CS 581 – Advanced Artificial Intelligence

Topic: Reinforcement Learning

Mustafa Bilgic

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

References

- Reinforcement Learning – an Introduction
  - by Sutton & Barto
  - http://incompleteideas.net/book/the-book.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 π
  - Bellman equations
  - Value iteration
  - Policy iteration
- Multi-armed bandits
  - Exploration vs exploitation trade-off
  - ϵ-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 π given $\Rightarrow V$, Q
  - Monte Carlo methods
  - Temporal-difference learning, specifically TD(0)
  - Unified view: TD(λ)
- Control π not given
  - Monte Carlo methods
  - Temporal-difference learning
    - Sarsa, N-step TD, TD(λ)
  - Q-learning
- Approximate methods
  - MC prediction
  - TD prediction
  - Semi-gradient SARSA control

Model-Free Prediction

Monte 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 – Value

- Given a policy π

$\pi(S_0)$ $\pi(S_1)$

- 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 π 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 π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

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

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

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

      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

$Q(s, a)$

- Given a policy π
- ~~Initialize $Q(s, a)$ for all (s,a) pairs~~
- Loop:
  - Sample an episode: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  - 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 α 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

- $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$
- 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(0) Prediction

- Given a policy π
- 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(0): 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_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  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(∞) is equivalent to MC
- TD(λ) considers all steps with proper weighting

N-STEP TD

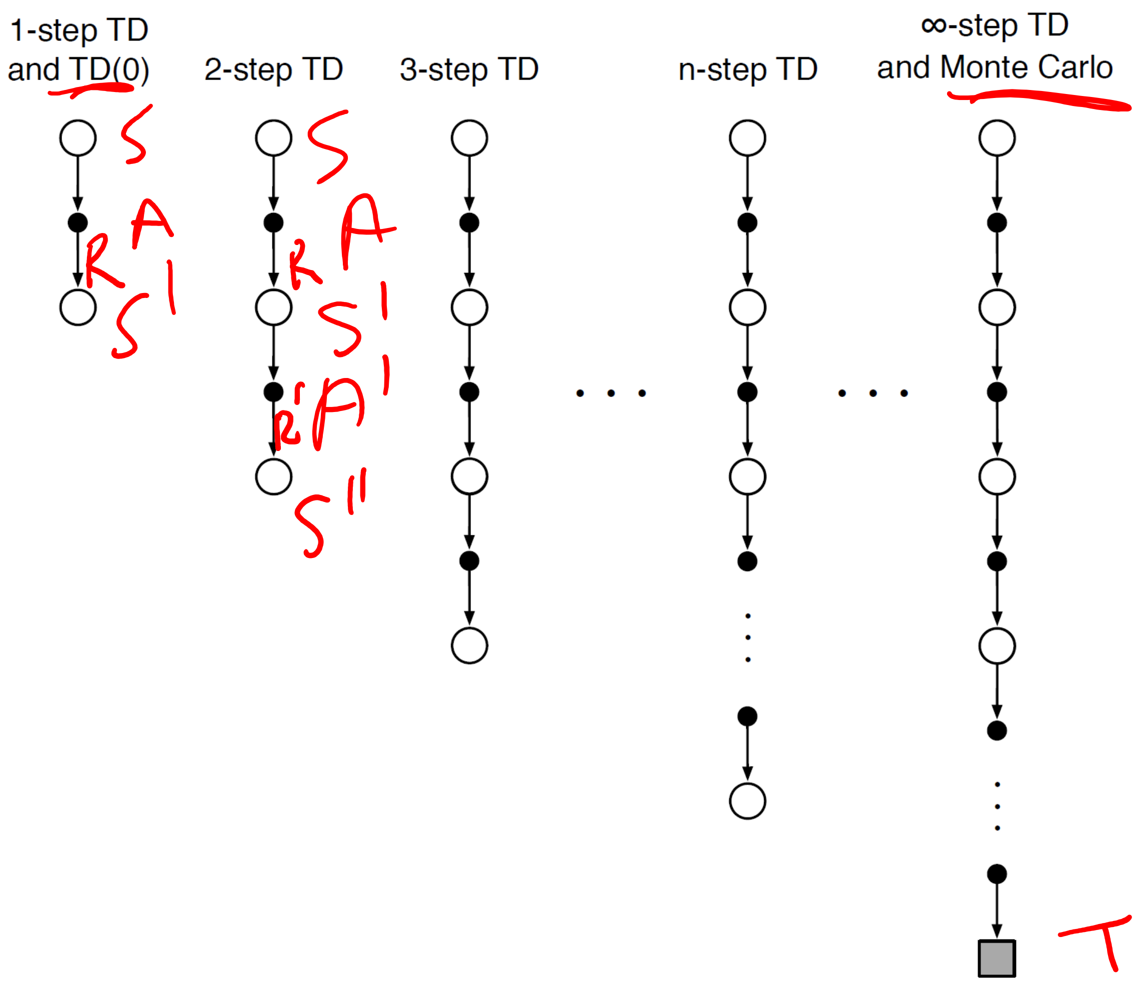

Figure from http://incompleteideas.net/book/the-book-2nd.html

TD(λ)

