COMS W4701: Artificial Intelligence

Lecture 11: RL Prediction

Tony Dear, Ph.D.
Department of Computer Science
School of Engineering and Applied Sciences

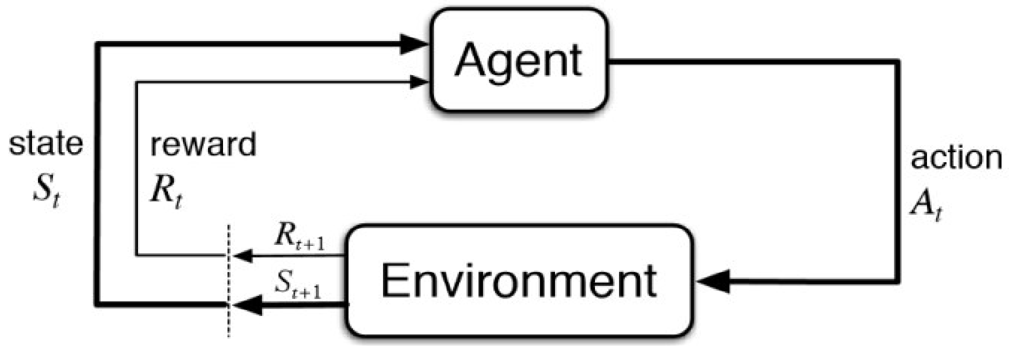
Today

- Reinforcement learning
- Monte Carlo (MC) prediction
- Temporal difference (TD) prediction
- Comparing DP, MC, and TD

Learning from Experience

- Dynamic programming requires knowledge of environment *model* (transition and reward functions), but often inaccessible or intractable

- **Reinforcement learning**: Find optimal policies through experience
- Interact with environment, receive rewards, and formulate policies



Dimensions of RL

- *Model-based* methods learn an approximation of the underlying model
- *Model-free* methods directly learn policies or value functions
- Can be useful even when model is known but DP is intractable

- *Prediction*: Given a fixed policy π, learn V^π
- *Control*: Learn an optimal policy π^*, *state-action* value function Q^*

- Model-free methods for both prediction and control include *Monte Carlo* and *temporal difference* algorithms

Monte Carlo Methods

- **Monte Carlo** methods: Generate sampled experience and average them for different states and actions

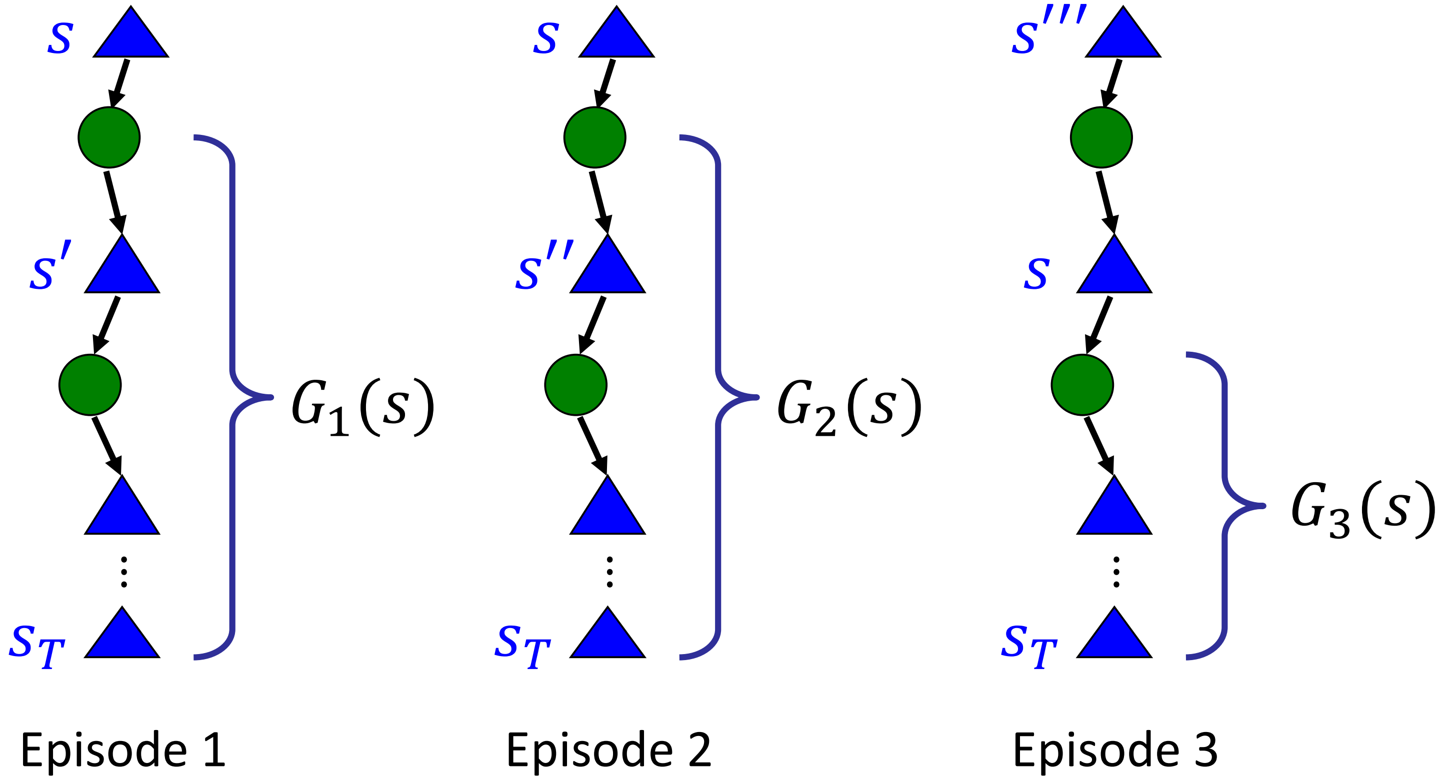
- Recall the definition of value function for a fixed policy π:

