

# CS 581 – ADVANCED ARTIFICIAL INTELLIGENCE

## TOPIC: REINFORCEMENT LEARNING



**Mustafa Bilgic**



<http://www.cs.iit.edu/~mbilgic>



<https://twitter.com/bilgicm>

# REFERENCES

- Reinforcement Learning – an Introduction
  - by Sutton & Barto
  - <http://incompleteideas.net/book/the-book-2nd.html>
- Introduction to RL lectures
  - by DeepMind; David Silver
  - <https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver>

# REQUIRED BACKGROUND

## ○ MDPs

- Value function  $V$ , action-value function  $Q$ , policy  $\pi$
- Bellman equations
- Value iteration
- Policy iteration

## ○ Multi-armed bandits

- Exploration vs exploitation trade-off
- $\epsilon$ -greedy approach

# RL AND MDPs

- MDPs are building blocks for RL
- RL has the additional complexity that the agent does not have access to the full specification of the MDP. For e.g.,
  - Transition probabilities are often unknown
  - Reward function is often unknown

# PREDICTION AND CONTROL

- Prediction
  - Given a policy, estimate the value function
- Control
  - Learn the optimal policy

# MODEL-FREE VS MODEL-BASED

- Model-free:
  - The agent does not have and does not learn a model of the how the environment works
- Model-based:
  - The agent learns/improves a model of the environment
- Note: we are not talking about an approximate “model” of a state representation (such as DL for value estimations of states); rather, we mean model of the environment, such as transition probabilities

# OUTLINE

- Prediction
  - Monte Carlo methods
  - Temporal-difference learning, specifically TD(0)
  - Unified view: TD( $\lambda$ )
- Control
  - Monte Carlo methods
  - Temporal-difference learning
    - Sarsa, N-step TD, TD( $\lambda$ )
  - Q-learning
- Approximate methods
  - MC prediction
  - TD prediction

# MODEL-FREE PREDICTION



# MONTÉ CARLO PREDICTION

- Task: prediction
  - Policy is given; compute the value functions
- Unknown environment dynamics
  - If the transition probabilities and the reward function were given, we could use the algorithms we saw earlier for complete MDPs
- Learn from *experience* – samples of episodes
- *Episode* is a sequence of state, action, reward triplets
- Assume all episodes reach a terminal state
- Basic idea: expectation = average over samples

# MC PREDICTION – $V$

- Given a policy  $\pi$
- Initialize  $V(s)$  for all  $s$
- Loop:
  - Sample an episode:  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
  - For each state  $s$  in the episode, compute the cumulative discounted reward starting from that state  $s$
- $V(s)$  is the average cumulative discounted reward starting from state  $s$
- If  $s$  appears multiple times in a single episode (i.e., loops):
  - First-visit MC: for each episode, only the first appearance of the state  $s$  is considered
  - Every-visit MC: every appearance is considered and averaged accordingly
- Converges to the true values as the number of visits approach infinity

# FIRST-VISIT MC PREDICTION

## First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy  $\pi$  to be evaluated

Initialize:

$V(s) \in \mathbb{R}$ , arbitrarily, for all  $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :

$G \leftarrow \gamma G + R_{t+1}$

Unless  $S_t$  appears in  $S_0, S_1, \dots, S_{t-1}$ :

Append  $G$  to  $Returns(S_t)$

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

Figure from <http://incompleteideas.net/book/the-book-2nd.html>

# MC PREDICTION – Q

- Given a policy  $\pi$
- Initialize  $Q(s, a)$  for all  $(s, a)$  pairs
- Loop:
  - Sample an episode:  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
  - For each state  $(s, a)$  pair in the episode, compute the cumulative discounted reward starting from that state  $s$  and taking action  $a$
- $Q(s, a)$  is the average cumulative discounted reward starting from state  $s$  and taking action  $a$
- Advantage of estimating Q instead of V:
  - Can use Q values to find a better policy, i.e., for control
  - Caveat: we need to explore other actions to find a better one (remember the exploration vs exploitation trade-off)

