

Recitation of Reinforcement Learning

Wednesday Oct 24th

Why Reinforcement learning?

- The input of real-world problem is hard to be easily formulated, the domain is super large and complicate.
- By formulating problem to reinforcement learning problem, it's much easier to measure “how good” is your behavior.

But, it's not always easier to get feedback:

Game like chess would only get reinforcement after reaching the end of game. But for some games with **complicate domain** but **immediate feedback** like ping-pong and learning to crawl.

How to think as Reinforcement Learning

Still assume a Markov decision process (MDP):

- A set of states $s \in S$

- A set of actions (per state) A

- A model $T(s,a,s')$

- A reward function $R(s,a,s')$

Let's make conditions different: We don't know T and R .

- We don't know which states are good.

- Must try some trials to learn from reinforcement.

Passive Reinforcement Learning

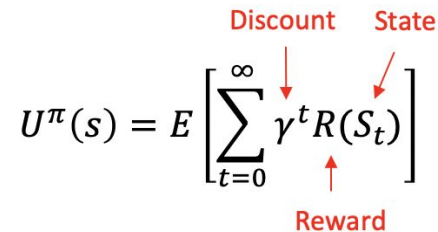
What do we know?

A fixed policy $\pi(s)$

You don't know the transitions $T(s,a,s')$

Goal: learn the values for each state under π .

Direct

$$U^\pi(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \right]$$


The equation is annotated with red text and arrows: 'Discount' with a downward arrow pointing to the γ term, 'State' with a diagonal arrow pointing to the S_t term inside the summation, and 'Reward' with an upward arrow pointing to the R term inside the summation.

Passive Reinforcement Learning

Idea:

The agent would run a set of trials.

For each trial, recursively update the value of each state.

Average the value by N trials.

Passive Reinforcement Learning

Suppose the reward for each step is -0.04.

What's the weakness of it?

The probability relation between state is ignored.

trials

↑ 0.88	→ 0.92	→ 0.96	1
↑ 0.84			
↑ 0.80			

mean utility

0.88	0.92	0.96	1
0.84		?	?
0.80	?	?	?

		→ -1.04	-1
→ -1.16	→ -1.12	↑ 1.08	

0.88	0.92	0.96	1
0.84		-1.04	-1
-0.18	-1.12	-1.08	?

Passive Reinforcement Learning

Adaptive Dynamic Programming:

It takes the cons of Bellman equation:

the probability relation is introduced.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

Understand the notation clearly!

Passive Reinforcement Learning

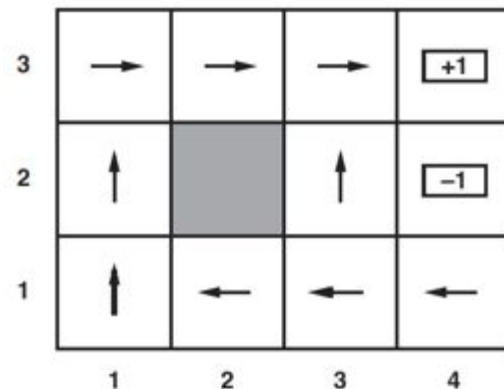
ADP:
$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^\pi(s')$$

Two trials:

1. $(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (2,3) \rightarrow (4,3)$
2. $(1,1) \rightarrow (1,2) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2)$

Let's try to compute value:

- Construct the probability table
- Write down value equation for each state



Passive Reinforcement Learning

Temporal Difference Learning:

The goal is not changed: iterate through the policy to update the value.

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

↑ ↑
Learning rate Observed
parameter successor of s

- Instead of being significantly impacted by the all the successor's value, we introduced the learning rate and the difference between current U and successor's U.
- Learning rate: if we change α from a fixed parameter to a function that decreases as the number of times a state has been visited increases, then $U^\pi(s)$ itself will converge to the correct value, carefully design your learning rate function

