CSC2541: Deep Reinforcement Learning

Lecture 3: Monte-Carlo and Temporal Difference

Slides borrowed from David Silver, Andrew Barto

Jimmy Ba

UNIVERSITY OF TORONTO

Algorithms

Multi-armed bandits		UCB-1, Thompson Sampling
Finite MDPs with model		dynamic programming
Linear model		LQR
Large/infinite MDPs		Theoretically intractable
		Need approx. algorithm

Outline

- MDPs without full model or with unknown model
    a. Monte-Carlo methods
    b. Temporal-Difference learning
- Seminar paper presentation

Monte-Carlo methods

- Problem: we would like to estimate the value function of an unknown MDP under a given policy.
- The state-value function can be decomposed into immediate reward plus discounted value of successor state.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Monte-Carlo methods

- Problem: we would like to estimate the value function of an unknown MDP under a given policy.
- The state-value function can be decomposed into immediate reward plus discounted value of successor state.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

- We can lump the stochastic policy and transition function under expectation.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

Monte-Carlo methods

- Idea: use Monte-Carlo samples to estimate expected discounted future returns

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

- Average returns observed after visits to s
    a. The first time-step t that state s is visited in an episode,
    b. Increment counter N(s) ← N(s) + 1
    c. Increment total return S(s) ← S(s) + Gt
    d. Value is estimated by mean return V(s) = S(s)/N(s)
- Monte-Carlo policy evaluation uses empirical mean return vs expected return

First-visit Monte Carlo policy evaluation

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

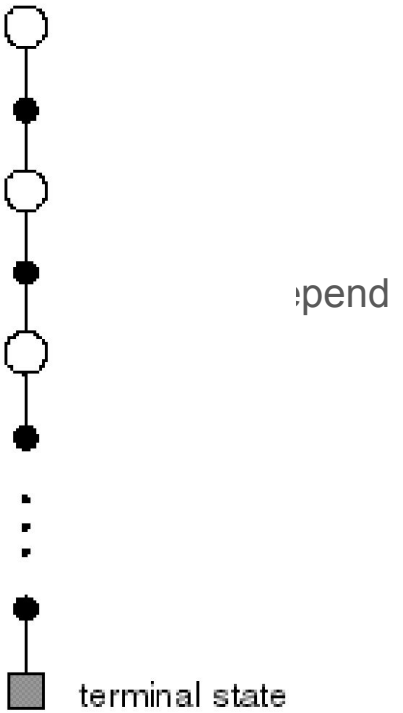
$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

Backup diagram for Monte-Carlo

- Entire episode included
- Only one choice at each state (unlike DP)
- MC does not bootstrap
- Time required to estimate one state does not depend on the total number of states



Off-policy MC method

- Use importance sampling for the difference in behaviour policy π' vs control policy π

Suppose we have n_s returns, $R_i(s)$, from state s, each with probability $p_i(s)$ of being generated by π and probability $p'_i(s)$ of being generated by π'. Then we can estimate

$$V^\pi(s) \approx \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)}}$$

which depends on the environmental probabilities $p_i(s)$ and $p'_i(s)$. However,

$$p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}^{a_k}_{s_k s_{k+1}}$$

and

$$\frac{p_i(s_t)}{p'_i(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}^{a_k}_{s_k s_{k+1}}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) \mathcal{P}^{a_k}_{s_k s_{k+1}}} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}.$$

Thus the weight needed, $p_i(s)/p'_i(s)$, depends only on the two policies and not at all on the environmental dynamics.

Monte-Carlo vs Dynamic Programming

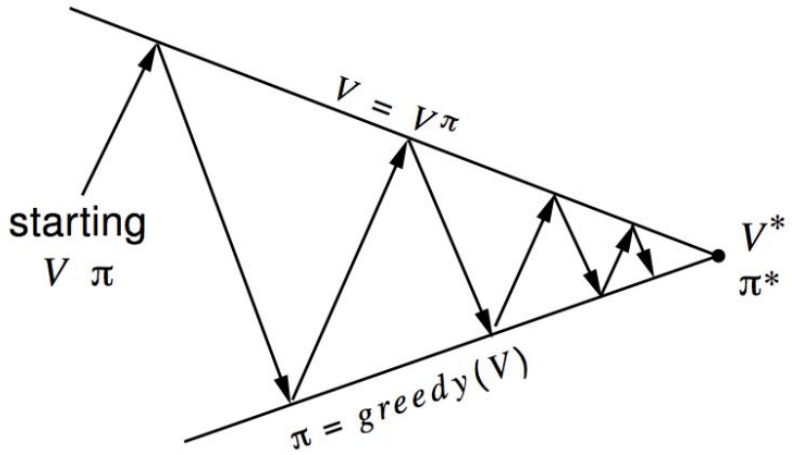
- Monte Carlo methods learn from complete sample returns
- Only defined for episodic tasks
- Monte Carlo methods learn directly from experience
    a. On-line: No model necessary and still attains optimality
    b. Simulated: No need for a full model
- MC uses the simplest possible idea: value = mean return
- Monte Carlo is most useful when
    a. a model is not available
    b. enormous state space

