15-780 – Graduate Artificial Intelligence: Markov Decision Processes

Aditi Raghunathan
Carnegie Mellon University

MDPs formal definition

States: s $\in S$ assumed to be discrete

Actions: a $\in \mathcal{A}$ (assumed to be discrete)

Transition probabilities: distribution over next states given current state and current action

$T(s, a, s')$ is probability of next state being s' when taking action a at state s

Rewards: mapping states or transitions to reals $R(s, a, s')$

Start state, end state, discount factor γ(default 1)

Policy and policy evaluation

A **policy** π is a mapping from each state s $\in S$ to an action a $\in \mathcal{A}$ (or distribution of actions)

Suppose you followed a path $s_0, a_1, r_1, s_1, a_2, r_2, s_2, a_2, r_2, s_3$ ...; the expected sum of discounted rewards is $r_1 + \gamma r_2 + \gamma^2 r_3 +$ ...

Value function $V_\pi: S \rightarrow R$ such that $V_\pi(s)$gives the expected sum of discounted rewards when following policy π from state s

Q-value of a policy $Q_\pi: S \times A \rightarrow R$ such that $Q_\pi(s, a)$ is the expected sum of discounted rewards when taking action a from state s and then following π

Recursive definitions

$V_\pi(s) = Q_\pi(s, \pi(s))$ otherwise *(= 0 if s is the end-state)*

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s') \; [\, R(s, a, s') + \gamma V_\pi(s') \,]$$

Algorithm for policy evaluation: start with arbitrary initialization $V_\pi(s) = 0 \; \forall s$ and then apply the recursion

How to find best policy?

Baseline: Compute $V_\pi(\text{start})$ for all policies π and pick best policy

Alternative: Write down recurrences like before

$$Q_\pi(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_\pi(s') \right]$$

$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_{\text{opt}}(s') \right]$$

$$V_{\text{opt}}(\text{start}) = 0$$

$$V_{\text{opt}}(s) = \max_a Q_{\text{opt}}(s,a)$$

Value iteration

$$Q_{opt}(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_{opt}(s') \right]$$

$$V_{opt}(\text{start}) = 0$$

$$V_{opt}(s) = \max_a Q_{opt}(s,a)$$

$$V_{opt}^{t+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_{opt}^{t}(s') \right]$$

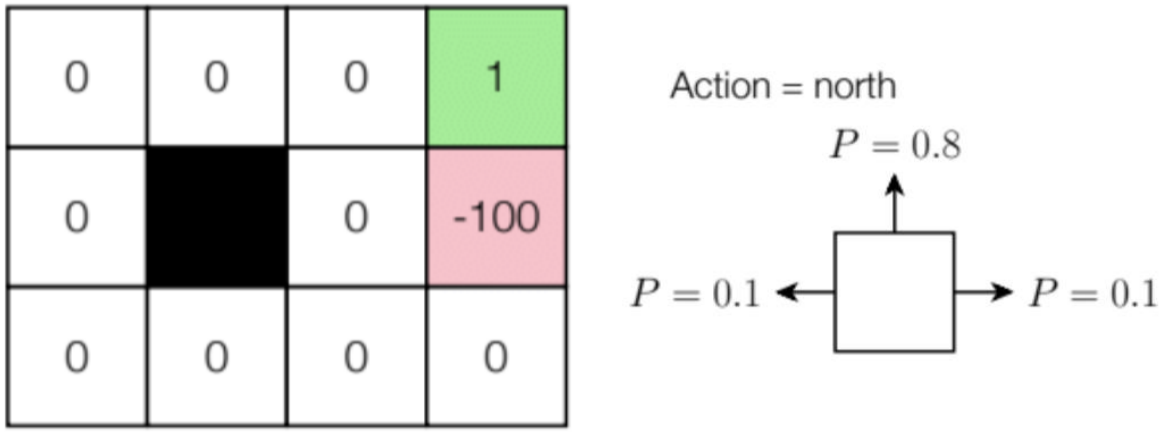
$$\pi_{opt} = \text{argmax}_a \, Q_{opt}(s,a)$$

Gridworld domain

Simple grid world with a goal state with reward and a “bad state”with reward -100

Actions move in desired direction with probability 0.8 and one of two perpendicular directions with probability 0.1 each

Taking an action that bumps into a wall leaves agent where it is



Value iteration

Running value iteration with $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