# INCREMENTAL UPDATE

- $\mu_{n+1} = \mu_n + \frac{1}{n+1} (x_{n+1} - \mu_n)$
- $\mu_{n+1} = \mu_n + \alpha (x_{n+1} - \mu_n)$
- Using a fixed  $\alpha$  instead of  $\frac{1}{n+1}$  allows us to handle non-stationary cases where the dynamics of the system changes
  - Recent experiences count more than earlier ones

# MC UPDATE

- $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-1} R_T$
- Average update
  - $V_t(S_t) = V_t(S_t) + \frac{1}{N(S_t)} (G_t - V_t(S_t))$
- Nonstationary update
  - $V_t(S_t) = V_t(S_t) + \alpha (G_t - V_t(S_t))$

# TEMPORAL DIFFERENCE

- In MC, each episode had to end at a terminal state
- MC ignored the Bellman equations
- Can we learn from partial/incomplete episodes?
  - Yes, but we need estimates of the values
- The method we will use is called Temporal Difference (TD) learning

# TD PREDICTION

- Given a policy  $\pi$
- For an episode:
  - MC
    - $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$
    - Need to wait till the end to calculate  $G_t$
  - TD: take one step and use Bellman equation
    - $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- This is called TD(0) because it updates values based on a single look ahead



# TD(0) PREDICTION

## Tabular TD(0) for estimating $v_\pi$

Input: the policy  $\pi$  to be evaluated

Algorithm parameter: step size  $\alpha \in (0, 1]$

Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

$A \leftarrow$  action given by  $\pi$  for  $S$

        Take action  $A$ , observe  $R, S'$

$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

    until  $S$  is terminal

Figure from <http://incompleteideas.net/book/the-book-2nd.html>

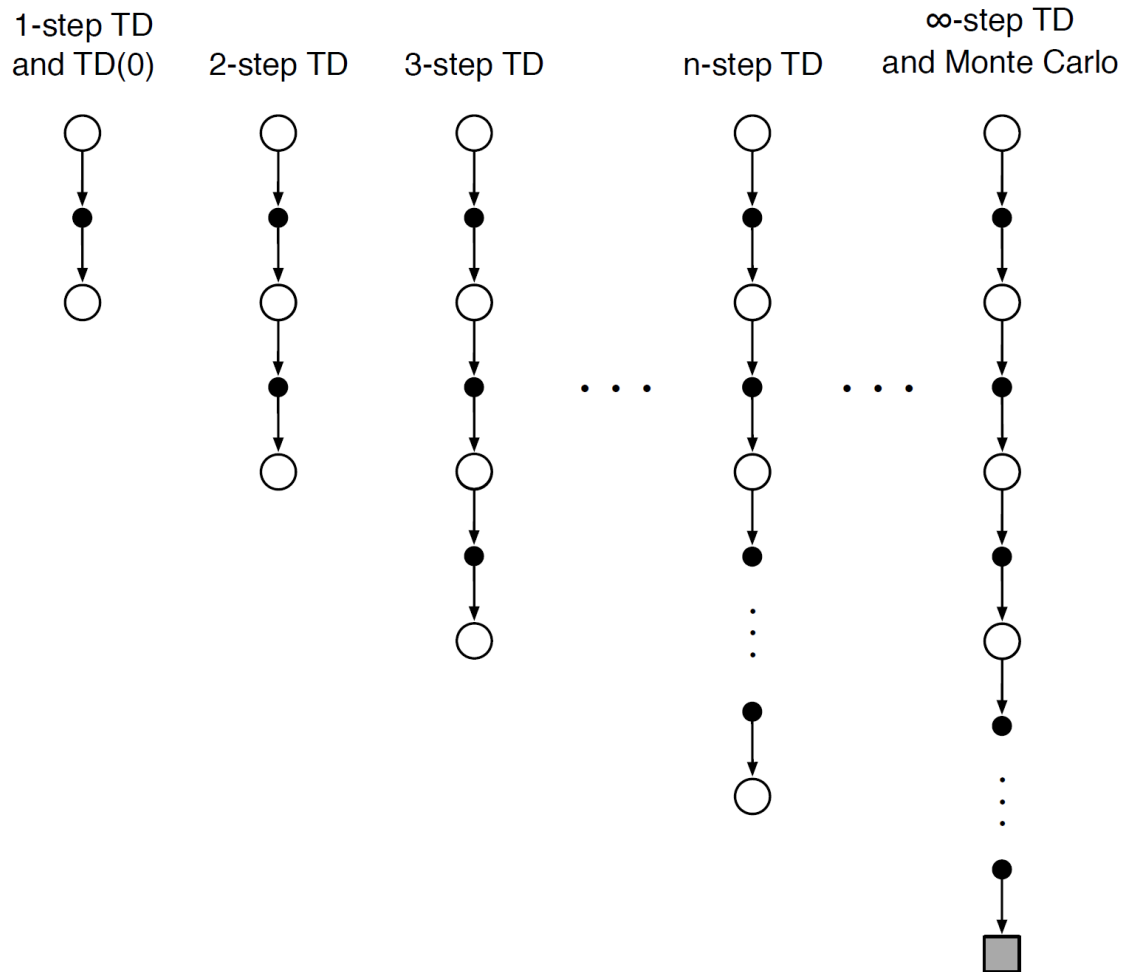
# MC vs TD(0)