Monte-Carlo control

- How to use MC to improve the control policy?

Monte-Carlo control

- How to use MC to improve the current control policy?
- MC estimate the value function of a given policy
- Run a variant of the policy iteration algorithms to improve the current behaviour



Policy improvement

- Greedy policy improvement over V requires model of MDP

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over Q(s, a) is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s, a)$$

Monte-Carlo methods

- MC methods provide an alternate policy evaluation process
- One issue to watch for:
    a. maintaining sufficient exploration ! exploring starts, soft policies
- No bootstrapping (as opposed to DP)

Temporal-Difference Learning

- Problem: learn Vπ online from experience under policy π
- Incremental every-visit Monte-Carlo:
  a. Update value V toward **actual** return G

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

  b. But, only update the value **after** an entire episode

Temporal-Difference Learning

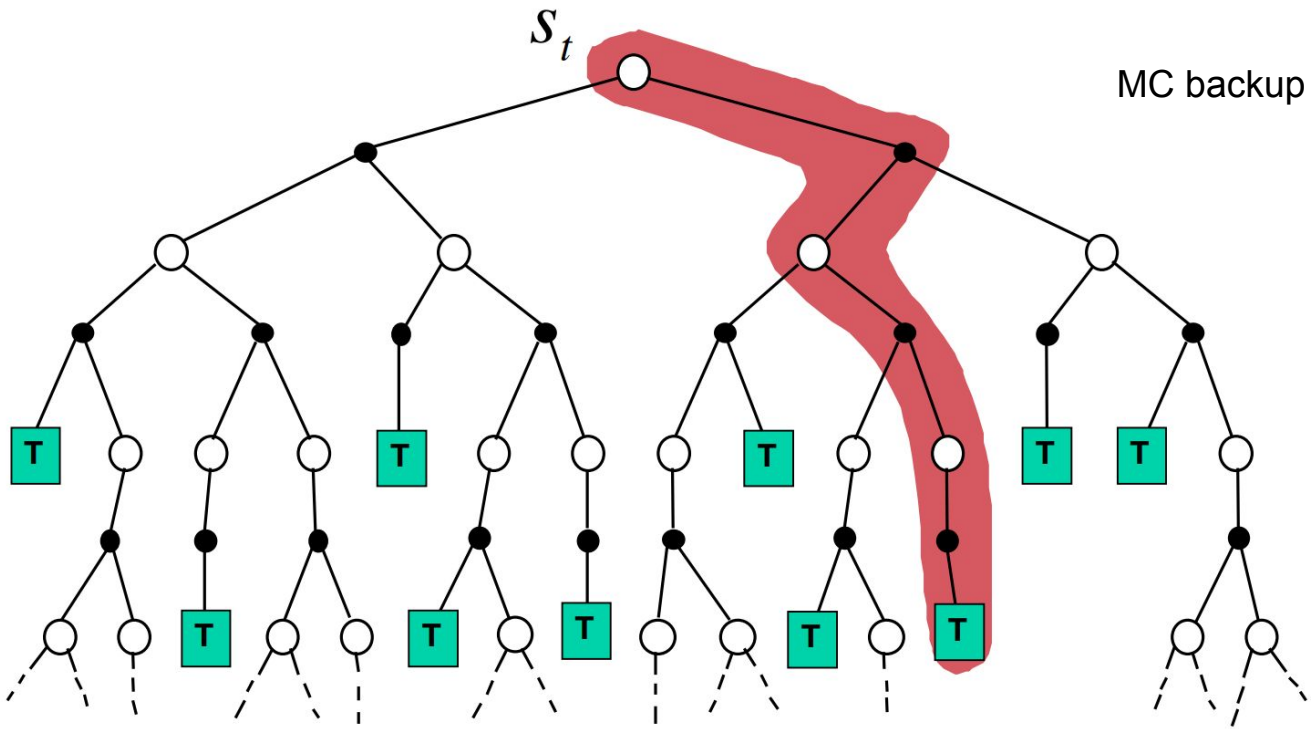
- Problem: learn Vπ online from experience under policy π
- Incremental every-visit Monte-Carlo:
  a. Update value V toward **actual** return G

