1 Temporal-Difference Learning

TD learning is a learning method that's specialized for prediction learning (the scalable model-free learning). TD learning is learning a prediction from another, later, learned prediction (i.e., learning a guess from a guess). The TD error is the difference between two predictions, the temporal difference; otherwise TD learning is the same as supervised learning, backpropagating the error. You can think one step TD (make a prediction, wait one step, see the result) as traditional supervised learning (supervisor tells you the result after one step).

Note prediction problem means policy evaluation, control problem means finding π_*.

TD combines some of the features of both MC and DP. TD does not require a model and can learn from interactions (like MC), and TD can bootstrap, thus learn online (without waiting till the end of episodes) (like DP).

TD(λ) unifies DP, MC, TD.

Prediction problem = policy evaluation (i.e., estimating v_π for a given π)

Control problem = finding an optimal policy

1.1 TD Prediction

Definition 1.1 ***constant-α MC:***

$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{[G_t - V(S_t)]}_{\textit{MC error}}$$

Unlike MC which has to wait until the end of an episode to update, TD only has to wait one time step.

Definition 1.2 ***TD(0)**, or one-step TD:*

$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]}_{\textit{TD error}}$$

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Init $V(s)\ \forall s \in S+, V(term.) = 0$
Loop for each episode:
  Init S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

- MC target is an estimate, because a sample return is needed for real expected return
- DP target is an estimate, because the next state value is unknown and the current estimate is used
- TD target is an estimate because (1) samples the expected values; (2) and uses current estimate V instead of the true v_π

TD methods combine the sampling of MC with the bootstrapping of DP. TD and MC updates are *sample update* because they involve looking ahead to a sample successor state.

- sample update: based on a single sample sample successor
- expected update: a complete distribution of all possible successors

Definition 1.3 ***TD error***:

$$\delta_t \doteq \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\textit{TD Target}} - V(S_t)$$

MC error can be written as a sum of TD errors (think of it this way, TD updates immediately at each time step, MC updates after an episode (which include a lot of timesteps)

$$\begin{aligned}\text{MC error} = G_t - V(S_t) &= \underbrace{R_{t+1} + \gamma G_{t+1}}_{G_t} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2}) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k\end{aligned}$$

1.2 Advantages of TD Prediction Methods

TD methods have an advantage over DP methods in that they do not require a model of the environment.

TD methods over MC methods is that they are naturally implemented in an online, fully incremental fashion.

TD converges faster than MC.

1.3 Optimality of TD(0)

Definition 1.4 ***batch updating****: updates are made only after processing each complete batch of training data.*

- max-likelihood estimate: find the parameter value whose probability of generating the data is greatest
- certainty-equivalence estimate: equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximated

In batch form, TD(0) is faster than MC methods because it computes the

true certainty-equivalence estimate.

1.4 Sarsa: On-policy TD control

This week, you will learn about using temporal difference learning for control, as a generalized policy iteration strategy. You will see three different algorithms based on bootstrapping and Bellman equations for control: Sarsa, Q-learning and Expected Sarsa. You will see some of the differences between the methods for on-policy and off-policy control, and that Expected Sarsa is a unified algorithm for both. You will implement Expected Sarsa and Q-learning, on Cliff World.

We consider transitions from state-action pair to state-action pair.

Definition 1.5 ***SARSA****,*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

If S_{t+1} *is terminal, then* $Q(S_{t+1}, A_{t+1}) = 0$*.*

Since the update rule uses every element of the quintuple of events,

$$(S_t, A_t, R, S_{t+1}, A_{t+1})$$

, that make up a transition from one state-action pair to the next. Hence, the algorithm name *SARSA*.

SARSA (on-policy TD control for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Init $Q(s, a)$, $\forall s \in \S^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:

Init S

Choose A from S using policy derived from Q (eg ϵ-greedy)

Loop for each step of episode:

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (eg ϵ-greedy)

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

$S \leftarrow S'$; $A \leftarrow A'$

until S is terminal

The reason that SARSA is on-policy is that it updates its Q-values using the Q-value of the next state S' and the current policy's action A'. It estimates the return for state-action pairs assuming the current policy continues to be followed.

1.5 Q-learning: Off-policy TD control

Definition 1.6 ***Q-learning***,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Init $Q(s, a)$, $\forall s \in \S^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:

  Init S

  Loop for each step of episode:

    Choose A from S using policy derived from Q (eg 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

The reason that Q-learning is off-policy is that it updates its Q-values using the Q-value of the next state S' and the greedy action a. In other words, it estimates the *return* (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it's not following a greedy policy.

	SARSA	Q-learning
Choosing A'	π	π
Updating Q	π	b

where π is, for example, a ϵ-greedy policy ($\epsilon > 0$ with exploration), and b is a greedy policy (e.g. $\epsilon = 0$, no exploration)

1. Given that Q-learning is using different policies for choosing next action A' and updating Q. In other words, it is trying to evaluate π while following another policy b, so it's an off-policy algorithm
2. In contrast, SARSA follows π all the time, hence it is an on-policy algorithm.

1.6 Expected Sarsa

Expected SARSA like Q-learning except that instead of the maximum over next state-action pairs, it uses the expected value (taking into account how likely each action is under the current policy).

Definition 1.7 ***Expected SARSA***,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \mathop{\mathbb{E}}_{\pi}[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Expected SARSA moves deterministically in the same direction as SARSA moves in expectation. Expected SARSA is more complex computationally than SARSA, but in return, it eliminates the variance due to the random selection of A_{t+1}.