$$V^{\pi}(s) = E\left[\sum_{t=0} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right]$$

- Idea: Approximate the expectation by taking *averages* of sample reward sequences over multiple *episodes*

State Value Estimation

- Idea: $V(s)$ can be estimated by averaging utilities observed *after* visiting s
- Think of each sample as a path from a given node to a leaf node in search tree



$\cdots$

$$V(s) = \frac{1}{N}\sum_{i=1}^{N} G_i(s)$$

First-Visit MC Prediction

- **MC prediction**: Estimate state values by averaging utilities over multiple episodes
- *First-visit* MC: A value is estimated after first visit to state within episode

- **Initialize** $V^{\pi}(s) \leftarrow 0$, $N(s) \leftarrow 0$ for each state $s \in S$
- **Loop**:
  - **Generate** episode E following π**:** $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
  - **For** each state s:
    - $G \leftarrow \sum_{j=t+1}^{T} \gamma^{j-(t+1)} r_j$, where s_t is first occurrence of s in E
    - $V^{\pi}(s) \leftarrow \frac{1}{N(s)+1}\left(N(s) \times V^{\pi}(s) + G\right)$
    - $N(s) \leftarrow N(s) + 1$

Example: Mini-Gridworld

- States: A, B, C; actions: L, R; rewards received upon entering each state
- Policy: $\pi(s) = L$ for all states s
- Suppose we use episodes with 5 transitions

A	B	C

- Episode 1: $(A, +3, A, -2, B, +1, C, -2, B, +3)$

$\gamma = 0.5$

$$V^{\pi}(A) \leftarrow G(A) = 3 + \gamma(-2) + \gamma^2(1) + \gamma^3(-2) + \gamma^4(3) = 2.1875$$

$$V^{\pi}(B) \leftarrow G(B) = 1 + \gamma(-2) + \gamma^2(3) = 0.75$$

$$V^{\pi}(C) \leftarrow G(C) = -2 + \gamma(3) = -0.5$$

Example: Mini-Gridworld

- States: A, B, C; actions: L, R; rewards received upon entering each state
- Policy: $\pi(s) = L$ for all states s
- Suppose we use episodes with 5 transitions

A	B	C

- Episode 2: $(A, -2, B, +3, A, -2, B, +1, C, -2)$

$\gamma = 0.5$

$$V^\pi(A) \leftarrow \frac{1}{2}\big(V^\pi(A) + G(A)\big) = \frac{1}{2}(2.1875 - 1) = 0.59375$$

$$V^\pi(B) \leftarrow \frac{1}{2}\big(V^\pi(B) + G(B)\big) = \frac{1}{2}(0.75 + 2) = 1.375$$

$$V^\pi(C) \leftarrow \frac{1}{2}\big(V^\pi(C) + G(C)\big) = \frac{1}{2}(-0.5 - 2) = -1.25$$

Example: Mini-Gridworld

- States: A, B, C; actions: L, R; rewards received upon entering each state
- Policy: $\pi(s) = L$ for all states s
- Suppose we use episodes with 5 transitions