Running value iteration with $\gamma = 0.9$

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

$\hat{V}$ at one iteration

Running value iteration with $\gamma = 0.9$

0.809	1.598	2.475	3.745
0.268		0.302	-99.59
0	0.034	0.122	0.004

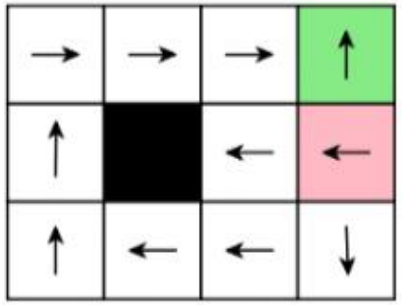
$\hat{V}$ at five iterations

Running value iteration with $\gamma = 0.9$

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$ at 1000 iterations

Running value iteration with $\gamma = 0.9$



Resulting policy after 1000 iterations

Convergence of value iteration

For $\gamma < 1$, value iteration converges to the optimal value

$$V_{opt}^{t+1}(s) = \underbrace{\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_{opt}^{t}(s')\right]}_{B: R^{|S|} \rightarrow R^{|S|}}$$

$$\max_{s \in S} |BV_1(s) - BV_2(s)| \leq \gamma \max_{s \in S} |V_1(S) - V_2(s)|$$

Contraction and hence convergence

Policy iteration

Can we iterate directly on policies instead?

Repeat

Do policy evaluation to compute $V_{\pi^t}(s)$

$$\pi^{t+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{\pi^t}(s')]$$

Policy iteration generally requires fewer iterations than value iteration, but each iteration involves solving a linear system (or running policy evaluation iterations)

Policy iteration on gridworld

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

0.418	0.884	2.331	6.367
0.367		-8.610	-105.7
-0.168	-4.641	-14.27	-85.05

V^{π} at one iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

5.470	6.313	7.190	8.669
4.803		3.347	-96.67
4.161	3.654	3.222	1.526

V^{π} at three iterations (converged)

Reinforcement learning

Value iteration and policy iteration assumes complete knowledge of transitions and rewards

Suppose I want to learn to walk

Reinforcement learning setting: Unknown transitions and/or rewards

Take actions in the world to learn about the world, with the goal of maximizing expected utility

Taking actions in the world often called “online”

Model-based reinforcement learning

Natural strategy: Collect a bunch of data, estimate the transitions and rewards

Data: $s_0, a_1, r_1, s_1, a_2, r_2, s_2, a_3, r_3, \ldots a_n, r_n$

$$\hat{T}(s, a, s') = \frac{\text{number times } (s, a, s') \text{ occurs}}{\text{number times } (s, a) \text{ occurs}}$$

$$\hat{R}(s, a, s') = \frac{\text{Sum } r \text{ over all occurences } (s, a, r, s')}{\text{number times } (s, a, s') \text{ occurs}}$$

Run value iteration with $\hat{T}$ and $\hat{R}$

Model-based Monte Carlo

Natural strategy: Collect a bunch of data, estimate the transitions and rewards

Data: $s_0, a_1, r_1, s_1, a_2, r_2, s_2, a_3, r_3, \ldots a_n, r_n$

$$\hat{T}(s, a, s') = \frac{\text{number times } (s, a, s') \text{ occurs}}{\text{number times } (s, a) \text{ occurs}}$$

$$\hat{R}(s, a, s') = \frac{\text{Sum } r \text{ over all occurences } (s, a, r, s')}{\text{number times } (s, a, s') \text{ occurs}}$$

Run value iteration with $\hat{T}$ and $\hat{R}$

What can go wrong here?

Data collection

You choose how to collect your data in reinforcement learning

Typically called “exploration” where we want to make sure we gather relevant data

Contrasts with supervised learning where the data just comes from the underlying distribution

Model-based to model-free

When we run value iteration with $\hat{T}$ and $\hat{R}$, what do we actually care about?

We end up estimating $\hat{Q}_{\text{opt}}(s, a)$

Can we estimate $\hat{Q}_{\text{opt}}(s, a)$ directly?

Model-free Monte Carlo

Data (generated according to π): $s_0, a_1, r_1, s_1, a_2, r_2, s_2, a_3, r_3, \ldots a_n, r_n$

$$u_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$$\hat{Q}_\pi(s, a) = \text{average of } u_t \text{ where } s_t = s, a_t = a$$

Never estimating transition probabilities or rewards

Model-free Monte Carlo

$\hat{Q}_\pi(s,a) = \text{average of } u_t \text{ where } s_t = s, a_t = a$

For each (s, a, u):

$$\hat{Q}_\pi(s,a) = (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$ What is η?