Watch out the difference between ADP and TD

function PASSIVE-TD-AGENT(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r'

persistent: π , a fixed policy

U , a table of utilities, initially empty

N_s , a table of frequencies for states, initially zero

s, a, r , the previous state, action, and reward, initially null

if s' is new **then** $U[s'] \leftarrow r'$

if s is not null **then**

increment $N_s[s]$

$U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$

if $s'.\text{TERMINAL?}$ **then** $s, a, r \leftarrow \text{null}$ **else** $s, a, r \leftarrow s', \pi[s'], r'$

return a

function PASSIVE-ADP-AGENT(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r'

persistent: π , a fixed policy

mdp , an MDP with model P , rewards R , discount γ

U , a table of utilities, initially empty

N_{sa} , a table of frequencies for state-action pairs, initially zero

$N_{s'|sa}$, a table of outcome frequencies given state-action pairs, initially zero

s, a , the previous state and action, initially null

if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$

if s is not null **then**

increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$

for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**

$P(t | s, a) \leftarrow N_{s'|sa}[t, s, a] / N_{sa}[s, a]$

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$

if $s'.\text{TERMINAL?}$ **then** $s, a \leftarrow \text{null}$ **else** $s, a \leftarrow s', \pi[s']$

return a

TD adjusts a state to agree with its observed successor, whereas ADP adjusts the state to agree with all of the successors that might occur, weighted by their probabilities.








Passive Reinforcement Learning

TD learning example:

For learning rate, let's define it to be: $\alpha(N) = 1/n$, which n denotes the number of times we have been in that state.

Trial	$\langle \text{state}, \text{reward} \rangle$ series
1	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
2	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{S} \langle (2, 2), 0 \rangle \xrightarrow{N} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
3	$\langle (1, 1), 0 \rangle \xrightarrow{S} \langle (2, 1), 0 \rangle \xrightarrow{N} \langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$

Let's solve the question:

	1	2	3
1			
2			
3		<div data-bbox="1516 884 1593 966">-1</div>	<div data-bbox="1661 884 1738 966">+1</div>

TD learning:

Trial	$\langle \text{state}, \text{reward} \rangle$ series
1	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
2	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{S} \langle (2, 2), 0 \rangle \xrightarrow{N} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
3	$\langle (1, 1), 0 \rangle \xrightarrow{S} \langle (2, 1), 0 \rangle \xrightarrow{N} \langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$

Trial 1

The initial utilities are all zero, and the only observed reward is in the final state in the trial, state $(3, 3)$. Thus, the only non-trivial update while performing the updates for trial 1 is for state $(3, 3)$:

$$\begin{aligned}
 V^\pi((3, 3)) &\leftarrow 0 + 1(1 + 0.9(0) - 0) \\
 &\leftarrow 1
 \end{aligned}$$

	1	2	3
1	1	1	1
2	0	0	1
3	0	0	1

(a) $N(s)$, after trial 1.

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	1.0

(b) $V^\pi(s)$, after trial 1.

TD learning:

Trial	$\langle \text{state}, \text{reward} \rangle$ series
1	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
2	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{S} \langle (2, 2), 0 \rangle \xrightarrow{N} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
3	$\langle (1, 1), 0 \rangle \xrightarrow{S} \langle (2, 1), 0 \rangle \xrightarrow{N} \langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$

Trial 2

The only non-zero update in this trial is for state $(2, 3)$:

$$\begin{aligned}
 V^\pi((2, 3)) &\leftarrow 0 + \frac{1}{2}(0 + 0.9(1) - 0) \\
 &\leftarrow 0.45
 \end{aligned}$$

	1	2	3
1	2	3	2
2	0	1	2
3	0	0	2

(c) $N(s)$, after trial 2.

	1	2	3
1	0	0	0
2	0	0	0.45
3	0	0	1.0

(d) $V^\pi(s)$, after trial 2.

TD learning:

Trial	$\langle \text{state}, \text{reward} \rangle$ series
1	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
2	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{S} \langle (2, 2), 0 \rangle \xrightarrow{N} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
3	$\langle (1, 1), 0 \rangle \xrightarrow{S} \langle (2, 1), 0 \rangle \xrightarrow{N} \langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$

Now we have two non-zero updates during this trial; first state $(1, 3)$, and then another update to state $(2, 3)$:

$$\begin{aligned}
 V^\pi((1, 3)) &\leftarrow 0 + \frac{1}{3}(0 + 0.9(0.45) - 0) \\
 &\leftarrow 0.135 \\
 V^\pi((2, 3)) &\leftarrow 0.45 + \frac{1}{3}(0 + 0.9(1) - 0.45) \\
 &\leftarrow 0.6
 \end{aligned}$$

	1	2	3
1	4	4	3
2	1	1	3
3	0	0	3

(e) $N(s)$, after trial 3.