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

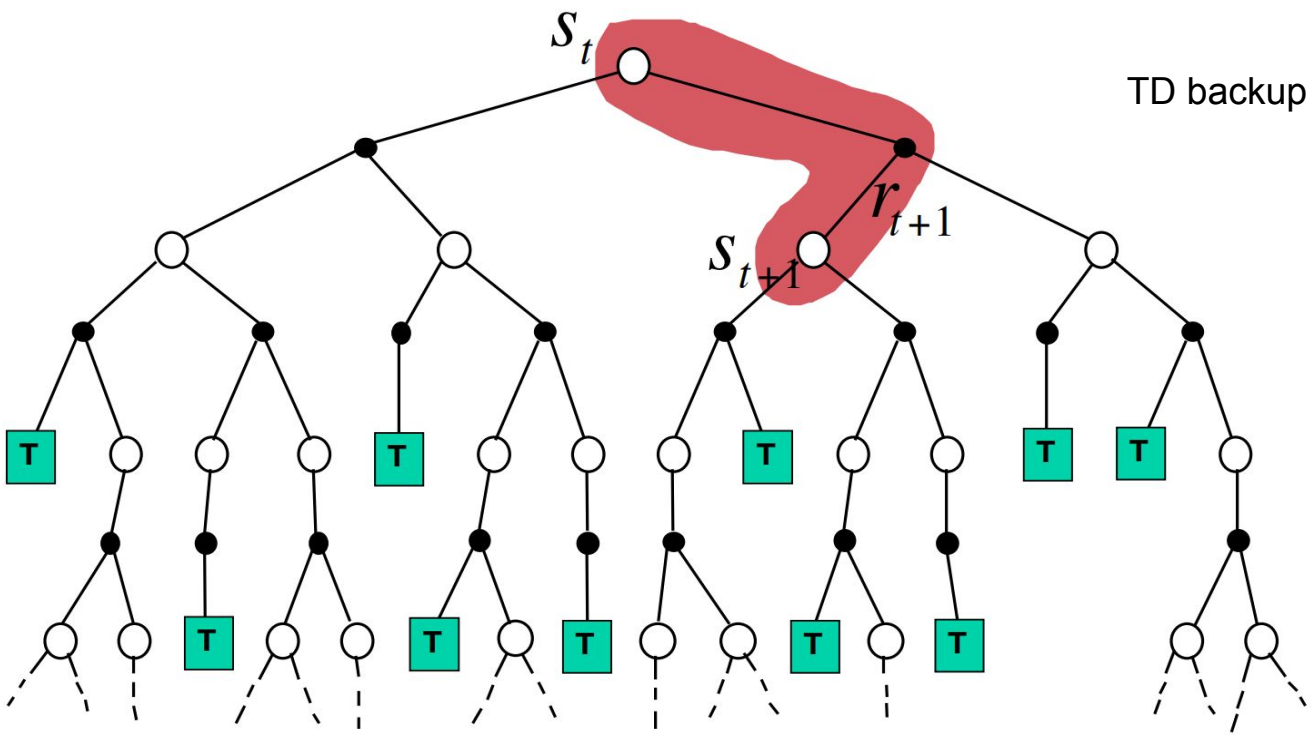
  b. But, only update the value **after** an entire episode
- Idea: update the value function using **bootstrap**
  a. Update value V toward **estimated** return