- Target
  - MC uses  $G_t$ ; TD(0) uses  $R_{t+1} + \gamma V(S_{t+1})$
  - MC does not use the Bellman equations whereas TD(0) does
- Bootstrapping
  - MC does not need an initial estimate of  $V$ , whereas TD(0) bootstraps the  $V$  values
- Episodic vs continuing
  - MC updates the estimates after reaching the terminal state
  - TD(0) updates the estimates after each action
  - MC works on episodic tasks whereas TD(0) works for both episodic tasks and for continuing tasks
- Bias/Variance trade-off
  - MC estimate is unbiased but is high variance
  - TD(0) estimate is biased but is low variance
- Convergence
  - Both converge to the true values (in the simplest case of tabular state representation)
  - TD typically converges faster than MC

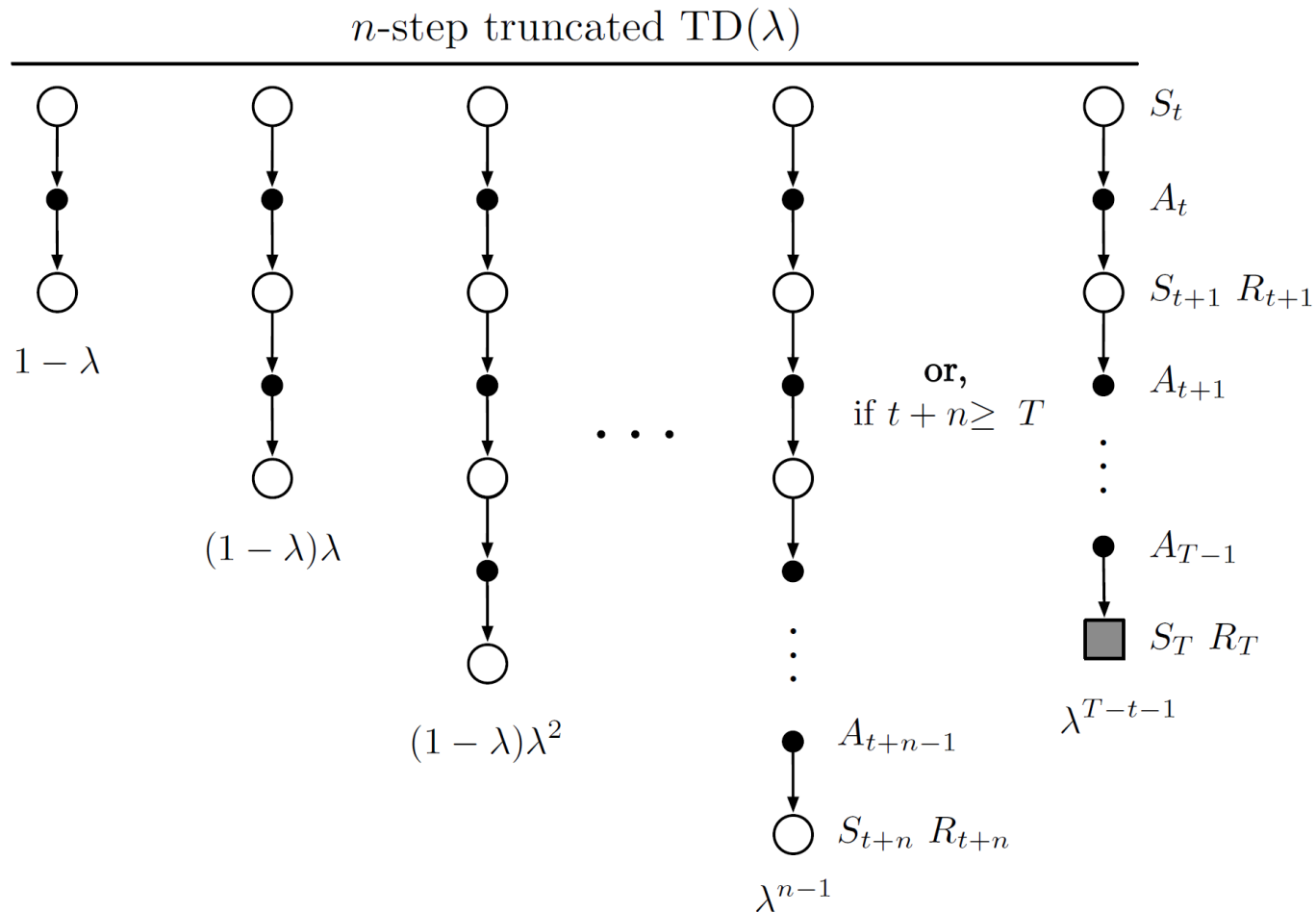
## N-STEP TD

- TD(0) bootstraps the value functions and looks at one step in the future
- TD(n) is generalization that looks at  $n+1$  steps to the future
- TD( $\infty$ ) is equivalent to MC
- TD( $\lambda$ ) considers all steps with proper weighting

# N-STEP TD



# TD( $\lambda$ )



# MODEL-FREE CONTROL

# ON-POLICY VS OFF-POLICY CONTROL

- On-policy control
  - The behavior/experience is generated by the same policy  $\pi$  that we are trying to improve
- Off-policy control
  - The behavior/experience is generated by a behavior policy  $b$  and we are trying to learn/improve policy  $\pi$