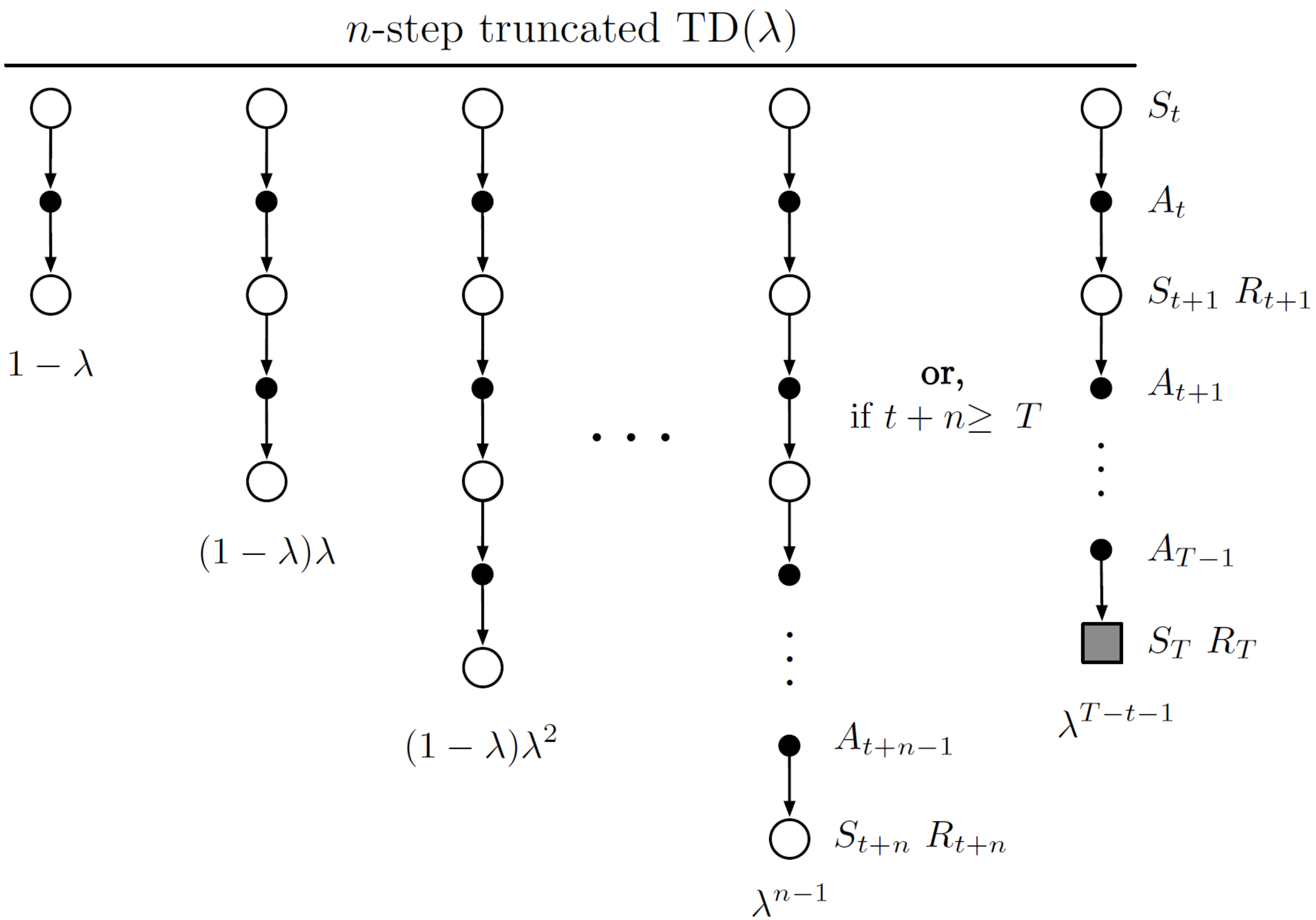

Figure from http://incompleteideas.net/book/the-book-2nd.html

Model-Free Control

On-Policy vs Off-Policy Control

- On-policy control
  - The behavior/experience is generated by the same policy π 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 π

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 ϵ–greedy approach
    - During sampling an episode:
      - With ϵ 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 π
- Loop:
  - Sample an episode: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    - For choosing an action A_i at state S_i:
      - With ϵ 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 π 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 ϵ-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 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

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

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 $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

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

Approximate Methods – Overview

- The state space is large
  - Tabular methods are impractical
- Need to estimate V or Q functions (or sometimes the policy directly without going through Q)
- Need our approximate methods to learn from experience and generalize to other cases
- Two tasks
  - Prediction: policy π is given; estimate the V and/or Q function
  - Control: find the best policy π

