Reinforcement Learning

Trong Truong

November 25, 2018

1 Cumulative reward

Cumulative reward is sometimes called return. The return from time t is the expected reward starting from time t:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \ldots = \sum_{k=0}^{\infty} r_{t+k+1}.$$

We can also add the discount factor:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $0 < \gamma < 1$.

2 Temporal difference update

$$\begin{aligned} V(s) &= V(s) + \alpha(V(s') - V(s)) \\ &= (1-\alpha)V(s) + \alpha V(s'), \end{aligned}$$

essentially a weighted average between the old and the new values.

3 Law of Effect

Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.—Thorndike, 1911, p. 244.

4 Policy Function

Definition 1. A policy π is a probability distribution over actions given states:

$$\pi(a|s) = P[A_t = a|S_t = s].$$

I.e. Given the current state, what is the most likely action to take. A policy fully defines the behavior of an agent.

Given an MDP $M = (S, A, P, R, \gamma)$:

- The state sequence $S_1, S_2, \ldots$ is a Markov process S, P^π. IOW, the policy defines the particular Markov process, given the set of all possible states S and actions A.
- The state reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process S, P^π, R^π, γ, where

$$\mathcal{P}^\pi_{ss'} = \sum_{a\in\mathcal{A}} \pi(a|s)\mathcal{P}^a_{ss'}$$
$$\mathcal{R}^\pi_s = \sum_{a\in\mathcal{A}} \pi(a|s)\mathcal{R}^a_s.$$

5 State value function and action value function

State value function: $V(s)$ is the expected return when starting from state s acting according to policy π :

$$V^\pi(s) = \mathbb{E}_\pi\left[G_t|s_t = s\right].$$

Action value function: $Q(s, a)$ is the value of an action in a given state and acting under a policy π :

$$Q^\pi(s, a) = \mathbb{E}_\pi\left[G_t|s_t = s, a_t = a\right].$$

Note that Q is more specific than V, in the sense that V takes into account all possible actions.

6 Markov Decision Process

Definition 2. A Markov Decision Process is a tuple (S, A, P, R, γ), where S is a finite set of states, A is a finite set of actions, P is a state transition probability matrix, R is a reward function, and γ is a discount factor.

Given a state $s \in S$ and an action $a \in A$, the probability of transitioning into a new state s' is given by $P(s, a)$, the reward of the transition is given by $R(s, a)$. This assumes that the environment is responding deterministically, not stochastically, i.e. taking action a in state s will always bring you to a particular and the same s' every time.

7 Discrete Markov Chain

Definition 3. Given a finite set of states $\Omega = \{X_0, \ldots, X_N\}$, a discrete Markov chain is a stochastic process that satisfies

$$P\left(X_{n+1} = x_{n+1} | X_n = x_n, \ldots, X_0 = x_0\right) = P\left(X_{n+1} = x_{n+1} | X_n = x_n\right),$$

i.e. the present is enough to determine the future.

Question. Are we working with finitely many states? (It's always finite in the real world.)

Notation: the probability of moving from state i to state j at time n is written

$$p_{ij}(n) = P\left(X_{n+1} = j | X_n = i\right).$$

Definition 4. Transition matrix:

$$P(n) = \begin{pmatrix} p_{00}(n) & p_{01}(n) & p_{02}(n) & \ldots & p_{0j}(n) & \ldots \\ p_{10}(n) & p_{11}(n) & p_{12}(n) & \ldots & p_{1j}(n) & \ldots \\ p_{20}(n) & p_{21}(n) & p_{22}(n) & \ldots & p_{2j}(n) & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{i0}(n) & p_{i1}(n) & p_{i2}(n) & \ldots & p_{ij}(n) & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}.$$

Properties:

- $P(n)$ is a square matrix.

- $\sum_j p_{ij}(n) = 1$ for all $j \in \Omega$. Row i gives the probabilities of moving from state i to all states, so they should all add up to 1.
- $p_{ij}(n) \geq 0$.

Note that in this example, the transition matrix is the same for all time steps n : $P = P(n)$.

8 State vector

We can describe the state of a Markov chain with a state vector $\pi^{(n)}$ where $\pi_j^{(n)}$ represents the probability of being in state j at time n.

