on-policy update

Assum initially $Q(s, a) = 0, \forall s, a$

equiprobable policy

$$Q(s_t, A_t) \leftarrow Q(s_t, A_t) + \alpha (R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, A_t))$$

$$Q(s_t, A_t) = \sum R_{t+1}$$

new $Q$ = Bellman 2.20

free backup $Q = \delta[R_{t+1} + \frac{1}{3} R_{t+2}]$
Types of RL algorithms

- Model-based
- Model-free
Types of Models

• Distributional model
• Sample model

- actually have all the probs given $s,A \rightarrow \text{prob}(s',A)$

more constrained model
given an $(s,A)$ it returning a sample $s',A$

given a distribution model, easy to come up with 1 samples
What is planning?

Direct RL

w/0 model

use estimates of state values

take actions in env.

w/o model

given model

may be imperfect

agent

env

state

take imaginary actions

get imaginary rewards & states

update policy

imagine taking an action

taking an action

 RL
Random-sample one-step tabular Q-planning

Loop forever:
1. Select a state, \( S \), and an action \( A \) at random
2. Send \( S, A \) to a sample model and obtain a sample next reward \( R \), and a sample next state, \( S' \)
3. Apply one-step tabular Q-learning to \( S, A, R, S' \)
   \[
   Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]
   \]
Online planning

value/policy

model

experience

model learning

planning

updating Q-learning

Q-value

direct RL

acting in real environment

(s', r')
Dyna Architecture
Tabular Dyna-Q Overview

• Direct RL method: one-step tabular Q-learning
• Model-learning method:
  • Assumes environment is deterministic
  • Table-based
  • Given $A_t, S_t \rightarrow R_{t+1}, S_{t+1}$, stores
    $\text{model}[(S_t, A_t)] = (R_{t+1}, S_{t+1})$
Tabular Dyna-Q Algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $a, s$

Loop forever:
1. $S \leftarrow$ current (nonterminal) state
2. $A \leftarrow \epsilon-greedy(S, Q)$
3. Take action $A$; observe resultant reward $R$ and state $S'$
4. $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
5. $Model(S', A) \leftarrow (R, S')$ (assumes deterministic environment)
6. Loop repeat $n$ times
   - $S \leftarrow$ random previously observed state
   - $A \leftarrow$ random action previously taken in $S$
   - $R, S' \leftarrow Model(S, A)$
   - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
Dyna Maze (Figure 8.2)
Dyna Maze (Figure 8.3)

WITHOUT PLANNING (n=0)

WITH PLANNING (n=50)

1/2 way through 2nd episode
model-free & source of uncertain: Q-value
model-based + additional source of uncertain: model
When the Model is Wrong: Optimistic Model
When the Model is Wrong: Pessimistic Model
Dyna-Q+: Dyna-Q + heuristics for encouraging

- Provide an implicit reward to exploring stale transitions

\[ Q(S, A) \leftarrow Q(S, A) + \alpha [R + k \sqrt{\tau(S, A)} + \gamma \max_a Q(S', a) - Q(S, A)] \]

- Allow actions that had never been tried from a state to be considered in planning (initial model was that such an action led back to the same state with a reward of 0)
When the Model is Wrong: Optimistic Model
When the Model is Wrong: Pessimistic Model
Dyna-Q tries all state-action pairs uniformly. Is there a better way?
Prioritized Sweeping (Det. Env.)

Initialize $Q(s, a)$ and $Model(s, a)$ for all $a, s$, $PQueue$ to empty
Loop forever:
1. $S \leftarrow$ current (nonterminal) state; 2. $A \leftarrow \text{policy}(S, Q)$
3. Take action $A$; observe resultant reward $R$ and state $S'$
4. $Model(S, A) \leftarrow (R, S')$ (assumes deterministic environment)
5. $P = R + \gamma \max_a Q(S', a) - Q(S, A)$
6. If $P > \Theta$, insert $(S, A)$ into $PQueue$ with priority $P$
7. Loop repeat $n$ times while $PQueue$ is not empty
   a. $S, A = \text{first}(PQueue); R, S' \leftarrow Model(S, A)$
   b. $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
   c. Loop for all $\bar{S}, \bar{A}$ predicted to lead to $S$:
      i. $\bar{R} = \text{pred. reward for } \bar{S}, \bar{A}, S$
      ii. $P = \bar{R} + \gamma \max_a Q(S', a) - Q(\bar{S}, \bar{A})$
      iii. If $P > \Theta$, insert $(\bar{S}, \bar{A})$ into $PQueue$ with priority $P$