Can view Model-free Monte Carlo as making updates to the estimate $\hat{Q}_\pi(s,a)$ based on data u

Model-free Monte Carlo

$$\hat{Q}_\pi(s,a) = (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

The "data" u comes from one observation of a trajectory and hence can be very noisy

Is there a way to reduce the noise in updates?

Bootstrapping from previous estimates

Suppose we observe (s, a, r, s') in the data

$$\hat{Q}_\pi(s,a) = (1-\eta)\hat{Q}_\pi(s,a) + \eta(r + \gamma\hat{Q}_\pi(s'))$$

SARSA algorithm

SARSA vs Model-free Monte Carlo

Both estimate $\hat{Q}_\pi(s,a)$ for some policy π

SARSA updates using $(r + \gamma \hat{Q}_\pi(s'))$ while Model-free Monte Carlo uses utility u

- Which estimator has smaller bias?
- Which estimator has smaller variance?
- Which estimator can be computed on-the-fly vs waiting for the episode to end?

Q-learning

So far we talked about policy evaluation (estimating $\hat{Q}_\pi(s, a)$)

What about the **optimal policy**?

We extend the ideas of SARSA and Model-free Monte Carlo

Have we done something similar before?

Q-learning

Recall MDP recurrence for $\mathrm{Q}_{\mathrm{opt}}(s,a)$

$$\mathrm{Q}_{\mathrm{opt}}(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_{\mathrm{opt}}(s') \right] \qquad \mathrm{V}_{\mathrm{opt}}(\mathrm{s}) = \max_a Q_{\mathrm{opt}}(s,a)$$

For estimating $\hat{Q}_{\mathrm{opt}}(s,a)$ from data:

$\hat{Q}_{\mathrm{opt}}(s,a)$ is the **empirical average** of $[\, R(s,a,s') + \gamma\, \mathrm{V}_{\mathrm{opt}}\,(s')]$

$$\hat{Q}_{\mathrm{opt}}(s,a) = (1 - \eta)\hat{Q}_{\mathrm{opt}}(s,a) + \eta\, [R(s,a,s') + \gamma \hat{V}_{opt}(s')$$

$$\hat{V}_{\mathrm{opt}}(\mathrm{s}') = \max_a \hat{Q}_{\mathrm{opt}}(s',a)$$

Q-learning vs SARSA

For each (s, a, r, s′):

$$\hat{Q}_\pi(s,a) = (1-\eta)\hat{Q}_\pi(s,a) + \underbrace{\eta(r + \gamma \hat{Q}_\pi(s'))}$$

Following same policy π

For each (s, a, r, s′):

$$\hat{Q}_{\text{opt}}(s,a) = (1-\eta)\hat{Q}_{\text{opt}}(s,a) + \eta\underbrace{(r + \gamma \max_a \hat{Q}_{\text{opt}}(s',a))}$$

Effectively following a different "optimal" policy

On-policy vs off-policy

On-policy: evaluate or improve the data-collecting policy

Off-policy: evaluate or learn the optimal policy using data from different policy

Classify into on-policy or off-policy:

- Model-based Monte Carlo
- Model-free Monte Carlo
- SARSA
- Q-learning

Summary of RL algorithms

Algorithm	**Estimating**	**Based on**
Model-Based Monte Carlo	$\hat{T}, \hat{R}$	$s_0, a_1, r_1, s_1, \ldots$
Model-Free Monte Carlo	$\hat{Q}_\pi$	u
SARSA	$\hat{Q}_\pi$	$r + \hat{Q}_\pi$
Q-Learning	$\hat{Q}_{\text{opt}}$	$r + \hat{Q}_{\text{opt}}$

What exploration policy to use?

Greedy: $\pi(s) = \arg\max_a \hat{Q}_{\text{opt}}(s, a)$

Efficient but can be inaccurate depending on initial values --- don't correct initial estimates

Random:

Accurate (doesn't get swayed by initial values) but inefficient

Sweet spot is a combination of the two

Properties of Q learning

- Q-learning converges to the optimal policy even if act suboptimally
  - With sufficient exploration
  - ϵ –greedy: combination of random and greedy
  - All states, actions have to be visited infinitely often

$$\hat{Q}_{\text{opt}}(s,a) = (1-\eta)\hat{Q}_{\text{opt}}(s,a) + \eta(r + \gamma \max_a \hat{Q}_{\text{opt}}(s',a))$$

- Some conditions on η
  - Large enough to enable updates
  - Small enough o prevent bouncing around