E.g. Suppose the starting state is $\pi^{(0)} = (1, 0, 0, 0)$, then the next state vector is just the first row of the transition matrix, which is $(0.90, 0.07, 0.02, 0.01)$. In general,

$$\pi^{(n)} = \pi^{(0)} P^n.$$

9 Transient and steady state distribution

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

$$\pi(t) = \pi(0)e^{Qt} = \pi(0)\left(\mathbb{I} + \sum_{k=1}^{\infty} \frac{Q^k t^k}{k!}\right)$$

The steady state distribution (if it exists) can be found by solving

$$\pi Q = 0.$$

10 N-armed bandit

Simple case: only action affects reward. Compared with Contextual bandit where state and action both together affect reward. Full RL problem: action affects state and reward, and state affects reward.

E.g. k slot machines are an example of a k-armed bandit.

Question. What is the diff between having multiple bandits and a k-armed bandit?

11 ϵ-greedy algorithm

Picks the best currently available option without taking into consideration the long-term effect of that decision: randomly pick the best available option with probability $1 - \epsilon$; OTW randomly pick a random action with probability ϵ.

12 Bellman Equation

State-value function can be decomposed into immediate reward plus discounted value of successor state:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi \left(S_{t+1}\right) | S_t = s\right]$$

Similarly for the action-value function:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma Q_\pi \left(S_{t+1}, A_{t+1}\right) | S_t = s, A_t = a\right].$$

13 Notation

NOTE. The environment can be stochastic as well: taking action a at state s might not always get you to the same state s'. E.g.

$$\mathcal{P}^a_{ss'} = \Pr\left(s_{t+1} = s' | s_t = s, a_t = a\right)$$

is the probability of ending up in state s' by taking action a at state s.

Similarly,

$$\mathcal{R}^a_{ss'} = \mathbb{E}\left[r_{t+1} | s_t = s, s_{t+1} = s', a_t = a\right].$$

14 Deriving the Bellman Equation

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}\left[R_t | s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} | s_t = s\right] + \mathbb{E}_{\pi}\left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s\right] \qquad (*)
\end{aligned}$$

The first term is

$$\mathbb{E}_{\pi}\left[r_{t+1} | s_t = s\right] = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'}.$$

And the second term:

$$\mathbb{E}_{\pi}\left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s\right] = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \gamma \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_{t+1} = s'\right].$$

Putting these back in $(*)$ and refactor, we get

$$\begin{aligned}
V^{\pi}(s) &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_{t+1} = s'\right]\right) \\
&= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right].
\end{aligned}$$

Similarly, the Bellman equation for the action value function is

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^{\pi}(s',a')\right].$$

These equations give us an iterative way of calculating value functions: knowing the next state gives us the value of the current state. (So you can calculate backwards.)

15 Alternative forms for the Bellman equations for V and Q

$$
\begin{aligned}
v_\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s'\in S} \mathcal{P}_{ss'}^a v_\pi(s') \right) \\
q_\pi(s,a) &= \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a'\in\mathcal{A}} \pi(a'|s')\, q_\pi(s',a') .
\end{aligned}
$$

Can write the Bellman equation as an induced Markov Reward Process:

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi,$$

with direct solution

$$v_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi.$$

16 Q-Learning Algorithm

The Q-table is initialized to a matrix where the columns correspond to actions and the rows to states. In this table, higher values are better. Given state s and action a, the new Q-value for s, a is

$$
\begin{aligned}
Q_{sa} &\leftarrow Q_{sa} + \alpha(r_{sa} + \gamma \max_{a'} Q_{s'a'} - Q_{sa}) \\
&= (1-\alpha) Q_{sa} + \alpha(r_{sa} + \gamma \max_{a'} Q_{s'a'}) \\
&= (1-\alpha) Q_{sa} + \alpha Q_{\text{target}},
\end{aligned}
$$

where α is the learning rate, γ is the discount factor.

In other words, the new Q-value is an average between the old Q-value Q_{sa} and the reward plus the discounted expected reward in the new state. The value of the action is based on a combination of its previous value and (*) how much reward the action gave us, plus a little bit of future reward.

This latter value (*) can also be interpreted as the target Q-value:

$$Q_{\text{target}} = r_{sa} + \gamma \max_{a'} Q_{s'a'}.$$

A higher α means we're putting more stock on the present and future, and less on the previously learned values; a higher γ means we're putting more stock in the future.

