Temporal Difference Learning

UBC MLRG Winter 2017 - Reinforcement Learning

Raunak Kumar

2017/01/31

University of British Columbia

So far

So far we know about

- Markov Decision Processes (MDPs)
- Estimating state value function v_π for a given policy π using DP and Monte Carlo methods
- Using variations of Generalized Policy Iteration to obtain an optimal policy

Today

Today we will talk about Temporal Difference (TD) Learning, which is a combination of DP and Monte Carlo ideas.

- Uses bootstrapping to update estimates based on other learned estimates (like DP)
- Learns directly from raw experience without a complete model of the environment (like Monte Carlo)

Note: The slides are based on (and figures taken from) the book by Sutton and Barto, Reinforcement Learning: An Introduction, Chapter 6.1 - 6.5

Monte Carlo Prediction

- Every-visit **Monte Carlo** method: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$
- G_t is the actual return following time t and α is a constant step-size parameter
- Must wait until the end of the episode to know G_t and hence, determine the increment to $V(S_t)$

TD Prediction

- **TD(0)** method: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- R_{t+1} is the observed reward, and $V(S_{t+1})$ is an estimate
- Only needs to wait until the next time step to update

TD Prediction

- $v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$
- **Monte Carlo** target is an estimate because a sample return is used instead of the real expected return
- **DP** target is an estimate because $v_\pi(S_{t+1})$ is not known, and $V(S_{t+1})$ is used
- **TD** combines both sampling and bootstrapping: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

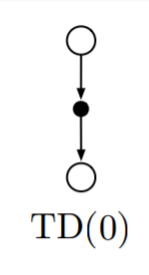
Tabular TD(0)

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
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

Backup Diagram

- **Sample** backups: Look ahead at a single successor state (like Monte Carlo, **TD**)
- **Full** backups: Look ahead at complete distribution of successors (like DP)



TD Error

- **TD Error**: $\delta_t = [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- This is the error made in the estimate *at that time*

TD Error

- **TD Error**: $\delta_t = [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- This is the error made in the estimate *at that time*
- Monte Carlo error can be written as the sum of TD Errors:

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\ &= \vdots \\ &= \sum_{k=0}^{T-t-1} \gamma^k \delta_{t+k} \end{aligned}$$

Monte Carlo error can be written as a sum of TD errors - why is this important?

Example

Since you are fascinated by machine learning, every day as you drive home from work, you predict how long your ride will be and note all the relevant details about time, weather etc.

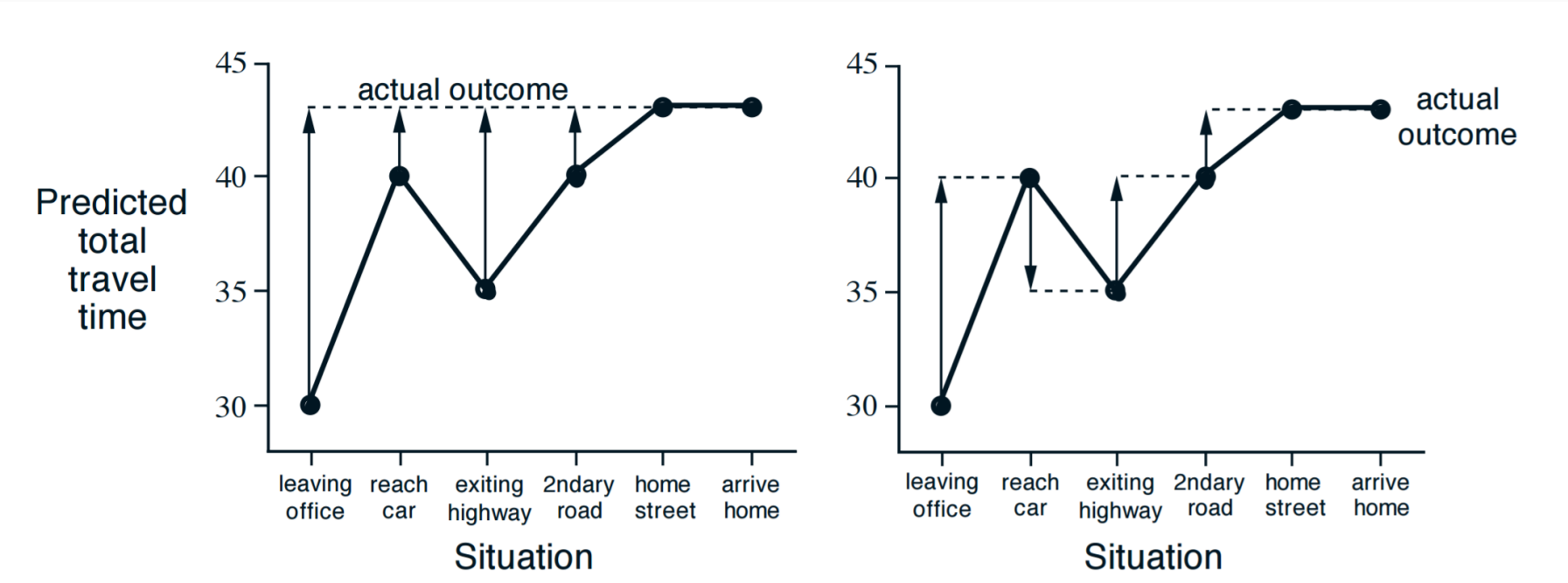
Example

Since you are fascinated by machine learning, every day as you drive home from work, you predict how long your ride will be and note all the relevant details about time, weather etc.