	1	2	3
1	0	0	0.135
2	0	0	0.6
3	0	0	1.0

(f) $V^\pi(s)$, after trial 3.

Active Reinforcement Learning

Here we need to choose best action

Model to be learnt from large number of trials.

The agent needs to compute transition model for all the actions (ADP algorithm for a passive agent can be used). The optimal policy, satisfying the Bellman equation, can be determined (for a regular Markov decision process)

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' | s, a) U(s') .$$

Exploration vs exploitation

In addition to possessed knowledge agent must perform random actions in all possible states for a fraction of time and other time use optimal policy.

Slows down the process but good for less explored states-action pair

$$U^+(s) \leftarrow R(s) + \gamma \max_a f \left(\sum_{s'} P(s' | s, a) U^+(s'), N(s, a) \right) .$$

Large value assigned to less explored state-action pair

$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

Q-learning

Optimal policy without current policy (model free). This is an alternative to TD learning

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma \max_a Q(s', a') - Q(s, a))$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

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

$S \leftarrow S'$

 until S is terminal

State-Action-Reward-State-Action (SARSA)

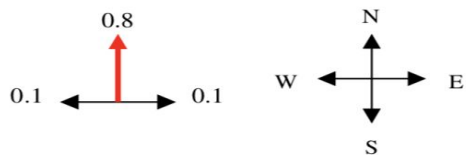
Variant of Q-learning, on-policy

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma Q(s', a') - Q(s, a))$$

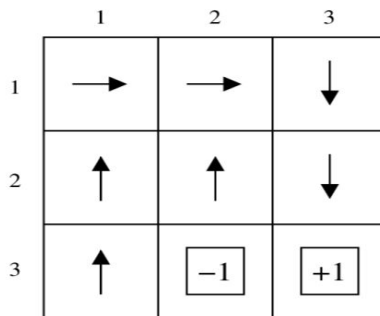
SARSA uses s, a, r, s', a' .

While Q-learning chooses the best action on state reached by action a , SARSA considers which action was in fact selected.

Example - Mini grid world



(a) Transition model of 3x3 world.



(b) Optimal policy π of 3x3 world.

Trial	$\langle \text{state}, \text{reward} \rangle$ series
1	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
2	$\langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{S} \langle (2, 2), 0 \rangle \xrightarrow{N} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$
3	$\langle (1, 1), 0 \rangle \xrightarrow{S} \langle (2, 1), 0 \rangle \xrightarrow{N} \langle (1, 1), 0 \rangle \xrightarrow{E} \langle (1, 2), 0 \rangle \xrightarrow{E} \langle (1, 3), 0 \rangle \xrightarrow{S} \langle (2, 3), 0 \rangle \xrightarrow{S} \langle (3, 3), 1 \rangle$

Example of Q-learning

$$Q(s, a) \leftarrow Q(s, a) + \alpha(N(s, a))(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) \quad (4)$$

where $N(s, a)$ denotes the number of times we have been in state s and taken action a , and $\alpha(n)$ is a function that increases and n decreases. Again we use $\alpha(n) = \frac{1}{n}$.

Initialize all $Q(s, a)$ to 0. (This can also be done randomly.) Then, for each trial, we perform the update equation for each $\langle s, a, s' \rangle$ tuple. We also keep a table of $N(s, a)$ counts.

Trial 1

As we said above, we will learned in this trial that state $(3, 3)$ is a terminal state with reward 1. Therefore, we will set the value of the Q-function for $(3, 3)$ to 1, for all a . And since all other utilities are zero, this is the only non-trivial update while performing updates for trial 1.

$$\begin{aligned} Q((3, 3), N) &\leftarrow 1 \\ Q((3, 3), E) &\leftarrow 1 \\ Q((3, 3), S) &\leftarrow 1 \\ Q((3, 3), W) &\leftarrow 1 \end{aligned}$$

The resulting $Q(s, a)$ is shown in Figure 4(b).

