4/11/2024: Q-Learning

But first: What is the optimal policy if $\underbrace{\text{we know the full MDP}}$?

we know $T(s,a,s')$ & Rewards(s,a,s')

$\underbrace{V_{OPT}(s)}_{\text{"optimal value"}}$ = maximum possible expected discounted sum of rewards for any policy, starting at state s

$\underbrace{Q_{OPT}(s,a)}_{\text{"Q-value"}}$ = maximum possible expected discounted sum of rewards for any policy, starting at state s & forced to take action a

$$V_{OPT}(s) = \begin{cases} 0 & \text{if } \mathrm{IsEnd}(s) \\ \underbrace{\max_{a \in \mathrm{Actions}(a)}}_{\text{pick the best } a} \underbrace{Q_{OPT}(s,a)}_{\text{play optimally after choosing } a} & \text{else} \end{cases}$$

$$Q_{OPT}(s,a) = \sum_{s' \in S} \underbrace{T(s,a,s')}_{\text{prob of going to } s'} \Big[\underbrace{\mathrm{Reward}(s,a,s')}_{\text{Reward now (no discounting)}} + \underbrace{\gamma \cdot V_{OPT}(s')}_{\text{Future reward (discounted)}}\Big]$$

If we knew $Q_{OPT}(s,a)$ for all s & a,
Optimal policy $\pi^*(s) = \arg\max_{a \in \mathrm{Actions}(s)} Q_{OPT}(s,a)$

Lesson: If we can estimate $Q_{OPT}(s,a)$ well, we can deduce the optimal policy!

Now: Reinforcement Learning

- We believe the world works like an MDP
- But we don't know $T(s,a,s')$ & $Reward(s,a,s')$
- Can only learn about world by interacting with it.

RL training pseudocode:

For episode $= 1, 2, 3, \dots$:

$s_1 \leftarrow s_{start}$

For $t = 1, \dots,$

- Agent choose action $a_t = \underbrace{\pi_{act}}_{\text{policy used to act during learning}}(s_t)$
- Agent receives:
  - Reward r_t
  - New state s_{t+1}
- Update agent's parameters (AKA learning)
- If IsEnd(s_{t+1}) : break

Learning algorithm: Q-Learning

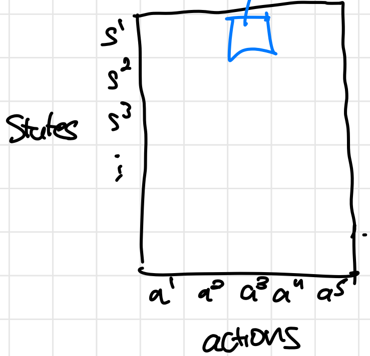
Goal: Learn $Q_{opt}(s,a)$ for every (s,a)

Version 1: Tabular Q-Learning

Maintain a big table of all states x all actions

Can initialize all to 0

$= \hat{Q}(s^1, a^3) =$ our estimate of $Q_{opt}(s^1, a^3)$



each # in table is 1 parameter to learn

How to learn $\hat{Q}(s,a)$?

We get 1 "training example" at a time

consisting of (s, a, r, s')

weighted average of old guess + new estimate of future reward

Update Rule:

$$\hat{Q}(s,a) \leftarrow \underbrace{(1-\eta)}_{\text{"learning rate" } (\approx 0.1)} \underbrace{\hat{Q}(s,a)}_{\text{old Q-value guess}} + \eta \underbrace{\Big(\underbrace{r}_{\text{reward now}} + \underbrace{\gamma \hat{V}(s')}_{\text{anticipated future reward}}\Big)}_{\text{Estimate of total future reward}}$$

where $\hat{V}(s) = \begin{cases} \max_{a \in Actions(s)} \hat{Q}(s,a) & \text{if not IsEnd}(s) \\ 0 & \text{else} \end{cases}$

How do we choose π_{act}?

- Natural answer (WRONG):

At each state s, choose $a \in \underset{a \in Actions(s)}{\arg\max} \hat{Q}(s,a)$

✓ optimal if $\hat{Q}(s,a) = Q_{opt}(s,a)$

✗ Bad idea early in training

Suppose we one visit s, take a, receive large reward.

⇒ $\hat{Q}(s,a)$ would be large

⇒ "Natural" policy would always take a in state s

All exploitation, no exploration

- Simple Fix: ε-Greedy

At each timestep, at state s:

- with probability $1-\varepsilon$ choose $\underset{a \in Action(s)}{\arg\max} \hat{Q}(s,a)$

(Exploitation)

- with probability ε: Choose random action in $Actions(s)$

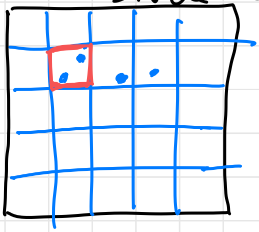
(Exploration)

Usually $\varepsilon = 0.1$ or 0.01

After training: Set $\varepsilon = 0$

How to deal with large state spaces?

Option 1: Discretize state space
- E.g. State might be continuous (eg location of robot)
- Divide continuum into many buckets



Discretize area into 5x5 grid = 25 states

Option 2: Version 2 of Q-Learning = Q-learning with Linear Function Approximation

Idea: Q-learning is kind of like regression where
$x = (s,a)$, $y = \underbrace{\text{Q-value}}$

Not like supervised learning b/c we don't know Q-values

Goal: Learn a linear model mapping (s,a) to Q-value
1. Need feature function $\phi(s,a) \in \mathbb{R}^d$
2. Learn parameter vector $w \in \mathbb{R}^d$ to predict $\hat{Q}(s,a) = w^T \phi(s,a)$

How to learn w? Re-examine old Q-learning formula.

$$\hat{Q}(s,a) \leftarrow (1-\eta)\hat{Q}(s,a) + \eta(r + \gamma \hat{V}(s'))$$

$$= \hat{Q}(s,a) - \eta \underbrace{\left(\hat{Q}(s,a) - [r + \gamma \hat{V}(s')]\right)}_{\text{"gradient"}}$$

learning rate

For Q-Learning with Function Approx:
minimize squared error between

$$\underbrace{\hat{Q}(s,a)}_{\text{"prediction"}} \quad \text{and} \quad \underbrace{r + \hat{V}(s')}_{\text{"target"}}$$

$$\text{Loss}(w) = \frac{1}{2}\left(\hat{Q}(s,a) - [r + \hat{V}(s')]\right)^2$$

$$\nabla_w \text{Loss}(w) = \cancel{\tfrac{1}{2}} \cdot \cancel{2} \cdot \left(w^T \phi(s,a) - [r + \hat{V}(s')]\right) \cdot \phi(s,a)$$

so the gradient descent update is:

$$w \leftarrow w - \eta \cdot \nabla_w \text{Loss}(w)$$

$$= w - \eta\left(w^T \phi(s,a) - [r + \hat{V}(s')]\right) \cdot \phi(s,a)$$