# ON-POLICY CONTROL



# MC CONTROL

- Unlike prediction, we are not given a policy
- The environment dynamics are unknown
- The agent needs to find an optimal policy through experience
  - samples of episodes
- Can't we combine policy iteration of MDPs and MC sampling?
  - Yes, but
  - We need to learn a  $Q$  function;  $V$  function would not be useful because we do not have the transition function to read the policy from  $V$  values
  - We cannot sample from a deterministic policy; otherwise, the agent does not learn
  - We need to balance exploration vs exploitation
    - We will use  $\epsilon$ -greedy approach
    - During sampling an episode:
      - With  $\epsilon$  probability, choose a random action
      - Otherwise, choose the action recommended by the policy

# MC CONTROL – PSEUDOCODE

- Initialize  $Q(s, a)$  for all  $(s, a)$  pairs
- Initialize a policy  $\pi$
- Loop:
  - Sample an episode:  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$ 
    - For choosing an action  $A_i$  at state  $S_i$ :
      - With  $\epsilon$  probability, choose a random action
      - Otherwise, choose the action recommended by the current policy
  - For each state  $(s, a)$  pair in the episode, compute the cumulative discounted reward starting from that state  $s$  and taking action  $a$
  - Update  $Q$  using the sample averages
  - Update  $\pi$  using the updated  $Q$