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

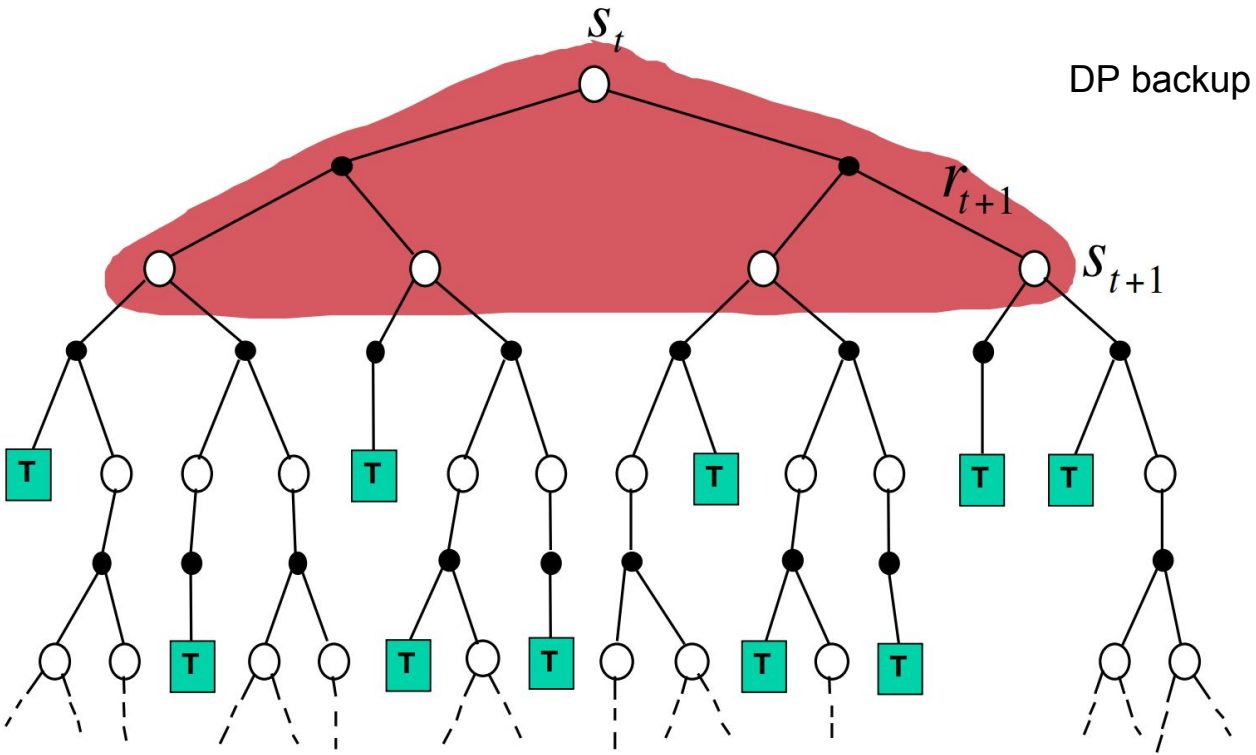
Temporal-Difference Learning



Temporal-Difference Learning



Temporal-Difference Learning



Temporal-Difference Learning TD(0)

- The simplest TD learning algorithm, TD(0)
- Update value V toward **estimated** return

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

a. $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target

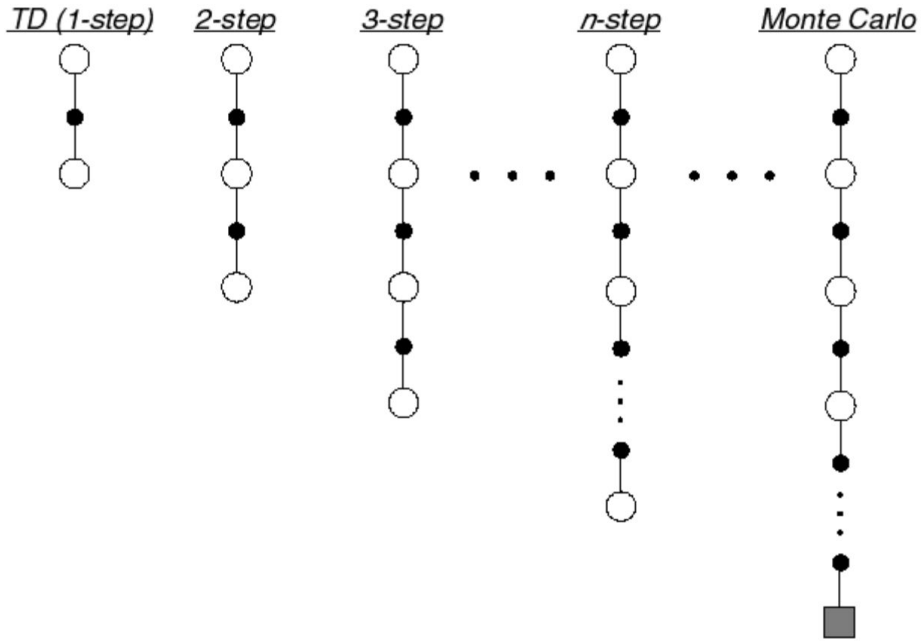
b. $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is the TD error

