Temporal-Difference Learning

wonseok jung

Introduction

- TD learning은 Monte Carlo와 Dynamic programming의 combination이다.
  - Directly learns from raw experience
  - Updating estimates based in part on other learned estimates. (bootstrap)

1. TD Prediction

- TD와 MC는 prediction problem을 풀기위해 직접 경험한 데이터를 사용한다.
- $policy \pi$를 따라 얻은 경험을 사용하여 estimate $V(S_t)$를 update한다.
- Monte Carlo는 $V(S_t)$를 update하기 위해 G_t (episode의 끝)까지 가서 value를 update한다.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

1.1 TD prediction

- TD methods need to wait only until the next time step

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + V(S_{t+1}) - V(S_t)]$$

- Target : $R_{t+1} + V(S_{t+1})$
- 위와같이 one step bootstrap한 것을 TD(0)라고 한다.

1.2 Monte Carlo와 TD의 Target 비교

- Monte Carlo의 Target : G_t

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

- Bellman Equation

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

- TD method의 Target : $R_{t+1} + \gamma v_\pi(S_{t+1})$

$$= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

1.3 TD error

- Monte Carlo와 TD 는 successor state 혹은 state-action pair를 사용하여 update한다.
- TD error : time t에서의 value와 successor state에서의 value의 차이
  $$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

2. Advantage of TD Predection Methods

- Do not require a model of the environment

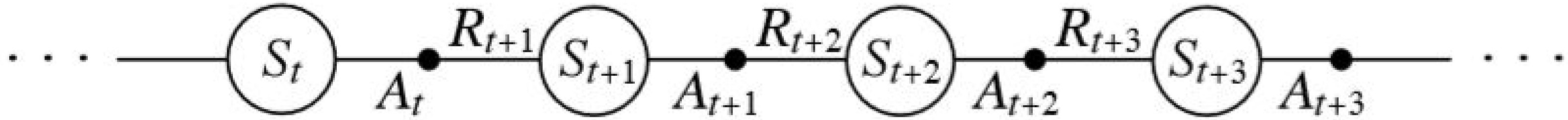
  -No need to know reward and transition probability
- Implemented in an online
  - Some applications have very long episode.
  - Monte Carlo would not suitable for the application.

3. TD Control

- Using of TD prediction for the control problem
- TD control is also faced need to trade off exploration and exploitation
- Two oapproaches
  - On Policy
  - Off policy

3.1 TD Control - on-policy

- Learn an action-value function
- On-policy method estimate $q_\pi(s, a)$ with current behavior policy π



3.1 TD Control - on-policy

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

- Update is done after every transition $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

- Target : $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

3.1 On-policy TD control :SARSA

- Sarsa Control Alorithm is given in the box

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

3.2 Off-policy TD control : Q-learning

- "One of the early breakthrough in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning" - Watkins,1989

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

- Target : $R_{t+1} + \gamma max_a Q(s_{t+1}, a)$

3.2 Off-policy TD control : Q-learning

- Q-learning also learns action-value function

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

3.2 Off-policy TD control : Q-learning

- Q-learning takes the maximum of "Next action" at "Next State"

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

3.3 Expected Sarsa

- Expected Sarsa is just like Q-learning (instead of the maximum over next state-action pairs using the expected value)
- How likely each action is under the current policy

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

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

3.4 Maximization Bias and Double Learning

- Q-learning : target policy is $greedy$
- Sarsa : target policy is often $\epsilon - greedy$
- Maximum over estimated values can lead to positive bias

3.4 Maximization Bias

- Example
  - Single state s with many actions a
  - Each true action-value $q(s,a)$ are all 0
  - Because of the uncertainty, some of estimate value $Q(s,a)$above and below 0
  - The maximum of the true value is 0, but the maximum of estimate value is positive.

3.4 Avoiding maximization bias?

- It is due to using the same samples both to determine the maximizing action and to estimate its value.
- Let's divide the plays two sets to learn two independent estimates
- $Q_1(a)$, $Q_2(a)$: each an estimate of true value $q(a)$
- Maximum value of two estimates
  - $A^* = argmax_a Q_1(a)$
  - $A^* = argmax_a Q_2(a)$

3.4 Double learning

- Using one estimate Q_1
- Determining maximizing action $A^* = argmax_a Q_1(a)$
- Provide another estimate of value Q_2 with $argmax_a Q_1(a)$
- $Q_2(A^*) = Q_2(argmax_a Q_1(a))$

3.4 Double learning

- This estimate will then be unbiased in the
  - $E[Q_2(A^*)] = q(A^*)$
- Role of the two estimates reversed
  - $Q_1(A^*) = Q_1(argmax_a Q_2(a))$
- This is the idea of Double learning

3.4 Double Q-learning

- Update rule of Double Q-learnign

$$Q_1(S_t, A_t) \leftarrow$$

$$Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, argmax_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)]$$

- The two approximate value functions are treated symmetrically

3.4 Complete algorithm for Double Q-learning

Double Q-learning

Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Summary : Temporal-Difference Learning

- We divided the problem into a prediction and control problem
- Prediction problem : TD methods are alternatives to Monte Carlo
- Control problem : generalized policy iteration(GPI)
- This is the idea that approximate policy and value functions should interact , both move toward their optimal values

Summary : Temporal-Difference Learning

- Prediction problem : predict return for the current policy
- Control problem : improving (ex:e-greedy) with respect to the current value function
- Two methods of TD control : on-policy, off-policy
  - On-policy : Sarsa
  - Off-policy : Q-learning , Expected Sarsa