Piazza poll

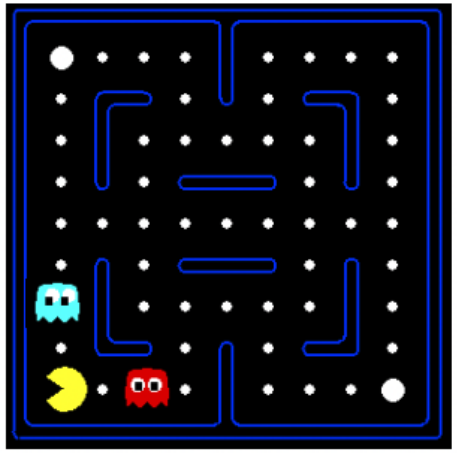
ϵ –greedy: does it make sense to increase or decrease ϵ across iterations?

ϵ: probability of taking a random action

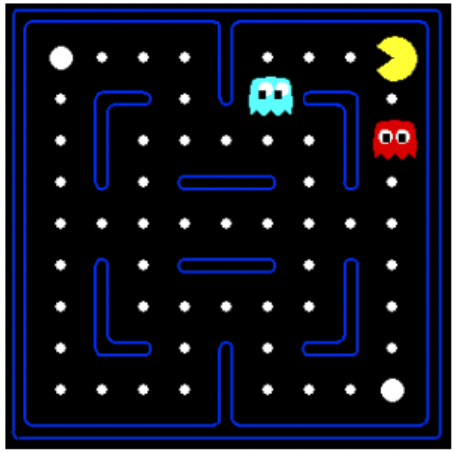
Generalization across states

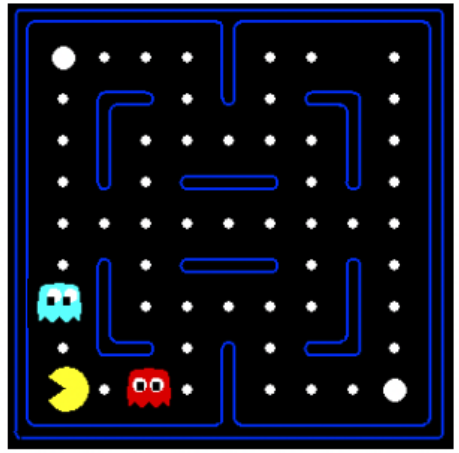
So far, all estimates $\hat{Q}_{opt}(s, a)$ are stored as a table of values

When does this become inefficient?



Suppose we estimate that this state is bad





We want to avoid these other states too..

Generalization across states

Parameterize $\hat{Q}_{opt}(s,a)$ as a function $Q_\theta(s,a)$

Example: linear functions

$\hat{Q}_\theta(s,a) = \hat{w}^{\mathrm{T}}\phi(s,a)$ and estimating $\hat{Q}_{opt}(s,a)$ is now estimating $\hat{w}$

Nowadays: $Q_\theta(s,a)$ is a neural network

We can now reduce the # quantities to estimate, but have to be careful that the features are actually appropriate: states with similar features should have similar values

Online least squares view of Q-learning

For each (s, a, r, s′):

$$\hat{Q}_{\text{opt}}(s,a) = (1-\eta)\hat{Q}_{\text{opt}}(s,a) + \eta(r + \gamma \max_a \hat{Q}_{\text{opt}}(s',a))$$

For each (s, a, r, s′):

$$\hat{Q}_{\text{opt}}(s,a) = \hat{Q}_{\text{opt}}(s,a) - \eta\big(\hat{Q}_{\text{opt}}(s,a) - \underbrace{(r + \gamma \max_a \hat{Q}_{\text{opt}}(s',a))}_{\text{Target } t}\big)$$

Equivalent to gradient descent on $(t - \hat{Q}_{\text{opt}}(s,a))^2$

Online least squares view of Q-learning

What happens when doing Q-learning with features?

Use the same objective $(t - \hat{Q}_{\text{opt}}(s,a))^2$ and apply chain rule for gradient descent

Example linear functions $\hat{Q}_{opt}(s,a) = \hat{w}^{\mathrm{T}}\boldsymbol{\phi}(s,a)$

For each (s, a, r, s'):

$$\hat{w} = \hat{w} - \eta\big(\hat{w}^{\top}\phi(s) - \underbrace{(r + \gamma \max_{a} \hat{w}^{\top}\phi(s',a))}_{\text{Target } t})\phi(s,a)$$