On-Policy & Off-Policy Learning

11 January 2021 12:18

Q-Learning

An agent tries to learn the optimal policy from its history of interaction with the environment. A history of an agent is a sequence of state-action rewards:

$$\langle s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, s_4 \ldots \rangle$$

which means that the agent was in state **s0** and did action **a0**, which resulted in it receiving reward **r1** and being in state **s1**, and then it did action **a1**, and so on.

We treat this history of interaction as a sequence of experiences, where an experience is a tuple

$$\langle s, a, r, s' \rangle$$

which means that the agent was in state **s**, took action a and then received reward **r**, ending up in state **s'**. These experiences will be the data from which the agent can learn what to do. As in decision-theoretic planning, the aim is for the agent to maximise its value which is usually the discounted reward.

Remember that **Q*(s, a)** is the expected value (cumulative discounted reward) of doing a in state **s**, and then following the optimal policy.

Q-learning uses temporal differences to estimate the value of **Q*(s, a)**

In Q-learning, the agent maintains a table of **Q[S, A]. Q[s,a]** represents its current estimate of **Q*(s,a)**

An experience provides on data point for the value of **Q(s,a)**. The data point is that the agent received the future value of

$r + \gamma V(s')$, where $V(s') = \max_{a'} Q(s', a')$;

This is the actual current reward plus the discounted estimated future value. This new data point is called a return. The agent can use the temporal difference equation to update its estimate for **Q(s,a)**

$$Q[s,a] \ := \ Q[s,a] + \alpha * \left(r + \gamma \max_{a'} Q[s',a'] - Q[s,a] \right)$$

or, equivalently,

$$Q[s,a] \ := \ (1-\alpha) * Q[s,a] + \alpha * \left(r + \gamma \max_{a'} Q[s',a'] \right).$$

Consider this Q-learning controller. The do(a) line specifies the action the controller commands the body to do, and the reward and the resulting state are percepts the controller receives from the body.

```
controller Q-learning($S, A, \gamma, \alpha$)
    Inputs
        $S$ is a set of states
        $A$ is a set of actions
        $\gamma$ the discount
        $\alpha$ is the step size
    Local
        real array $Q[S, A]$
        states $s$, $s'$
        action $a$
    initialize $Q[S, A]$ arbitrarily
    observe current state $s$
    repeat
        select an action $a$
        $do(a)$
        observe reward $r$ and state $s'$
        $Q[s,a] := Q[s,a] + \alpha * (r + \gamma * \max_{a'} Q[s',a'] - Q[s,a])$
        $s := s'$
    until termination
```

The Q-learning leans an approximation of the optimal Q-function as long as the agent explores enough, and there is no bound on the number of times it tries an action in any state (it does not always do the same subset of actions in a state)

Consider the example we looked at early with the {health, sick}, {relax, party} problem.

Example 12.3. *Consider the two-state MDP of Example 9.27. The agent knows there are two states* $\{healthy, sick\}$ *and two actions* $\{relax, party\}$. *It does not know the model and it learns from the* s, a, r, s' *experiences. With a discount,* $\gamma = 0.8$, $\alpha = 0.3$, *and* Q *initially 0, the following is a possible trace (to a few significant digits and with the states and actions abbreviated):*

s	a	r	s'	$Update = (1-\alpha) * Q[s,a] + \alpha(r + \gamma max_{a'} Q[s',a'])$
he	re	7	he	$Q[he,re] = 0.7 * 0 + 0.3 * (7 + 0.8 * 0) = 2.1$
he	re	7	he	$Q[he,re] = 0.7 * 2.1 + 0.3 * (7 + 0.8 * 2.1) = 4.07$
he	pa	10	he	$Q[he,pa] = 0.7 * 0 + 0.3 * (10 + 0.8 * 4.07) = 3.98$
he	pa	10	si	$Q[he,pa] = 0.7 * 3.98 + 0.3 * (10 + 0.8 * 0) = 5.79$
si	pa	2	si	$Q[si,pa] = 0.7 * 0 + 0.3 * (2 + 0.8 * 0) = 0.06$
si	re	0	si	$Q[si,re] = 0.7 * 0 + 0.3 * (0 + 0.8 * 0.06) = 0.014$
si	re	0	he	$Q[si,re] = 0.7 * 0.014 + 0.3 * (0 + 0.8 * 5.79) = 1.40$