A	B	C

- Episode 3: $(C, +1, C, -2, B, +3, A, -2, B, +3)$

$\gamma = 0.5$

$$V^\pi(A) \leftarrow \frac{1}{3}\big(2V^\pi(A) + G(A)\big) = \frac{1}{3}(2(0.59375) - 0.5) = 0.229$$

$$V^\pi(B) \leftarrow \frac{1}{3}\big(2V^\pi(B) + G(B)\big) = \frac{1}{3}(2(1.375) + 2.75) = 1.833$$

$$V^\pi(C) \leftarrow \frac{1}{3}\big(2V^\pi(C) + G(C)\big) = \frac{1}{3}(2(-1.25) + 0.6875) = -0.604$$

Finer Points

- Different states will have different visited frequencies, but all states will be visited infinitely often in the limit—values will converge to true V^{π}

- Estimates of different state values are *independent* (in contrast to DP)
- Accuracy of $V^{\pi}(s)$ does *not* depend on accuracy of $V^{\pi}(s')$

- Result: Computational complexity of estimating specific state values is independent of state space size!
- Can choose to focus on certain states and ignore others

Constant-α Monte Carlo

- Another way of writing the state value updates:

$$V^{\pi}(s_t) \leftarrow \frac{NV^{\pi}(s_t) + G_t}{N+1} = V^{\pi}(s_t) + \frac{1}{N+1}\left(G_t - V^{\pi}(s_t)\right)$$

- Update: "new estimate" = "old estimate" + "step size" × ("target" – "old estimate")

- In some problems, we may want to give a higher weight to *recent* returns
- **Constant-α MC** *exponentially* decays the weights on past returns by factor $1 - \alpha$

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \boldsymbol{\alpha}\left(G_t - V^{\pi}(s_t)\right) = \alpha \sum_{i=1}^{t} (1-\alpha)^{t-i} G_i$$

Recency-Weighted Average

- For constant α, we end up applying an *exponentially smaller* weight to each return value further back in the past:

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\big(G_t - V^\pi(s_t)\big) = \alpha G_t + (1-\alpha)V^\pi(s_t)$$

$$= \alpha G_t + (1-\alpha)(\alpha G_{t-1} + (1-\alpha)V^\pi(s_{t-1})\,)$$

$$= \alpha G_t + (1-\alpha)\alpha G_{t-1} + (1-\alpha)^2 V^\pi(s_{t-1})$$

$$= \cdots = \alpha \sum_{i=1}^{t} (1-\alpha)^{t-i} G_i$$

Temporal-Difference Learning

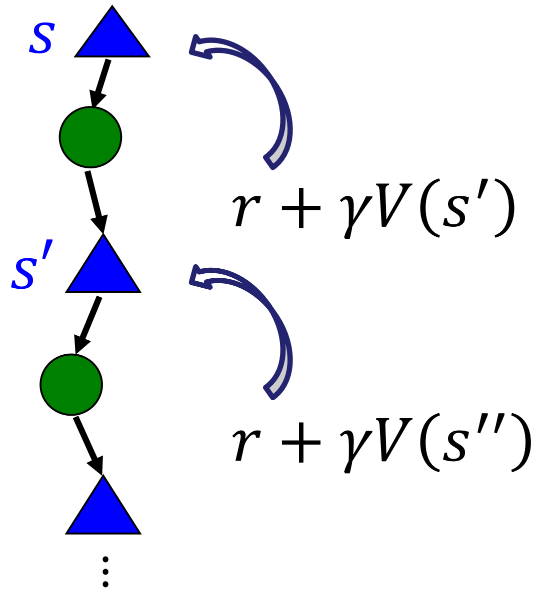
- MC requires episodic structure—what about infinite horizon problems?
- State values in MC are estimated entirely independently of each other
- Maybe we can borrow the idea of the *Bellman update* from dynamic programming

- **One-step TD ($\boldsymbol{TD(0)}$)**: We can replace the *target* term with the sum of immediate reward with discounted successor state value
- We can update $V^{\pi}(s)$ *immediately* before the episode ends

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\big(\overbrace{\underbrace{r_{t+1} + \gamma V^{\pi}(s')}_{\text{Target}} - V^{\pi}(s)}^{\text{TD error } \delta_t}\big)$$

$TD(0)$ for Prediction

- **Given:** Policy π, *learning rate* α between 0 and 1
- **Initialize** $V^{\pi}(s) \leftarrow 0$
- **Loop**:
  - **Initialize** starting state s if needed
  - **Generate** sequence $(s, \pi(s), r, s')$
  - $V^{\pi}(s) \leftarrow V^{\boldsymbol{\pi}}(s) + \alpha\big(\boldsymbol{r} + \boldsymbol{\gamma V^{\pi}(s')} - V^{\pi}(s)\big)$
  - $s \leftarrow s'$



