- **Model-Free Prediction** - See slide 45
  - Estimate value function using:
    1. Monte Carlo
    2. Temporal Difference
    3. TD(λ)
  - This lecture just looks at predicting reward for a policy w/o models
  - Next lecture looks at control
  - **Two Dichotomies:**
    - Planning vs. Model Free / Prediction vs. Control

- **Monte Carlo Reinforcement Learning:**
  - Monte Carlo methods learn directly from episodes of experience
    - ↪ Requires that experiences terminate
  - **Goal:** Learn v_π from episodes of experience under policy π

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

- ↪ In Monte Carlo, we use empirical mean return
  1. **First-visit MC Evaluation:** In each episode, obtain the value function from each state using trajectories obtained after that state is reached for the **first time**. Average value function from each state using many episodes.

$N(s)$: # of first visits to s in the episodes

↪ $N(s) \leftarrow N(s) + 1$ upon first visit

$S(s)$: Total return from visits to s

↪ $S(s) \leftarrow S(s) + G_t$ from first visit

$\therefore V(s) = S(s)/N(s)$

- By Law of Large numbers, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$

② Every visit MC Evaluation: Just like FVMC, but increment counter $N(s)$ and Score $S(s)$ for Every visit to each state.

- Blackjack as an MDP
  - ↳ State space has 3 different variables:
    1. Current sum (only consider states with decisions ⇒ Sum between 12 - 21).
    2. Dealer's showing card
    3. Do I have "usable" ace (yes, no) ← can take values 1 or 11
  - ↳ Actions: Hit, miss
  - ↳ Reward: +1 for win; 0 for tie; -1 for miss
  - ↳ Policy evaluated using every visit MC by simulating Same # of episodes
    - ↳ Noise is observed around rare states, since you don't encounter rare states in MC as frequency
  - ↳ Sample returns and model VF without a model.

- Incremental Means: Means can be computed incrementally

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

  - ↳ New mean = old mean + Step * error
  - ↳ Correct prediction in direction of error

- Incremental MC Updates:

For each state S_t with return G_t:

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

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

↑ Always forward-looking, using reward from state obtained at the end of the episode.

For non-stationary problems, track a running mean and forget old episodes

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

↑ Constant Learning Rate

↳ Don't store N

↳ Memoryless algorithms

- **Temporal Difference Learning (TD)**

↳ Learn from actual atomic experiences (even incomplete episodes)

↳ Model-Free Approach

↳ Bootstrap: Update guess of value function as you get more data.

Goal: Learn V^π online from experiences under π

↳ TD(0) = Simplest TD

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{\left(\underbrace{R_{t+1} + \gamma V(s_{t+1})}_{\text{Estimated return from one step.}} - V(s_t) \right)}_{\text{TD Error}}$$

↑ "TD Target": Random target depending on next time step

↳ Estimated, biased algorithm

↳ TD learning resembles how we learn in real life

- TD can learn before you see the final outcome, while MC requires complete episodes
- TD can learn for non-terminal processes
  - ↳ TD will eventually find the correct value function

- **Bias / Variance Tradeoff:** Return G_t is an unbiased estimate of $V_\pi(s_t)$
  - ↳ TD Target contains $V(s_{t+1})$ which is not necessarily the true value function ∴ Introduces bias
  - ↳ G_t incurs noise from many steps ⇒ High variance
  - ↳ TD looks at noise from a single step ⇒ Low variance

  - ↳ MC has very good convergence even w/ function approx
    - ↳ MC is not very sensitive to initialization
    - ↳ MC is 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 w/ function approx.
    - ↳ More sensitive to initial value

- **Random Walk TD Example:**
  - ↳ 1D Random walk with uniform left/right probability; reward 1 at RHS and 0 elsewhere
  - ↳ TD learning converges faster than MC.
    - ↳ By bootstrapping, you learn more efficiently than when incurring full variance of episode using MC

- **Batch MC and TD:**
  - VF is average of G_t observed from each state in the batch ← MC estimate
  - Alternatively, look at empirical transition probabilities and returns associated with each state in the trajectories ← TD Estimate

- MC converges to the solution with the minimum Mean Squared Error: $\sum_{k=1}^{K} \sum_{t=1}^{T} \left(g_t^k - V(s_t^k) \right)^2$
  - k: Episodes

- TD(0) converges to solution of maximum likelihood Markov Model
  - First get empirical transition probabilities
  - Calculate mean reward from each state

- Markov Decision Process has Markov Property, which is exploited by TD, which makes use of this property to be more efficient

- MC ignores Markov Property, which can be beneficial if process exhibits non-Markov behavior

- Monte Carlo samples trajectories from States and uses those samples to estimate VF
  - TD learning looks just 1 step ahead
  - DP look 1 step look ahead, but did not sample, instead did full look-ahead

- **Bootstrapping vs. Sampling:**
  - Bootstrapping uses estimates instead of real returns for updates
    - MC = No bootstrapping
    - TD = Bootstrap
    - DP = Bootstrap

↳ Sampling:

↳ MC, TD sample

↳ DP does not sample, instead using full-width look-ahead

• **TD(λ)** lets you specify the depth of your backups

↳ As you take steps, increase depth, you ground yourself in real dynamics and rewards. Real dynamics always brings you closer to ground truth.

↳ What if you use multiple steps in TD learning.

↳ TD can be generalized to n-step experiences

$G_t^{(1)} \Rightarrow n=1$, 1 step return $\therefore \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$

$G_t^{(2)} \Rightarrow n=2$, 2 step return $\therefore \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$

$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$

$G_t^{(\infty)} = MC$

$\therefore$ n-step TD Update:

$$V(S_t) \longleftarrow V(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$$

↳ As $n \to \infty$, you get very high errors for short training times, like MC

↳ There is a sweet spot around $n = 3$ or 5 which minimizes error.

* We want an algorithm that efficiently considers all n at once

↳ We can <u>average over n-step returns</u> and call that the target

↳ How can we efficiently combine all n?

↳ λ return!

↳ G_t^{λ} combines n-step returns $G_t^{(n)}$ using weights $(1-\lambda)\lambda^{n-1}$

$\therefore \; G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$ ⟵ Geometric weighting

$\therefore$ Forward-viewing λ:

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{\lambda} - V(S_t) \right)$$

* Weighting decays with the number of steps you look into the future

↳ With geometric weightings, you can run TD(λ) with same time as TD(0).

↳ Forward-view TD(λ) suffers the same consequences as MC

↳ $\lambda = 1 \Rightarrow$ MC
$\lambda = 0 \Rightarrow$ TD(0)

• Backward View TD(λ):

↳ Eligibility Trace: Assigning credit for recency and frequency

↳ Combines Frequency and recency heuristics

↳ Backward view TD(λ):

① Keep an eligibility trace for each state s

② Update value $V(s)$ for each state s in proportion to TD Error δ_t and eligibility trace $E_t(s)$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

↳ VF updated on previous states

↳ When $\lambda = 0$, only current state is updated

↳ When $\lambda = 1$, there is no recency decay ⇔ MC

* The sum of offline updates is the same for forward-view and backward-view TD(λ)

↑ Theorem.

TD(λ) gives you spectrum between TD(0) and MC