With α *fixed, the* Q*-values will approximate, but not converge to, the values obtained with value iteration in Example 9.31. The smaller* α *is, the closer it will converge to the actual* Q*-values, but the slower it will converge.*

Exploration & Exploitation

The Q-learner controller does not specify what the agent should actually do. The agent learns a Q-function that can be used to determine an optimal action. There are two things that are useful for the agent to do:

- exploit the knowledge taht is has found for the current state s by doing one of the actions a that maximises Q[s,a]
- explore in order to build a better estimate of the optimal Q-function; it should sometimes select a different action from the one that it currently thinks is best

Epsilon-Greedy Strategy

Where **0 <= epsilon <= 1** is the explore probability, is to select the greedy action that maximises Q[s,a] all but **epsilon** of the time, and pick a random action epsilon of the time. It is possible to change **epsilon** over time which is intuitively correct since we'd want to act randomly at the start and gradually act more in line with our knowledge as we accumulate more information about the world.
However, this treats all of the actions, apart from the best action, equivalently.

Soft-Max

If there are few seemingly good actions it may be more sensible to select among the good actions to explore those further, rather than expending effort exploring actions that look less promising. One way to do this is to select an action a with a probability depending on the value of **Q[s,a]**. A common method is to use a Gibbs or Boltzmann distribution, where the probability of selecting action **a** in state **s** is proportional to

$$e^{Q[s,a]/\tau}$$

Thus in state **s** the agent selects action a with probability

$$\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}$$

Where tau is the temperature specifying how randomly we should choose values. When t is high, the actions are chosen in almost equal amounts. As the temperature is reduce, we choose the actions with the highest **Q[s,a]** more often. When tau = 0, we choose the best action always.

Optimism in the Face of uncertainty

Initialise the Q-function to values that encourage exploration. If the Q-values are initialized to high values, the unexplored areas will look good, so that a greedy search will tend to explore. This does encourage exploration, but can cause the agent to hallucinate that some state-action pairs are good for a long time, with no real evidence. A state only begins to look bad when all its actions look bad.

In noisy environments, OITFOU can mean that a good action never gets explored more because by random chance, it gets a low Q-value from which it never recovers.

On-Policy & Off-Policy Learning

Q-learning is an off-policy learner.

- ✧ An off-policy learner learns the value of an optimal policy independently of the agent's actions, as long as it explores enough.
- ✧ An off-policy learner can learn the optimal policy even if it is acting randomly.

- A learning agent should, however, try to exploit what it has learned by choosing the best action, but it cannot just exploit because then it will not explore enough to find the best action.
- An off-policy learner does not learn the value of the policy it is following, because the policy it is following includes exploration steps.

However, there may be cases where ignoring what the agent actually does is dangerous: where there are large negative rewards. An alternative is to learn the value of the policy that the agent is actually carrying out, which includes exploration steps, so that it can iteratively be improved.

SARSA

An on-policy reinforcement learning algorithm that estimates the value of the policy being followed.

An experience in SARSA is of the form <s, a, r, s', a'> which means the agent was in state **s** and did action **a**, ending up with reward **r** in state **s**' from which it decided to do **a**'

This provides a new experience to update Q(s,a). The new value that this experience provides is:

$$r + \gamma Q\left(s', a'\right).$$

The Q-values that SARSA computes depend on the current exploration policy which for example may be greedy with random steps. It can find a different policy that Q-learning in situations when exploring incur large penalties. For example, when a robot goes near the top of the stairs, even if this is an optimal policy, it may be dangerous for exploration steps.

SARSA will discover this and adopt a policy that keeps the robot away from the stairs. It will find a policy that is optimal, taking into account the exploration inherent in the policy.

SARSA is useful when deploying an agent that is exploring in the world. If you want to do offline learning, and then use that policy in an agent that does not explore, Q-learning may be more appropriate.