Temporal-Difference Learning

Deboroto Das Robin

Kent State University

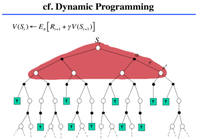
October 15, 2018

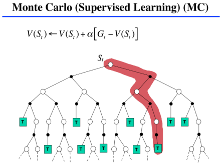
Future AI Methods Requirements

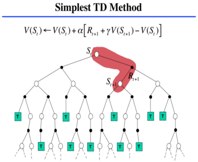
- Scalable : MC is scalable but DP not
- Model Free :
  - Not restricted to specific model : DL, CNN, all are model based
- Prediction Based
- Learns & predicts dynamically from environment : self driving car/drone

Techniques

- DP :
  - Neither model free nor scalable
  - Considers all possible steps
  - Bootstraps : pupdates existing estimates
- MC :
  - Doesn't ***bootstraps***
  - Estimates using sample of an episode
  - Visits all states & actions to a terminal state(full episode), finding out G_t
- TD :
  - Combines goodness of ***MC*** & ***DP***

- 

- 

- 

TD Prediction

Policy Evaluation (the prediction problem):
for a given policy π, compute the state-value function v_π

Recall: Simple every-visit Monte Carlo method:

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

Step-size parameter

target: the actual return after time t

The simplest temporal-difference method TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\left[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{target: an estimate of the return}} - V(S_t)\right]$$

TD(0)

- TD(0) or one step TD (update the value function after any individual step)
- Only wait until the next time step to update the value estimates
- MC wait for full episode

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

TD Learning $V(S_t) \leftarrow \underbrace{V(S_t)}_{\text{Previous estimate}} + \alpha[\underbrace{\underbrace{R_{t+1}}_{\text{Reward t+1}} + \underbrace{\gamma V(S_{t+1})}_{\text{Discounted value on the next step}}}_{\text{TD Target}} - \underline{V(S_t)}]$

-
- More generalized TD(λ) available in next chapters

TD(0) Algorithm

- Algorithm is

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

- Error at each step = R_{t+1} + γ $V(S_{t+1})$ - $V(S_t)$

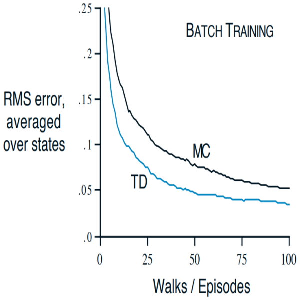
Advantages of TD Prediction Methods)

- Do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
  - Do not wait for full episode
- Less memory : Don’t look all steps like DP
- Less peak computation
- Converges like MC : given certain assumptions
-

Optimality of TD(0)

Assume we have finite amount of experience : 10 episodes or 100 time steps for a Random Walk scenario

- Batch Update :
  - time step t : time to move to next step
  - TD prediction equation: $V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
  - Increment = $\alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
  - At each t, increment is caculated ***But*** $V(S_t)$ is updated after 100 steps
- If α is small, under batch update $TD(0)$ converges to single answer
- MC also converges to single(but different) answer
-



Performance of TD(0)

- Optimal solution requires : for N states
  - N^2 memory & N^3 computation
  - Infesible for large problem
- TD reaches to convergence faster than MC
  - I am not clear about the logic !!!!
  - certainty-equivalence estimate – I am not clear about that
- For large state-space problem TD is only feasible solution

On-policy / Off-policy

- On-Policy :
  - attempt to evaluate or improve the ***policy(π)*** used to make decisions
  - Example : SARSA
- Of-Policy :
  - attempt to evaluate or improve the policy other than π
  - Example : Q-Learning

Sarsa: State-Action-Reward-State-Action

- Simple TD :
  - Moves from S_t to S_{t+1}
  - Learns predicting optimal state
  - Action is defined by ***Policy*** π
  - π is not changed
- On-Policy :
  - Changes π
  - How:
    - Instead of $S_t \longrightarrow S_{t+1}$ — Do $Q(S_t, A_t) \longrightarrow Q(S_{t+1}, A_{t+1})$
    - State action pair is learned $\longrightarrow \pi$ is changed
    - Actions are updated $A \longrightarrow A'$

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

  Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) =$

  Repeat (for each episode):

    Initialize S

    Choose A from S using policy derived from Q (e.g., ϵ-greedy)

    Repeat (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

Q-Learning:

- Of-Policy :
  - Does not change π
  - How:
    - Instead of $S_t \longrightarrow S_{t+1}$ — Do $Q(S_t, A_t) \longrightarrow Q(S_{t+1}, A_{t+1})$
    - State action pair is learned $\longrightarrow \pi$ is changed
    - Actions are ***NOT*** updated: NO $A \longrightarrow A'$

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

  Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state},$
  Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
      Choose A from S using policy derived from Q (e.g., ϵ-greedy)
      Take action A, observe R, S'
      $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
      $S \leftarrow S'$
    until S is terminal