TD Bootstraps and Samples

- **Bootstrapping**: update involves an estimate
    a. MC does not bootstrap
    b. TD bootstraps
    c. DP bootstraps
- **Sampling**: update does not involve an expected value
    a. MC samples
    b. TD samples
    c. DP does not sample

Backup diagram for TD(n)

- Look farther into the future when you do TD backup (1, 2, 3, …, n steps)



Advantages of TD Learning

- TD 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
    a. TD can learn online after every step
    b. MC must wait until end of episode before return is known
- You can learn without the final outcome
    a. TD can learn from incomplete sequences
    b. TD works in continuing (non-terminating) environments

TD vs MC Learning: bias/variance trade-off

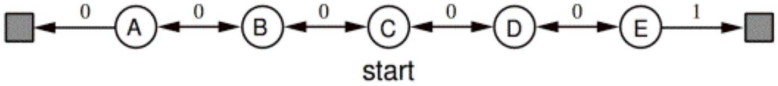
- Return $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$ is unbiased estimate of Vπ
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is unbiased estimate of Vπ
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of Vπ
- TD target is much lower variance than the return:
    a. Return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

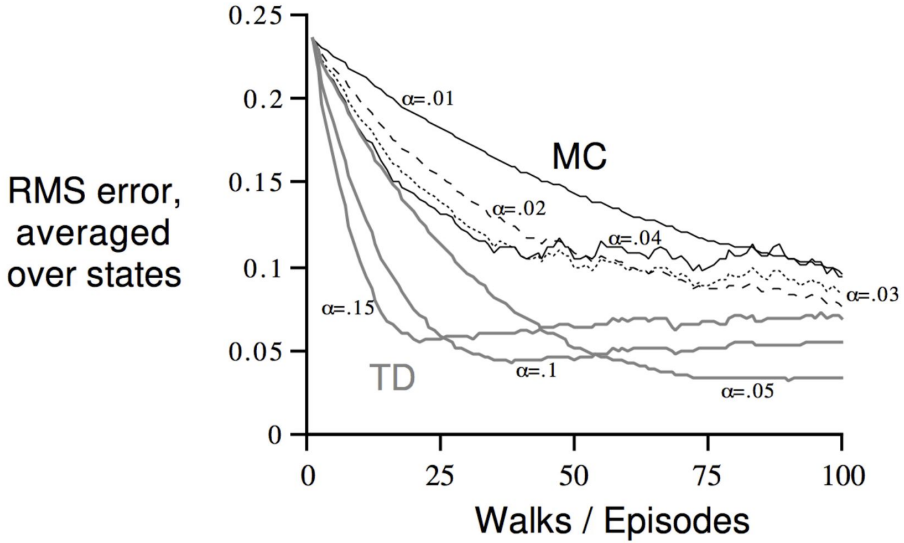
TD vs MC Learning

- TD and MC both converges, but which one is faster?

TD vs MC Learning

- TD and MC both converges, but which one is faster?
- Random walk example:





TD vs MC Learning: bias/variance trade-off

- MC has high variance, zero bias
    a. Good convergence properties
    b. (even with function approximation)
    c. Not very sensitive to initial value
    d. Very simple to understand and use
- TD has low variance, some bias
    a. Usually more efficient than MC
    b. TD(0) converges to $V\pi$
    c. (but not always with function approximation)

On-Policy TD control: Sarsa

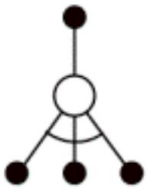
- Turn TD learning into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize $Q(s, a)$ arbitrarily
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)
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'; a \leftarrow a';$
    until s is terminal

Off-Policy TD control: Q-learning

One - step Q - learning :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$



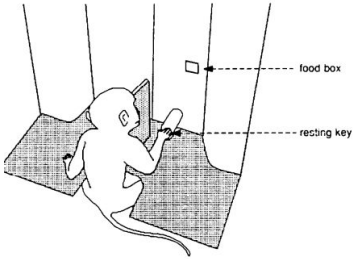
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$;
  until s is terminal