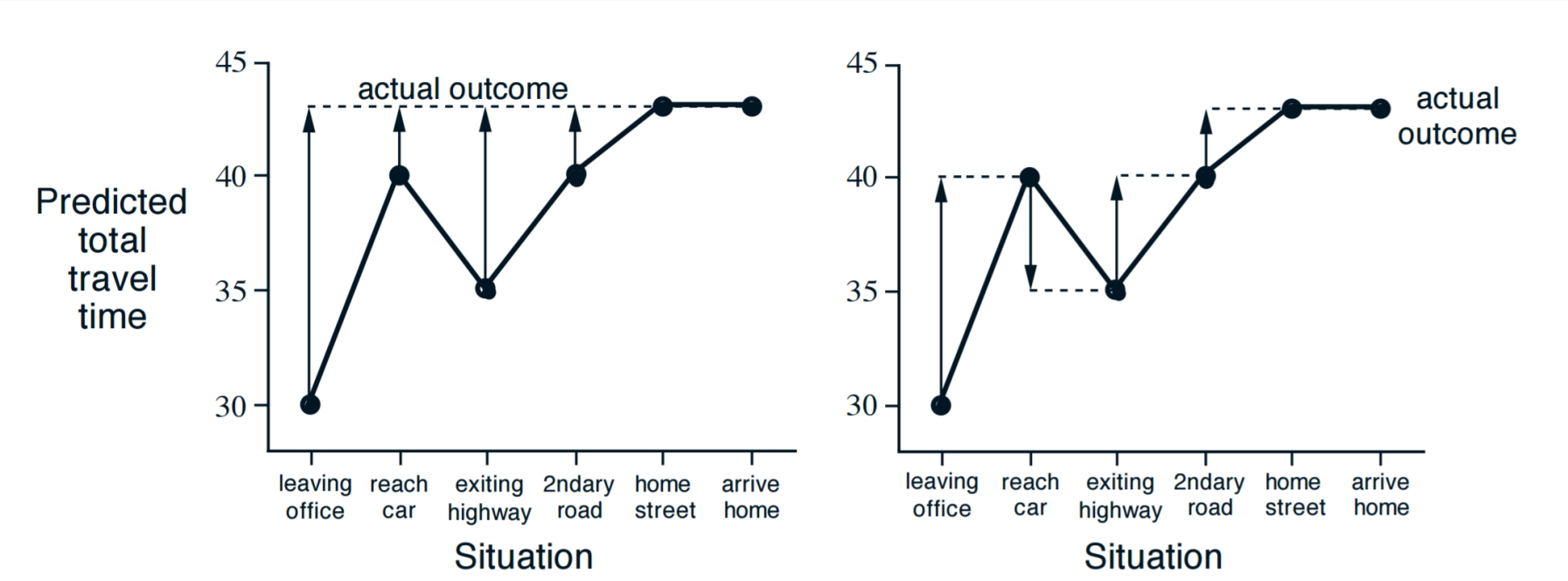
State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

- The rewards are the elapsed time on each leg of the journey.
- $\gamma = 1$, so the return for each state is the actual time to go from that state
- Value of each state is the expected time to go, and the second column gives an estimate of this

Example - MC vs TD



Example - MC vs TD



Consider another situation: You've moved to a different building, but you still enter the highway at the same location and around the same time. What would MC do? What would TD do?

Advantages of TD Prediction Methods

- “Developing and answering such questions will take the rest of this book and more”
- Advantage over DP: do not require a full model of the environment
- Advantage over Monte Carlo: on-line, fully incremental fashion
  - Can update every time step instead of at the end of episodes
  - If episodes are very long, this can be an important consideration
- Does TD work?
  - Proven to converge, discussed in the next few chapters
  - We don’t have a formal proof which method converges faster
  - In practice, TD has been found to work better

Batch Updating

- Consider a scenario where there's a small amount of experience available, for example 10 episodes
- Given an approximate value function, V, the increments are calculated at every time step
- But the value function is changed only once, by the sum of all the increments (**batch update**)
- Repeat with the experience and new value function till convergence

Why Batch Update

- Under batch updating, TD(0) converges deterministically to a single answer, independent of α (sufficiently small)
- constant-α MC also converges deterministically under similar conditions

Why Batch Update

- Under batch updating, TD(0) converges deterministically to a single answer, independent of α (sufficiently small)
- constant-α MC also converges deterministically under similar conditions but to a different answer