Trial 2

The only non-zero update in this trial is for state-action pair $\langle (2, 3), S \rangle$:

$$\begin{aligned} Q((2, 3), S) &\leftarrow 0 + \frac{1}{2}(0 + 0.9(1) - 0) \\ &\leftarrow 0.45 \end{aligned}$$

The resulting $Q(s, a)$ is shown in Figure 4(d).

Example of Q-learning - contd

	1	2	3
1	0 0 1 0	0 0 1 0	0 0 0 1
2	0 0 0 0	0 0 0 0	0 0 0 1
3	0 0 0 0	0 0 0 0	1

(a) $N(s, a)$, after trial 1.

	1	2	3
1	0 0 0 0	0 0 0 0	0 0 0 0
2	0 0 0 0	0 0 0 0	0 0 0 0
3	0 0 0 0	0 0 0 0	1 1 1 1

(b) $Q(s, a)$, after trial 1.

	1	2	3
1	0 0 2 0	0 0 2 1	0 0 0 2
2	0 0 0 0	1 0 0 0	0 0 0 2
3	0 0 0 0	0 0 0 0	2

(c) $N(s, a)$, after trial 2.

	1	2	3
1	0 0 0 0	0 0 0 0	0 0 0 0
2	0 0 0 0	0 0 0 0.45	0 0 0 1
3	0 0 0 0	0 0 0 1	1 1 1 1

(d) $Q(s, a)$, after trial 2.

	1	2	3
1	0 0 3 1	0 0 3 1	0 0 0 3
2	0 0 0 0	0 0 0 0	0 0 0 3
3	0 0 0 0	0 0 0 0	3

(e) $N(s, a)$, after trial 3.

	1	2	3
1	0 0 0 0	0 0 0 0	0 0 0 0.135
2	0 0 0 0	0 0 0 0.6	0 0 0 1
3	0 0 0 0	0 0 0 1	1 1 1 1

(f) $Q(s, a)$, after trial 3.

Figure 4: Learned utilities, using Q-learning.

Trial 3

As in passive TD learning, now we have two non-zero updates during this trial; first state (1, 3), and then another update to state (2, 3):

$$\begin{aligned}
 Q((1, 3), S) &\leftarrow 0 + \frac{1}{3}(0 + 0.9(0.45) - 0) \\
 &\leftarrow 0.135 \\
 Q((2, 3), S) &\leftarrow 0.45 + \frac{1}{3}(0 + 0.9(1) - 0.45) \\
 &\leftarrow 0.6
 \end{aligned}$$

Approximate Q-learning

In a large state space, state-space or state-action space(Q values) can be estimated (mostly linear approximation) using independent features of states(-action pair).

Example: distance between two points in euclidean space, velocity of two agents etc

Tractable

$$\hat{U}_{\theta}(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \cdots + \theta_n f_n(s) .$$

Utility value

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

Q-value

Improve weights

Gradient descent

$$\hat{y} = \sum_k w_k f_k(x)$$

$$error(w) = \frac{1}{2} \left(y - \sum_k w_k f_k(x) \right)^2$$

$$\frac{\partial error(w)}{\partial w_m} = - \left(y - \sum_k w_k f_k(x) \right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x) \right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(\underbrace{r + \gamma \max_a Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{estimate}} \right) f_m(s, a)$$

Approximate Q-update

Improve weights

Utility

$$w_i \leftarrow w_i + \alpha [R(s) + \gamma \hat{U}_w(s') - \hat{U}_w(s)] \frac{\partial \hat{U}_w(s)}{\partial w_i}$$

Q-value

$$\hat{U}_w(x, y) = w_0 + w_1 x + w_2 y$$

$$w_i \leftarrow w_i + \alpha [R(s) + \gamma \max_{a'} \hat{Q}_w(s', a') - \hat{Q}_w(s, a)] \frac{\partial \hat{Q}_w(s, a)}{\partial w_i}$$

$$\hat{Q}_w(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \cdots + w_n f_n(s, a)$$

Example of approximate Q-values: 1

We would like to use a Q-learning agent for Pacman, but the state size for a large grid is too massive to hold in memory. To solve this, we will switch to feature-based representation of Pacman's state.