Inefficient for large state spaces.

This looks like a special case of the temporal difference update rule we had before. It's also derived from the Bellman Equation. We should do that.

17 Deep Q-Learning: Q-Learning + Deep Neural Network

Replace the Q-Table / agent with a neural network.

18 Experience replay / replay buffer: separating experience from learning

This solves two things: (1) Reduces forgetting and (2) reduces correlating experience.

(1) because you're constantly learning from old experiences, not just the most recent ones.

(2). Correlating experiences means that if you learn things sequentially, consecutive experiences are highly correlated. This causes you to rely on your correlated bias to react to an experience, instead of treating each experience as independent and explore other more optimal actions.

The general strategy is to separate learning from interaction. First you interact with the environment to gather experience, but you only start learning after you've filled the experience / replay buffer. "This helps avoid being fixated on one region of the state space. This prevents reinforcing the same action over and over." Variety is key!

(Learning V.S. Performing is often called Explore V.S. Exploit. This one is about separating Seeing/Experience from Believing/Learning LOL.)

19 Temporal difference update for DQN

$$\Delta w = \alpha \left[\Big(R + \gamma \max_a Q\left(s', a, w\right) \Big) - Q(s,a,w) \right] \nabla_w Q(s,a,w).$$

Recall how this is similar to the update rule for the Q-Table:

$$\begin{aligned} Q_{sa} &\leftarrow Q_{sa} + \alpha(Q_{\text{target}} - Q_{sa}) \\ &= (1-\alpha)Q_{sa} + \alpha Q_{\text{target}}, \end{aligned}$$

which can be written as

$$\begin{aligned} Q^{\text{new}}_{sa} - Q^{\text{old}}_{sa} &= \alpha(Q_{\text{target}} - Q^{\text{old}}_{sa}) \\ \Delta Q_{sa} &= \alpha(Q_{\text{target}} - Q_{sa}). \end{aligned}$$

This multiply gradient $\nabla_w Q(s,a,w)$ doesn't make sense though. Shouldn't it be a quotient, like this?

$$\Delta w = \frac{\alpha\left[(R + \gamma \max_a Q\left(s', a, w\right)) - Q(s,a,w)\right]}{\nabla_w Q(s,a,w)},$$

because $\nabla_w Q(s,a,w)$ should look like $\frac{\Delta Q}{\Delta w}$.

20 Optimal state and action-value function

$v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s).$$

Similarly for the action-value function

$$q_*(s,a) = \max_\pi q_\pi(s,a).$$

21 Partial ordering of policies

Define $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \forall s$. Define similar ordering on action value functions?

Theorem 5 (An optimal policy exists). *For any MDP,*

- *There exists an optimal policy π_* s.t. $\pi_* \geq \pi$ for all π.*
- *All optimal policies achieve the optimal value function: $v_{\pi_*}(s) = v_*(s)$.*
- *All optimal policies achieve the optimal action value function: $q_{\pi_*}(s,a) = q_*(s,a)$.*

22 Finding an optimal policy

Knowing the optimal action value function gives us a deterministic optimal policy:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \operatorname{argmax}_{a \in A} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

23 Bellman optimality equations

$$v_*(s) = \max_a q_*(s,a)$$
$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s')$$

Putting those together we get the Bellman optimality equation for v_* :

$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') .$$

Similarly for the optimal action:

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a') .$$

24 Policy Gradient

Another name for learning the policy (the best action to take in any given state), as opposed to learning the value of each state (and the best state) for the agent to be in. Can combine both approaches: learn both the best states and the best actions to take in each state?

25 Negative log-likelihood and cross-entropy loss function

$$\text{Loss} = -A \log(\pi),$$

where A represents "advantage", a measure of how much better an action was compared to a base line. π is the policy, in this case corresponds to the chosen action's weight. Intuitively, a better action decreases loss and vice versa.

A more general version for binary classification:

$$C = -\frac{1}{n} \sum_x \left(\ln a_y^L \right)$$

Even more general is the cross-entropy loss for multi-class classification problems:

$$C = -\frac{1}{n} \sum_x (y \ln a + (1-y) \ln(1-a))$$