Example - Random Walk

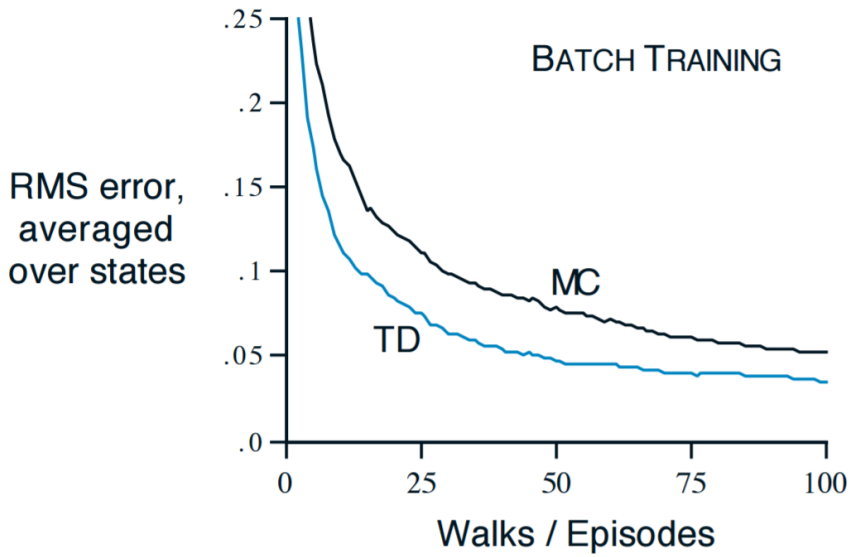


Figure 6.3: Performance of TD(0) and constant-α MC under batch training on the random walk task.

Another Example

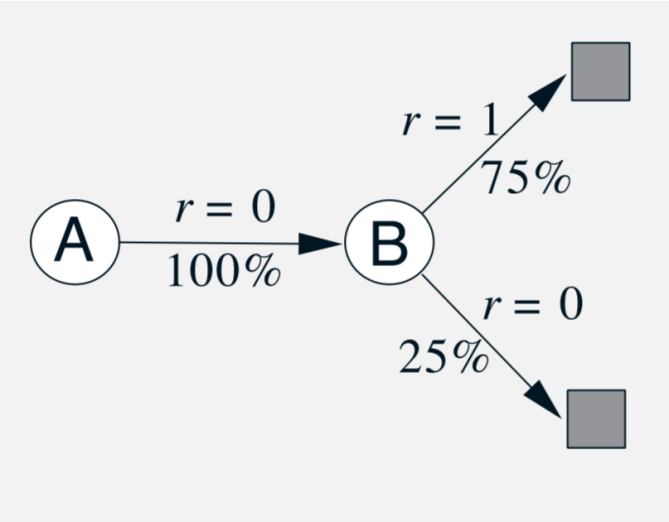
- There is an unknown Markov reward process with the following eight episodes:
  - A,0,B,0
  - B,1
  - B,1
  - B,1
  - B,1
  - B,1
  - B,1
  - B,0
- What are the best values for the estimates $V(A)$ and $V(B)$?

Another Example

- Optimal value for $V(B)$ is $\frac{3}{4}$

Another Example

- Optimal value for $V(B)$ is $\frac{3}{4}$
- What about $V(A)$?
  - Batch Monte Carlo method: 0. We saw A once with reward 0. It gives us minimum squared error on existing data.
  - Batch TD method: $\frac{3}{4}$. If the process is Markov, we expect this answer to be better on future data.



Optimality of TD(0)

- Batch MC method always finds estimates that minimizes **mean squared error** on the training set.

Optimality of TD(0)

- Batch MC method always finds estimates that minimizes **mean squared error** on the training set.
- Batch TD method always finds estimates that would be exactly correct for the **MLE model of Markov** process.

Optimality of TD(0)

- Batch MC method always finds estimates that minimizes **mean squared error** on the training set.
- Batch TD method always finds estimates that would be exactly correct for the **MLE model of Markov** process.
- This is called the **certainty-equivalence estimate**
- In general, Batch TD(0) converges to the certainty-equivalence estimate

SARSA: On-Policy TD Control

- First, we will learn an action-value function, $q_\pi(s, a)$ using TD(0)
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
- Rule uses a quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, aka "SARSA"
- Now, we design an on-policy algorithm
  - We continually estimate q_π for policy π
  - Change π toward greediness with respect to q_π

SARSA: On-Policy TD Control

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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

Continually estimate q_π and at the same time change π toward greediness with respect to q_π

SARSA: On-Policy TD Control

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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

Convergence properties depends on the policy's dependence on Q

Always converges to an optimal policy and q_π as long as all state-action pairs are visited an infinite number of times and the policy converges to the greedy policy

Q-Learning: Off-Policy TD Control

- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
- The learned Q function directly approximates the optimal action-value function, independent of π
- π still affects which state-action pairs are visited and updated
- Only requirement for convergence: all pairs continue to be updated

Q-Learning: Off-Policy TD Control

Q-learning: An off-policy TD control algorithm

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