# TD CONTROL – SARSA

- Like MC Control, except we use the one step look ahead instead of waiting till the episode terminates
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
- Because we use  $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ , this algorithm is also called SARSA
- Like MC Control, we need to introduce randomness into choosing actions, rather than strictly following a deterministic policy
  - Use  $\epsilon$ -greedy approach

# SARSA ALGORITHM

## Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in S^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

Initialize  $S$

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

Loop for each step of episode:

Take action  $A$ , observe  $R, S'$

Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

until  $S$  is terminal

$\pi = \arg\max Q$

Figure from <http://incompleteideas.net/book/the-book-2nd.html>

$$\varepsilon = \frac{1}{K}$$

# OFF-POLICY CONTROL

# Q-LEARNING

## Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -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

Figure from <http://incompleteideas.net/book/the-book-2nd.html>

# APPROXIMATE METHODS

$$s \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$s \begin{bmatrix} \cdot & A \end{bmatrix}$$

- So far, we assumed we could represent the  $V$  and  $Q$  functions as tables
- This is unrealistic in real-world settings
- When the state and/or action space is large, we need to approximate the  $V$  and  $Q$  functions
- $v(s; \mathbf{w}) \approx v_{\pi}(s)$ 
  - $\mathbf{w}$  can be the weights of linear model, a neural network, etc.
- A good/useful model allows one to generalize from one state to another
- We can use supervised learning methods, with some caveats
  - For e.g., the data is not stationary, the sequences in an episode are not i.i.d., etc.

# PREDICTION

- MSVE =  $\sum_s P(s)[v_\pi(s) - v(s; \mathbf{w})]^2$
- Gradient-based approach
  - $\mathbf{w}_{i+1} = \mathbf{w}_i - \frac{1}{2}\alpha \nabla [v_\pi(s) - v(s; \mathbf{w}_i)]^2$
  - $\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha [v_\pi(s) - v(s; \mathbf{w})] \nabla v(s; \mathbf{w}_i)$
- Replace  $v_\pi(s)$  with its MC or TD estimate



# MC PREDICTION

## Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy  $\pi$  to be evaluated

Input: a differentiable function  $\hat{v} : \mathcal{S} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Algorithm parameter: step size  $\alpha > 0$

Initialize value-function weights  $\mathbf{w} \in \mathbb{R}^d$  arbitrarily (e.g.,  $\mathbf{w} = \mathbf{0}$ )

Loop forever (for each episode):

    Generate an episode  $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$  using  $\pi$

    Loop for each step of episode,  $t = 0, 1, \dots, T - 1$ :

$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$

Figure from <http://incompleteideas.net/book/the-book-2nd.html>

# TD PREDICTION

## Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy  $\pi$  to be evaluated

Input: a differentiable function  $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$  such that  $\hat{v}(\text{terminal}, \cdot) = 0$

Algorithm parameter: step size  $\alpha > 0$

Initialize value-function weights  $\mathbf{w} \in \mathbb{R}^d$  arbitrarily (e.g.,  $\mathbf{w} = \mathbf{0}$ )

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A \sim \pi(\cdot|S)$

        Take action  $A$ , observe  $R, S'$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})] \nabla \hat{v}(S, \mathbf{w})$

$S \leftarrow S'$

    until  $S$  is terminal

Figure from <http://incompleteideas.net/book/the-book-2nd.html>

# OTHER TOPICS

- Control with approximate functions
  - Estimate the  $Q$  function instead of the  $V$  function
- Policy gradient methods
  - The policy is a parameterized function that can select actions without going indirectly through the value functions
- Eligibility traces
  - Efficient and iterative implementation of TD(n)
- Off policy learning
  - Two policies: a *target policy* – the policy being learned, and a *behavior policy* – the policy that generates the sequences