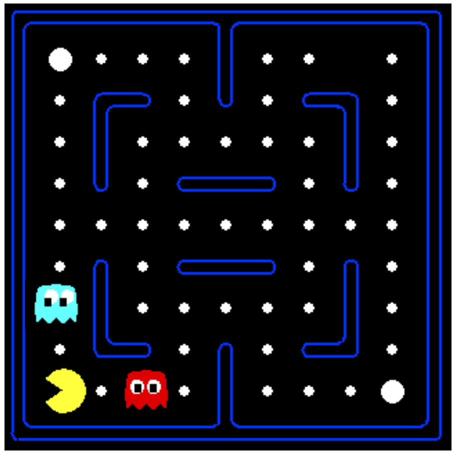
Let's say we discover through experience that this state is bad:



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



Or even this one!



(From Dan Klein and Pieter Abbeel)

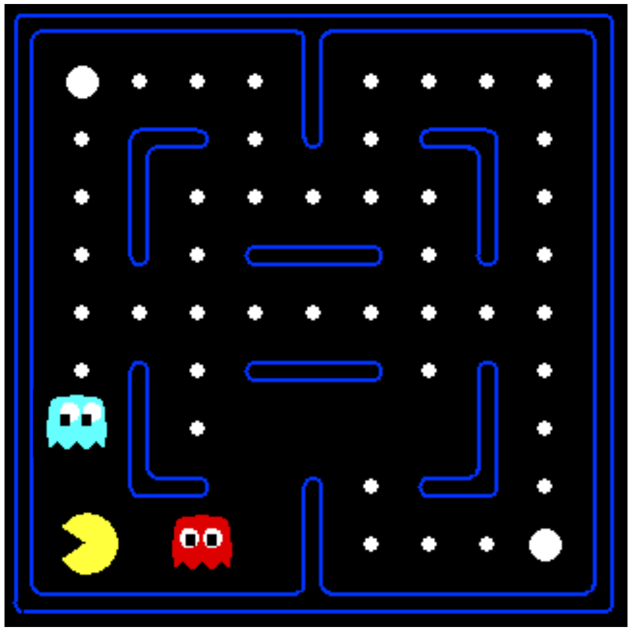
Video of Demo Q-Learning Pacman – Tiny – Watch All

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:

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

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:

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

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

PACMAN Q-Learning (Linear function approx.)

Deriving the TD via incremental optimization that minimizes Bellman errors

- Mean Square Error and Mean Square Bellman error

LSTD: Why doing incremental optimization when we can

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?

Remainder of the lecture

- Policy gradients methods
- Policy gradient theorem
- Extensions

So far we talked about value function approximation, and an induced policy class.

- We can directly work with a parametric policy class.

- Examples:

Optimize policy using SGD

How to estimate the gradient?

- Policy gradient theorem:

Proof of Policy Gradient Theorem

REINFORCE Algorithm

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 $t$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$
```

Actor-Critic algorithm

- REINFORCE with a given baseline

- Actor-Critic: Learn the baseline and use the baseline for "bootstrapping"

Next lecture

- Wrap up RL algorithms
- Exploration: Multi-armed bandits