1. Say our two minimal features are the number of ghosts within 1 step of Pacman (F_g) and the number of food pellets within 1 step of Pacman (F_p). You'll notice that these features depend only on the state, not the actions you take. Keep that in mind as you answer the next couple of questions. For this pacman board:



Extract the two features (calculate their values).

$$f_g = 2, f_p = 1$$

2. With Q Learning, we train off of a few episodes, so our weights begin to take on values. Right now $w_g = 100$ and $w_p = -10$. Calculate the Q value for the state above.

First of all, the Q value will not depend on what action is taken, because the features we extract do not depend on the action, only the state.

$$Q(s, a) = w_g * f_g + w_p * f_p = 100 * 2 + -10 * 1 = 190$$

Example of approximate Q-values: 1

3. We receive an episode, so now we need to update our values. An episode consists of a start state s , an action a , an end state s' , and a reward r . The start state of the episode is the state above (where you already calculated the feature values and the expected Q value). The next state has feature values $F_g = 0$ and $F_p = 2$ and the reward is 50. Assuming a discount of $\gamma = 0.5$, calculate the new estimate of the Q value for s based on this episode.

$$\begin{aligned}Q_{new}(s, a) &= R(s, a, s') + \gamma * \max_{a'} Q(s', a') \\&= 50 + 0.5 * (100 * 0 + -10 * 2) \\&= 40\end{aligned}$$

4. With this new estimate and a learning rate (α) of 0.5, update the weights for each feature.

$$\begin{aligned}w_g &= w_g + \alpha * (Q_{new}(s, a) - Q(s, a)) * f_g(s, a) = 100 + 0.5 * (40 - 190) * 2 = -50 \\w_p &= w_p + \alpha * (Q_{new}(s, a) - Q(s, a)) * f_p(s, a) = -10 + 0.5 * (40 - 190) * 1 = -85\end{aligned}$$

Note that now the weight on ghosts is negative, which makes sense (ghosts should indeed be avoided). Although the weight on food pellets is now also negative, the difference between the two weights is now much lower.

Approximate Q-learning: 2

Consider the following MDP: We have infinitely many states $s \in \mathbb{Z}$ and actions $a \in \mathbb{Z}$, each represented as an integer. Taking action a from state s deterministically leads to new state $s' = s + a$ and reward $r = s - a$. For example, taking action 3 at state 1 results in new state $s' = 1 + 3 = 4$ and reward $r = 1 - 3 = -2$.

We perform approximate Q-Learning, with features and initialized weights defined below.

Feature	Initial Weight
$f_1(s, a) = s$	$w_1 = 1$
$f_2(s, a) = -a^2$	$w_2 = 2$

(a) [?? pts] Write down $Q(s, a)$ in terms of w_1 , w_2 , s , and a .

$$Q(s, a) = w_1 * s - w_2 * a^2$$

(b) [?? pts] Calculate $Q(1, 1)$.

$$Q(1, 1) = w_1 f_1(1, 1) + w_2 f_2(1, 1) = 1 * 1 - 2 * 1^2 = -1$$

Approximate Q-learning: 2

- (c) [?? pts] We observe a sample (s, a, r, s') of $(1, 1, 0, 2)$. Assuming a learning rate of $\alpha = 0.5$ and discount factor of $\gamma = 0.5$, compute new weights after a single update of approximate Q-Learning.

$$diff = (0 + 0.5 * \max_a Q(2, a)) - (-1)$$

$$diff = (0.5 \max_a (2 - 2 * a^2)) + 1$$

$$diff = (0.5(2 + 2 * \max_a (-a^2))) + 1$$

$$diff = (0.5(2 + 2 * 0)) + 1$$

$$diff = 1 + 1 = 2$$

$$w_1: \underline{1 + 0.5 * 2 * 1 = 2}$$

$$w_2: \underline{2 + 0.5 * 2 * -1^2 = 1}$$

- (d) [?? pts] Compute the new value for $Q(1,1)$.

$$\underline{Q(1,1) = w_1 * 1 + w_2 - 1^2 = 2 * 1 + 1 * -1^2 = 1}$$

More examples on Reinforcement learning

https://inst.eecs.berkeley.edu/~cs188/fa18/mt1_prep.html