Example: Mini-Gridworld

A	B	C

- All values initialized to 0; $\gamma = 0.8$, $\alpha = 0.5$
- Policy to evaluate: $\pi(s) = L$ for all states

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\left(r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\right)$$

- Observed state and reward sequence: $(A, +3, A)$

$$\begin{pmatrix} V^\pi(A) \\ V^\pi(B) \\ V^\pi(C) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$V^\pi(A) \leftarrow V^\pi(A) + \alpha\left(r + \gamma V^\pi(A) - V^\pi(A)\right)$$

$$V^\pi(A) \leftarrow 0 + 0.5(3 + 0.8(0) - 0) = 1.5$$

Example: Mini-Gridworld

- All values initialized to 0; $\gamma = 0.8$, $\alpha = 0.5$
- Policy to evaluate: $\pi(s) = L$ for all states

A	B	C

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\big(r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\big)$$

- Observed state and reward sequence: $(A, -2, B)$

$$\begin{pmatrix} V^\pi(A) \\ V^\pi(B) \\ V^\pi(C) \end{pmatrix} = \begin{pmatrix} 1.5 \\ 0 \\ 0 \end{pmatrix}$$

$$V^\pi(A) \leftarrow V^\pi(A) + \alpha\big(r + \gamma V^\pi(B) - V^\pi(A)\big)$$
$$V^\pi(A) \leftarrow 1.5 + 0.5(-2 + 0.8(0) - 1.5) = -0.25$$

Example: Mini-Gridworld

A	B	C

- All values initialized to 0; $\gamma = 0.8$, $\alpha = 0.5$
- Policy to evaluate: $\pi(s) = L$ for all states

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\big(r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\big)$$

- Observed state and reward sequence: $(B, +1, C)$

$$\begin{pmatrix} V^\pi(A) \\ V^\pi(B) \\ V^\pi(C) \end{pmatrix} = \begin{pmatrix} -0.25 \\ 0 \\ 0 \end{pmatrix}$$

$$V^\pi(B) \leftarrow V^\pi(B) + \alpha\big(r + \gamma V^\pi(C) - V^\pi(B)\big)$$

$$V^\pi(B) \leftarrow 0 + 0.5(1 + 0.8(0) - 0) = 0.5$$

Example: Mini-Gridworld

- All values initialized to 0; $\gamma = 0.8$, $\alpha = 0.5$
- Policy to evaluate: $\pi(s) = L$ for all states

A	B	C

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha\left(r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)\right)$$

- Observed state and reward sequence: $(C, -2, B)$

$$\begin{pmatrix} V^{\pi}(A) \\ V^{\pi}(B) \\ V^{\pi}(C) \end{pmatrix} = \begin{pmatrix} -0.25 \\ 0.5 \\ 0 \end{pmatrix}$$

$$V^{\pi}(C) \leftarrow V^{\pi}(C) + \alpha\left(r + \gamma V^{\pi}(B) - V^{\pi}(C)\right)$$

$$V^{\pi}(C) \leftarrow 0 + 0.5(-2 + 0.8(0.5) - 0) = -0.8$$

Finer Points

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha\delta_t$$

- TD methods perform updates immediately with no episodic structure (MC)
- Useful if problems have long episodes or are continuing tasks

- If α is constant, V^{π} will continue jumping around with each new target
- May be desirable if problem is nonstationary

- We can also shrink α over time if we want V^{π} to converge

Optimality of $TD(0)$

- Suppose we use a *sequence* of α_n step size values over time
- Stochastic approximation theory assures us that $TD(0)$ if α_n meets the following conditions:

