

# Lecture 4: Model-Free Prediction

David Silver

# Outline

- 1 Introduction
- 2 Monte-Carlo Learning
- 3 Temporal-Difference Learning
- 4  $TD(\lambda)$

# Model-Free Reinforcement Learning

- Last lecture:
  - Planning by dynamic programming
  - Solve a *known* MDP
- This lecture:
  - Model-free prediction
  - Estimate the value function of an *unknown* MDP
- Next lecture:
  - Model-free control
  - Optimise the value function of an *unknown* MDP

# Monte-Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to *episodic* MDPs
  - All episodes must terminate

## Monte-Carlo Policy Evaluation

- Goal: learn  $v_\pi$  from episodes of experience under policy  $\pi$

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

- Monte-Carlo policy evaluation uses *empirical mean return* instead of *expected return*

# First-Visit Monte-Carlo Policy Evaluation


- To evaluate state  $s$
- The **first** time-step  $t$  that state  $s$  is visited in an episode,
- Increment counter  $N(s) \leftarrow N(s) + 1$
- Increment total return  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- By law of large numbers,  $V(s) \rightarrow v_\pi(s)$  as  $N(s) \rightarrow \infty$



# Every-Visit Monte-Carlo Policy Evaluation

- To evaluate state  $s$
- **Every** time-step  $t$  that state  $s$  is visited in an episode,
- Increment counter  $N(s) \leftarrow N(s) + 1$
- Increment total return  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- Again,  $V(s) \rightarrow v_{\pi}(s)$  as  $N(s) \rightarrow \infty$

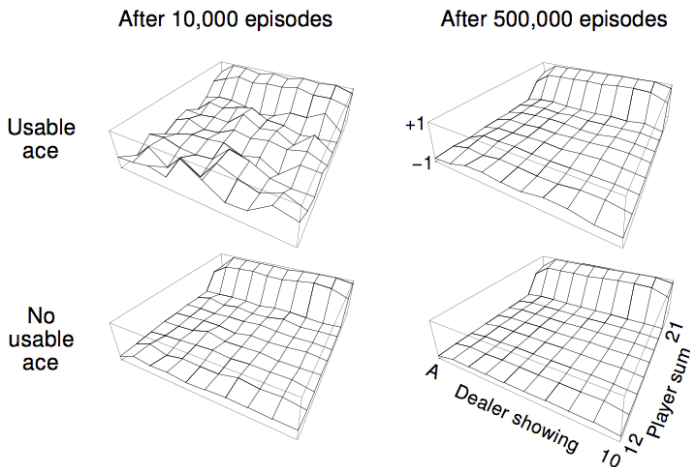
# Blackjack Example

- States (200 of them): 
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- Action **stick**: Stop receiving cards (and terminate)
- Action **twist**: Take another card (no replacement)
- Reward for **stick**:
  - +1 if sum of cards  $>$  sum of dealer cards
  - 0 if sum of cards = sum of dealer cards
  - -1 if sum of cards  $<$  sum of dealer cards
- Reward for **twist**:
  - -1 if sum of cards  $>$  21 (and terminate)
  - 0 otherwise
- Transitions: automatically **twist** if sum of cards  $<$  12





# Blackjack Value Function after Monte-Carlo Learning



Policy: **stick** if sum of cards  $\geq 20$ , otherwise **twist**

# Incremental Mean

The mean  $\mu_1, \mu_2, \dots$  of a sequence  $x_1, x_2, \dots$  can be computed incrementally,

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$



# Incremental Monte-Carlo Updates

- Update  $V(s)$  incrementally after episode  $S_1, A_1, R_2, \dots, S_T$
- For each state  $S_t$  with return  $G_t$

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

- In **non-stationary** problems, it can be useful to **track a running mean**, i.e. forget old episodes.

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

# Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

# MC and TD

- Goal: learn  $v_\pi$  online from experience under policy  $\pi$
- Incremental every-visit Monte-Carlo
  - Update value  $V(S_t)$  toward *actual* return  $G_t$

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

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value  $V(S_t)$  toward *estimated* return  $R_{t+1} + \gamma V(S_{t+1})$

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

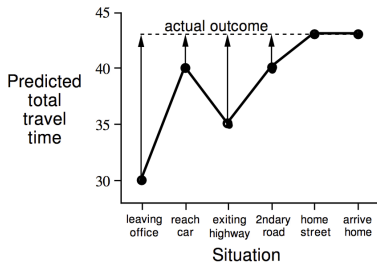
- $R_{t+1} + \gamma V(S_{t+1})$  is called the **TD target**
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  is called the **TD error**

## Driving Home Example

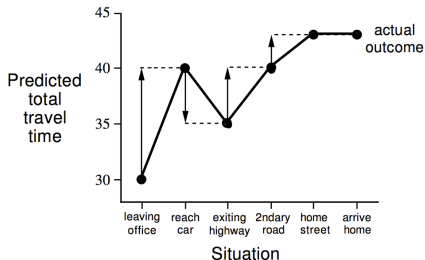
<b>State</b>	<b>Elapsed Time (minutes)</b>	<b>Predicted Time to Go</b>	<b>Predicted Total Time</b>
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

# Driving Home Example: MC vs. TD

Changes recommended by  
Monte Carlo methods ( $\alpha=1$ )



Changes recommended  
by TD methods ( $\alpha=1$ )



# Advantages and Disadvantages of MC vs. TD

- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments



# Bias/Variance Trade-Off

- **Return**  $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$  is *unbiased* estimate of  $v_\pi(S_t)$
- **True TD target**  $R_{t+1} + \gamma v_\pi(S_{t+1})$  is *unbiased* estimate of  $v_\pi(S_t)$
- TD target  $R_{t+1} + \gamma V(S_{t+1})$  is *biased* estimate of  $v_\pi(S_t)$
- TD target is **much lower variance** than the return:
  - Return depends on **many** random actions, transitions, rewards
  - TD target depends on **one random action**, transition, reward

# Advantages and Disadvantages of MC vs. TD (2)

- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to  $v_{\pi}(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

