ACTIVE REINFORCEMENT LEARNING

① Generally, we don't just want to evaluate the expected utility of a particular policy, but rather we want to find the policy with the best expected utility. One step in this direction is to rewrite TDLEARNING in terms of the expected utility $U(q,\sigma)$ of <u>taking action σ</u> in state q, rather than the expected utility of <u>policy π</u> in state q:

TDLEARNING(Q, Σ, R, π):

set $U(q,\sigma) = 0$, $n_q = 1$ for all $(q,\sigma) \in Q \times \Sigma$

repeat:

observe transition $q \xrightarrow{\pi(q)} q'$

update $U(q, \pi(q)) = U(q, \pi(q)) + \alpha(n_q)\left(R_0(q) + \gamma \cdot U(q', \pi(q')) - U^{\pi}(q) \right)$

n_q += 1

Active Reinforcement Learning

(2) Rather than averaging

$$R_0(q) + \gamma \cdot U(q', \pi(q'))$$

for each transition $q \xrightarrow{\pi(q)} q'$ we observe, we now want to average

$$\underbrace{R_0(q)}_{\text{reward for state } q} + \underbrace{\gamma \max_{\sigma' \in \Sigma} U(q', \sigma')}_{\text{utility of the observed next state, given we perform the best action in that state}}$$

for each transition $q \xrightarrow{\sigma} q'$ we observe.

---

(3) In TDLearning, we collect observations by rigidly following a fixed policy π. Now we have the flexibility to choose different actions ("experimentation") to observe their outcomes. This gives us the following "active" algorithm:

set $U(q,\sigma) = 0$, $n_{q,\sigma} = 1$ for all $(q,\sigma) \in Q \times \Sigma$

$q = q_0$

repeat:

  choose action σ

  observe transition $q \xrightarrow{\sigma} q'$

  $U(q,\sigma) \mathrel{+}= \alpha(n_{q,\sigma}) \left[\left(R_0(q) + \gamma \max_{\sigma'} U(q', \sigma') \right) - U(q,\sigma) \right]$

  $n_{q,\sigma} \mathrel{+}= 1$

Active Reinforcement Learning

④ This algorithm is called Q-learning. The main piece we left unspecified is how to "choose action σ."

A simple approach is:

with prob p:

  choose random action σ ("exploration")

otherwise:

  choose $\arg\max_{\sigma} U(q, \sigma)$ ("exploitation")

Active Reinforcement Learning

(5) Generally in blackjack, the player gets to see one of the dealer's two cards (called the "up card"). This gives a player additional information on which to base a decision. For instance, if the player has a total of 14, but the dealer's up card is a 5, then it might be good to STAND, knowing that the dealer is likely to draw a 10 and then be forced to HIT on 15 (which is likely to result in a bust). We could make a state machine for this more typical version of blackjack:

$$Q = \{\text{START}, \text{WIN}, \text{LOSE}, \text{DRAW}\} \cup \{A_{k,u} \mid k \in [2,21], u \in [1,10]\}$$

$$\Sigma = \{\text{DEAL}, \text{HIT}, \text{STAND}\}$$

$$\begin{aligned}
\Delta = &\{(\text{START}, \text{DEAL}, A_{k,u}) \mid k \in [2,20], u \in [1,10]\} \\
&\cup \{(A_{k,u}, \text{STAND}, q) \mid k \in [2,20], u \in [1,10], q \in \{\text{WIN}, \text{LOSE}, \text{DRAW}\}\} \\
&\cup \{(A_{21,u}, \text{STAND}, q) \mid u \in [1,10], q \in \{\text{WIN}, \text{DRAW}\}\} \\
&\cup \{(A_{k,u}, \text{HIT}, A_{k+m,u}) \mid k \in [2,20], u, m \in [1,10], k+m \leq 21\} \\
&\cup \{(A_{k,u}, \text{HIT}, \text{LOSE}) \mid k \in [12,21], u \in [1,10]\}
\end{aligned}$$

$$q_0 = \text{START}$$

$$F = \{\text{WIN}, \text{LOSE}, \text{DRAW}\}$$

we assume aces always count as 1

here, [i, j] denotes the integers from i to j (inclusive)

⑥ The trouble with this new state machine is that it has many more states than the "up card ignorant" formulation:

"upcard ignorant"

$4 + 20$

$= 24$

"upcard informed"

$4 + 20 \cdot 10$

$= 204$

---

⑦ This means that QLEARNING is now trying to estimate $U(q, \sigma)$ for approximately 400 state-action pairs (~200 states, ~2 actions/state) rather than approximately 40 state-action pairs, which intuitively will take much more experimentation and exploitation.