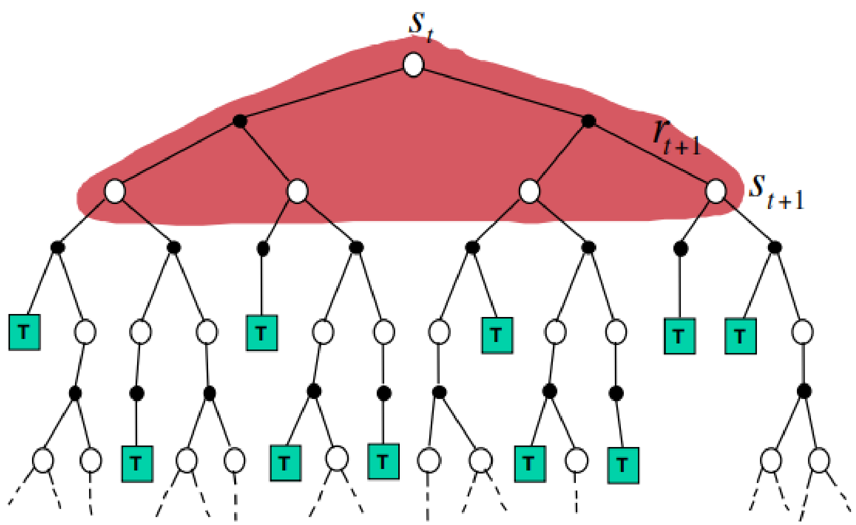
$$\sum_{n=1}^{\infty} \alpha_n = \infty \qquad \sum_{n=1}^{\infty} \alpha_n^2 < \infty$$

- First condition ensures that initial steps are large enough to overcome initial conditions or random fluctuations
- Second condition ensures that updates do eventually shrink to 0

- These conditions are satisfied by the sample averaging method $\alpha_n = \frac{1}{n}$

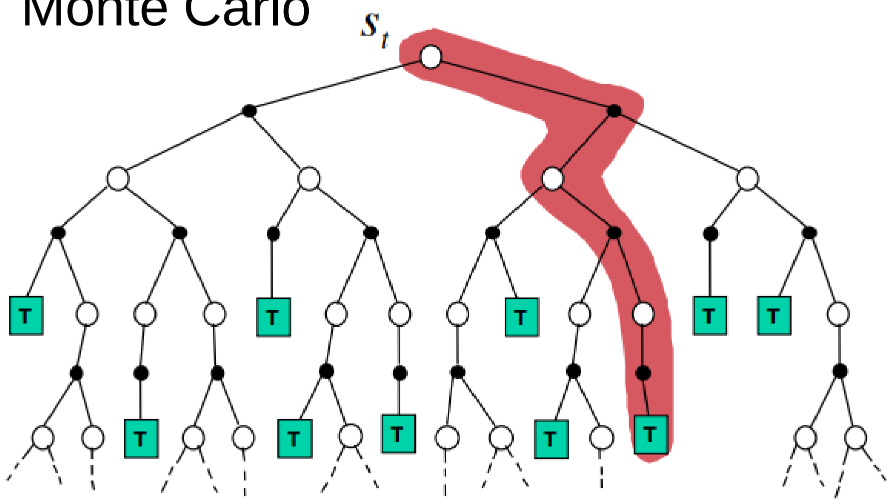
MDP Method Comparison

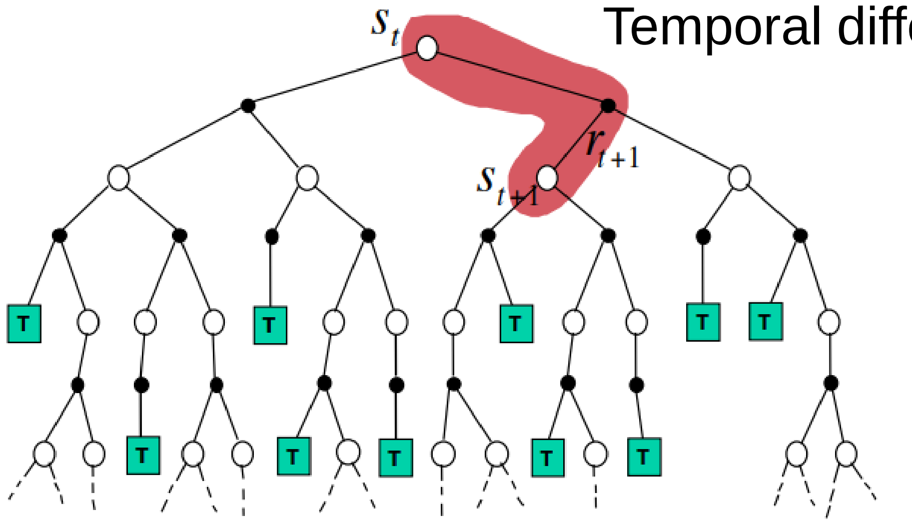
https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf



Dynamic programming

Monte Carlo





Temporal difference

Summary

- Reinforcement learning is used to estimate values/policies from data
- Instead of using underlying models, agent observes state-action-rewards

- Prediction problem: Evaluate a given policy

- Monte Carlo methods estimate by averaging samples of episodic returns

- Temporal difference methods bootstrap by using estimates to inform other estimates