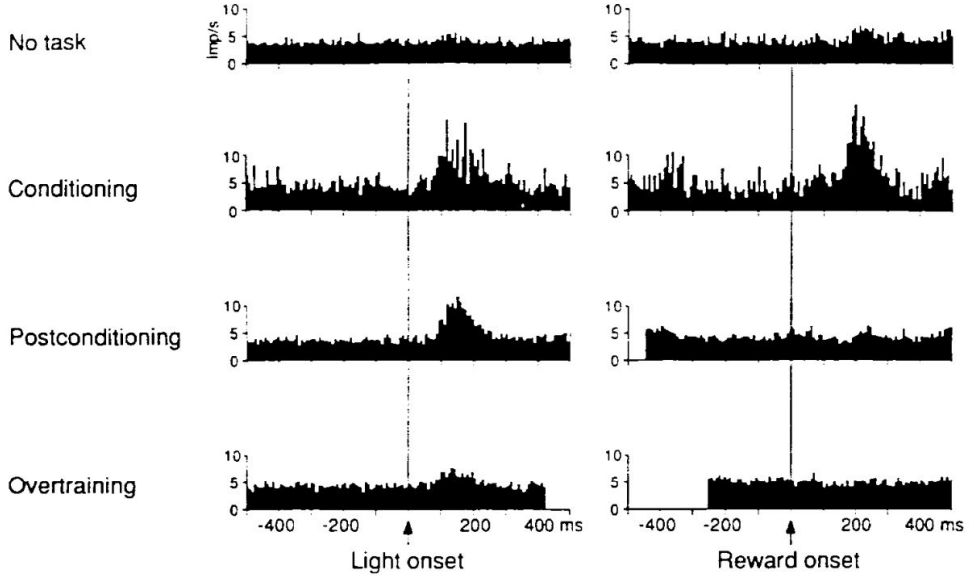
TD Learning

- TD methods approximates DP solution by minimizing TD error
- Extend prediction to control by employing some form of policy iteration
    a. On-policy control: Sarsa
    b. Off-policy control: Q-learning
- TD methods bootstrap and sample, combining aspects of DP and MC methods

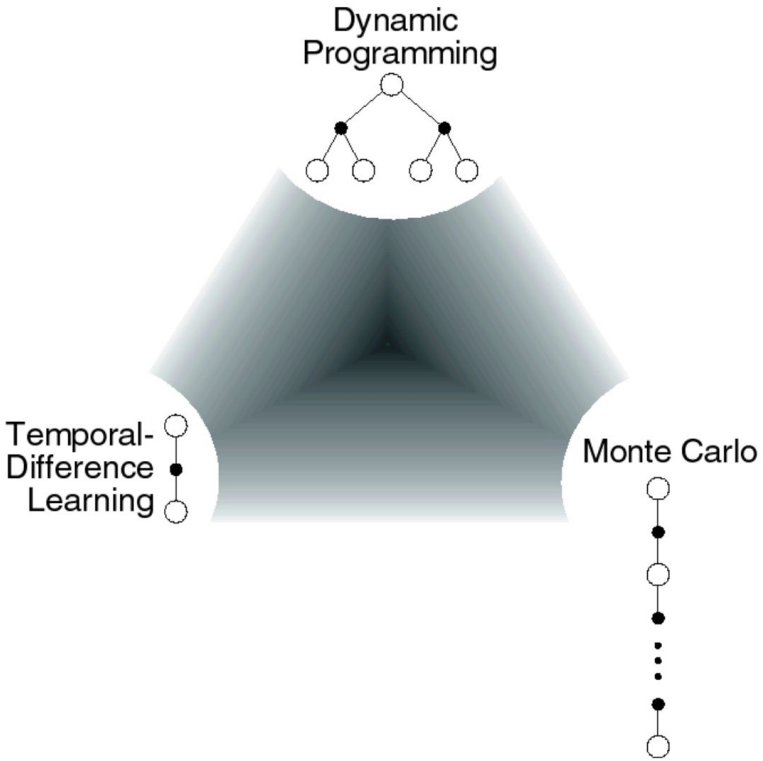
Dopamine Neurons and TD Error

- Wolfram Schultz, Peter Dayan, P. Read Montague. A Neural Substrate of Prediction and Reward, 1992





Summary


Dynamic
Programming
Temporal-
Difference
Learning
Monte Carlo

Questions

- What is common to all three classes of methods? – DP, MC, TD
- What are the principle strengths and weaknesses of each?
- What are the principal things missing?
- What does the term bootstrapping refer to?
- What is the relationship between DP and learning?