For instance, to get a good estimate of $U(A_{6,7}, \text{HIT})$, we have to reach a card total of 6 with a dealer upcard of 7 enough times to get a reliable sense of its utility.

So it will take longer to train. This problem only worsens as we add more information about the current state (e.g. card-counting statistics, "hard" or "soft" 16, etc.)

Active Reinforcement Learning

(8) One way to get reinforcement learning to scale to larger state spaces is to assume that each utility is a sum of weights:

$$U(A_{14,5}, \text{HIT}) = \theta^{total}_{14,\text{HIT}} + \theta^{up}_{5,\text{HIT}}$$

this is shared!

$$U(A_{16,5}, \text{HIT}) = \theta^{total}_{16,\text{HIT}} + \theta^{up}_{5,\text{HIT}}$$

$$U(A_{16,10}, \text{HIT}) = \theta^{total}_{16,\text{HIT}} + \theta^{up}_{10,\text{HIT}}$$

this is shared!

For example, we are assuming that the expected reward of HITTING given a card total of 16 and an upcard of 5 can be expressed as the sum of

- the "value" of HITTING given a card total of 16 $(\theta^{total}_{16,\text{HIT}})$
- the "value" of HITTING given an upcard of 5. $(\theta^{up}_{5,\text{HIT}})$

Generally:

$$U(A_{k,u}, \sigma) = \sum_{k',u'} \theta^{total}_{k'} \delta^{total}_{k'}(A_{k,u}) + \theta^{up}_{u'} \delta^{up}_{u'}(A_{k,u})$$

where:

$$\delta^{total}_{k'}(A_{k,u}) = \begin{cases} 1 & \text{if } k=k' \\ 0 & \text{otherwise} \end{cases} \qquad \delta^{up}_{u'}(A_{k,u}) = \begin{cases} 1 & \text{if } k=k' \\ 0 & \text{otherwise} \end{cases}$$

ACTIVE REINFORCEMENT LEARNING

(9) The advantage of this assumption is that we can generalize learning across states. For instance, it is generally bad to HIT on 20 (regardless of the upcard). Even if we don't have many examples of state $A_{20,8}$ in our data so far, if we've seen several states of the form $A_{20,u}$, then we can still know not to HIT on $A_{20,8}$ because the weight $\theta^{total}_{20,HIT}$ will be low.

Rather than learning ~400 independent utilities $U(q,\sigma)$, we need to estimate 40 $\theta^{total}_{k,\sigma}$ weights and 20 $\theta^{up}_{u,\sigma}$ weights. So convergence (if it happens) should be faster.

ACTIVE REINFORCEMENT LEARNING

10) The modification to QLEARNING is minimal:

QLEARNING

$\boxed{\text{set } \theta_i = 0 \text{ for all } i \in [1, I]}$

set $n_{q,\sigma} = 1$ for all $(q,\sigma) \in Q \times \Sigma$

$q = q_0$

repeat:

choose action σ

observe transition $q \xrightarrow{\sigma} q'$

$$\boxed{\theta_i} \mathrel{+}= \alpha(n_{q,\sigma})\left[\left(R_0(q) + \gamma \max_{\sigma'} U(q',\sigma')\right) - U(q,\sigma)\right] \quad \boxed{\text{if } \delta_i(q) = 1}$$

$n_{q,\sigma} \mathrel{+}= 1$

---

11) Why would such a modification work? Well, it doesn't always. But the rationale is this. If the weights θ_i have been set perfectly, then:

$$U(q,\sigma) = R_0(q) + \gamma \max_{\sigma'} U(q',\sigma')$$

$$\Rightarrow \left(R_0(q) + \gamma \max_{\sigma'} U(q',\sigma')\right) - U(q,\sigma) = 0$$

Thus, we won't update the weights if they're correct.

12) If the weights that compose $U(q,\sigma)$ are wrong, then either $U(q,\sigma)$ is underestimated:

$$\sum_{i \mid \delta_i(q)=1} \theta_i < R_0(q) + \gamma \max_{\sigma'} U(q',\sigma')$$

$$\Rightarrow \left(R_0(q) + \gamma \max_{\sigma'} U(q',\sigma')\right) - \sum_{i \mid \delta_i(q)=1} \theta_i \text{ is positive.}$$

So we increase the weights θ_i, or $U(q,\sigma)$ is overestimated, so we (similarly) decrease the weights θ_i.

However, there are no convergence guarantees. Sometimes however, it works spectacularly.