Approximate Prediction

- We are given a policy π
  - We want to estimate $V(s)$ and/or $Q(s, a)$
- The state space is large
  - We cannot use the tabular methods (MC, TD) from earlier slides directly
- Assume, for now, someone gives you the true $v_\pi(s)$
  - Like supervised learning
- Task: learn $\hat{v}(s; \boldsymbol{w}) \approx v_\pi(s)$

$\hat{v}(s; \boldsymbol{w})$

- $\hat{v}(s; \boldsymbol{w})$
  - Input is s; output is the value of the state
  - A regression problem
- $\hat{v}(s; \boldsymbol{w})$ can be
  - A linear model (e.g., linear regression), a decision tree, a neural network, etc.
- How do we represent the state s?
  - Features
    - For example, for chess, these could be indicator functions regarding the pieces and the position
  - Input to a convolutional neural network
- A good model generalizes from “seen” states to “unseen” states

Approximate Prediction – Objective Function

- MSE $= \sum_s P(s)[v_\pi(s) - \hat{v}(s; \boldsymbol{w})]^2$
  - $v_\pi(s)$ -- true value of the state s under policy
  - $P(s)$ -- how much we care about each state
  - $\hat{v}(s; \boldsymbol{w})$ -- our estimate of $v_\pi(s)$

Approximate Prediction – Gradient

- MSE $= \sum_s P(s)[v_\pi(s) - \hat{v}(s;\boldsymbol{w})]^2$
- Gradient-based approach; minimize MSE
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{current} - \frac{1}{2}\alpha\nabla[v_\pi(s) - \hat{v}(s;\boldsymbol{w}_{current})]^2$
  - $\boldsymbol{w}_{new} = \boldsymbol{w} + \alpha[v_\pi(s) - \hat{v}(s;\boldsymbol{w})]\nabla\hat{v}(s;\boldsymbol{w})$

Approximate Prediction – Linear Model

- Gradient-based approach
  - $\boldsymbol{w}_{new} = \boldsymbol{w} + \alpha[v_\pi(s) - \hat{v}(s;\boldsymbol{w})]\nabla\hat{v}(s;\boldsymbol{w})$
- Assume $\hat{v}(s;\boldsymbol{w})$ is a linear model
  - $\hat{v}(s;\boldsymbol{w}) = \boldsymbol{w} \times \boldsymbol{x}(s) = \sum w_i * x_i(s)$
  - where $\boldsymbol{x}(s)$ is a vector of feature values and $x_i(s)$ is the value of feature i on state s
- $\nabla\hat{v}(s;\boldsymbol{w}) = \nabla\big(\boldsymbol{w} \times \boldsymbol{x}(s)\big) = \boldsymbol{x}(s)$
  - $\boldsymbol{w}_{new} = \boldsymbol{w} + \alpha[v_\pi(s) - \hat{v}(s;\boldsymbol{w})]\boldsymbol{x}(s)$
- $\frac{\partial\hat{v}(s;\boldsymbol{w})}{\partial w_i} = x_i(s)$

Approximate Prediction – Practice

- $\boldsymbol{w}_{new} = \boldsymbol{w} + \alpha[v_\pi(s) - \hat{v}(s;\boldsymbol{w})]\nabla\hat{v}(s;\boldsymbol{w})$
- $v_\pi(s)$ is of course unknown
- Replace $v_\pi(s)$ with its MC or TD estimate
- MC
  - $v_\pi(S_t) = G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots$
  - $\boldsymbol{w}_{new} = \boldsymbol{w} + \alpha[G_t - \hat{v}(S_t;\boldsymbol{w})]\nabla\hat{v}(S_t;\boldsymbol{w})$
- TD(0)
  - $v_\pi(S_t) = R_t + \gamma\hat{v}(S_{t+1};\boldsymbol{w})$
  - $\boldsymbol{w}_{new} = \boldsymbol{w} + \alpha[R_t + \gamma\hat{v}(S_{t+1};\boldsymbol{w}) - \hat{v}(S_t;\boldsymbol{w})]\nabla\hat{v}(S_t;\boldsymbol{w})$
- TD(n) and TD(λ) are similar

Gradient MC Prediction

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
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 π
    Loop for each step of episode, $t = 0, 1, \dots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[G_t - \hat{v}(S_t,\mathbf{w}) \big] \nabla \hat{v}(S_t,\mathbf{w})$

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 π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ 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'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S is terminal

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

Approximate Control

- Estimate the Q function instead of V
- Design choices
  - $Q(s, a; \boldsymbol{w})$ – input is a state and an action; output is the value
  - $Q(s; \boldsymbol{w})$ – input is a state and outputs are values for each of the available actions
- The control methods from earlier slides, for example SARSA, can be used for approximate control

APPROXIMATE CONTROL – GRADIENT SARSA

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

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

Recommended Reading

- Chapter 16 of the reinforcement learning – applications and case studies
- TD-Gammon
- Checkers
- IBM Watson's wagering system
- Memory control
- Video games
- The game of Go
- Personalized web
- Therman soaring