1.6.1 TD control and Bellman equations

TD control algorithms are based on Bellman equations.

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s', a'))$$

SARSA, on-policy, uses the sample based version of the Bellman equation $R + \gamma Q(S_{t+1}, A_{t+1})$, learns q_π.

Expected SARSA, on/off-policy, uses the same Bellman equation as SARSA, but samples it differently. It takes an expectation over the next action values. $R + \gamma \sum_{a'} \pi(a'|S_{t+1})Q(S_{t+1}, a')$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma max_{a'} q_*(s', a'))$$

Q-learning, off-policy, uses the Bellman optimality equation $R+\gamma max_{a'}Q(S_{t+1}, a')$, it learns q_*.

SARSA can do better than Q-learning when performance is online, because on-policy control methods account for their own exploration.

1.7 Learning Objectives (UA RL MOOC)

Lesson 1: Introduction to Temporal Difference Learning

1. Define temporal-difference learning

We want to update the return, but not want to wait till the end of the upate. We want to incrementally update. TD is a way to incrementally estimate the return through bootstrapping

2. Define the temporal-difference error

$$\delta_t \doteq \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD Target}} - V(S_t)$$

3. Understand the TD(0) algorithm

see chap 6.1

Lesson 2: Advantages of TD

4. Understand the benefits of learning online with TD

We can update our estimate at each time step.

5. Identify key advantages of TD methods over Dynamic Programming and Monte Carlo methods

Unlike DP, TD can can directly learn from experience. Unlike MC, TD can update the values on every step TD also converges faster than MC

6. Identify the empirical benefits of TD learning

TD converges faster to a lower final error in the random walk environment.

7. TD can be used both in continuing tasks and episodic tasks (because TD

updates at every time step!)

8. Both TD(0) and MC methods converge to the true value function asymptotically, given that the environment is Markovian.

9. TD and MC use sample updates, DP uses expected updates.

10. Suppose we have current estimates for the value of two states: $V(A) = 1, V(B) = 1$ in an episodic setting. We observe the following trajectory: A, 0, B, 1, B, 0, T where T is a terminal state. Apply TD(0) with step-size, $\alpha = 1$, and discount factor, $\gamma = 0.5$. What are the value estimates for state A and state B at the end of the episode?

Solution: My mistake was that I thought I need to update all the way to the end, then backpropagate the state value back. Actually, here are the transition pairs, (S, R, S') = $(A, 0, B)$, $(B, 1, B)$, and $(B, 0, T)$. We can update the value estimate at each time step.

For example, after observing $(A, 0, B)$, we have

$$V(A) \leftarrow V(A) + \alpha[R + \gamma V(B) - V(A)]$$

Note that in above, B is the next state S_{t+1}.

$$V(A) = 1 + 1[0 + 0.5 * 1 - 1]$$

So, $V(A) = 0.5$ and $V(B) = 1$ (remains the same).

Lesson 4: TD for control

11. Explain how GPI can be used with TD to find improved policies

Recall GPI with MC improve policy after each episode. Instead, GPI with TD will improve the policy after just one policy evaluation step.

12. Describe the Sarsa Control algorithm

see chap 6.4 (SARSA, uses state-action transition)

13. Understand how the Sarsa control algorithm operates in an example MDP

see the Windy Gridworld example

14. Analyze the performance of a learning algorithm

see the Windy Gridworld example

Lesson 5: Off-policy TD control: Q-learning

15. Describe the Q-learning algorithm

Q-learning, like SARSA, solves the Bellman equation using samples from environment. But instead of using the standard Bellman equation, Q-learning uess the Bellman's optimality equation for action values. The optimality equations enable Q-learning to directly learn Q_* instead of switching between policy improvement and policy evaluation steps.

16. Explain the relationship between Q-learning and the Bellman optimality equations.

SARSA is sample-based version of *policy iteration* which uses Bellman equations for action values, that each depend on a fixed policy.

Q-learning is a sample-based version of *value iteration* which iteratively applies the Bellman optimality equation.

17. Apply Q-learning to an MDP to find the optimal policy

see Windy gridword

18. Understand how Q-learning performs in an example MDP

see Windy gridword

19. Understand the differences between Q-learning and Sarsa

SARSA (on-policy)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Q-learning: (off-policy)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

20. Understand how Q-learning can be off-policy without using importance sampling

In SARSA, the agent bootstraps off the value of the value of the action it's going to take next, which is sampled from its behavior policy. Q-learning instead bootstraps off of the *largest* action value in its next state. The is like sampling an action under an estimate of the optimal policy rather than the behavior policy. Since Q-learning learns about the best action it could take rather the actions it actually takes, it is learning off-policy.

21. Describe how the on-policy nature of SARSA and the off-policy nature of Q-learning affect their relative performance

cliff walking example, Q-learning follows optimal policy, but its own ϵ-greedy action selection makes it fall off and result in lower reward.

Lesson 6: Expected Sarsa

22. Describe the Expected Sarsa algorithm

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)]$$

23. Describe Expected Sarsa's behaviour in an example MDP

24. Understand how Expected Sarsa compares to Sarsa control

25. Understand how Expected Sarsa can do off-policy learning without using importance sampling

26. Explain how Expected Sarsa generalizes Q-learning

27. Q-learning does not learn about the outcomes of exploratory actions because the update in Q-learning only learns about the greedy action

28. SARSA, Q-learning, and Expected SARSA have similar targets on a transition on a *terminal* state, because the target in this case only depends on reward.