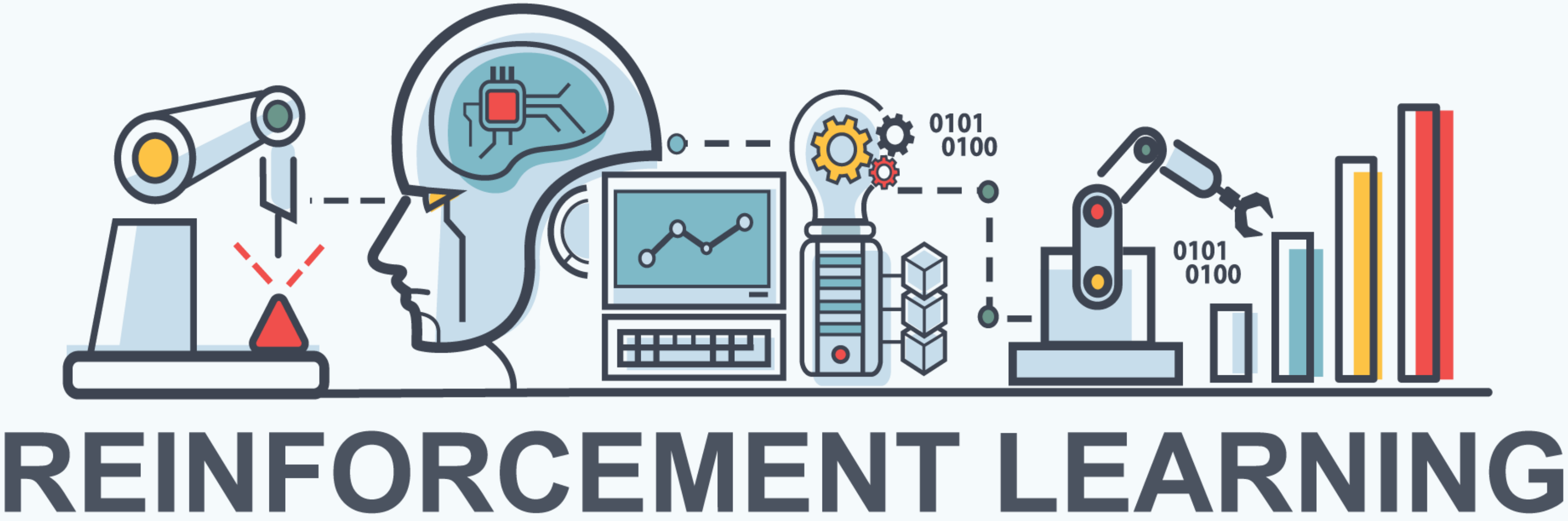


CP8319/CPS824
Lecture 12

Instructor: Nariman Farsad

* Some of the slides in this deck are adopted from courses offered David Silver, Emma Brunskill, and Sergey Levine.

Today's Agenda

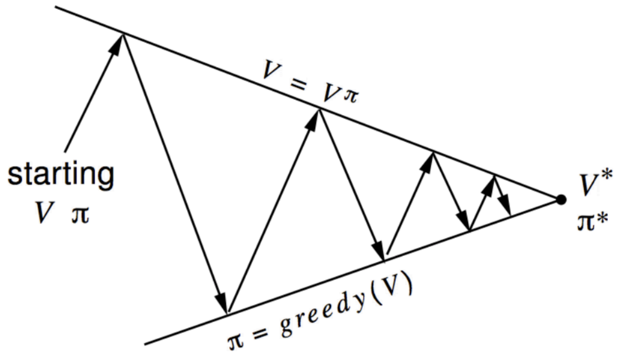
1. **Review of Previous Lectures**

2. Value Function Approximation Policy Evaluation

Policy Iteration: Known Model

Set $i = 0$
Initialize $\pi_0(s)$ randomly for all states s
While $i\ == \ 0$ or $\| \pi_i \ - \ \pi_{i-1} \|_1 > \ 0$ (L1-norm, measures if the policy changed for any state):

- $v^{\pi_i} \leftarrow$ MDP value function **policy evaluation** of π_i
- $\pi_{i+1} \leftarrow$ **Policy improvement** on v^{π_i}
- $i = i + 1$



Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

Model Free Policy Iteration

Set $i = 0$

Initialize $\pi_0(s)$ randomly for all states s

While $i == 0$ or $\| \pi_i - \pi_{i-1} \|_1 > 0$ (L1-norm, measures if the policy changed for any state):

- $Q^{\pi_i} \leftarrow$ MDP value function **MC policy Q evaluation** of π_i
- $\pi_{i+1} \leftarrow$ **ϵ-greedy Policy improvement** on Q^{π_i}
- $i = i + 1$

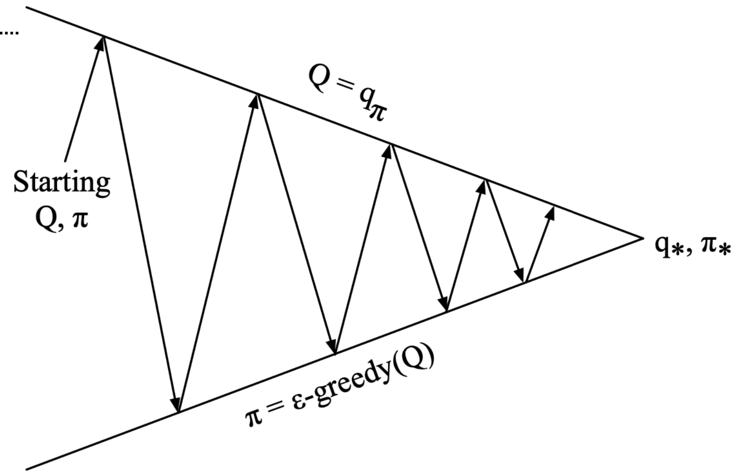
---

greedy(Q)

$$\pi_{i+1}(a|s) = \begin{cases} 1, & \text{if } a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, Q^{\pi_i}(s, a') \\ 0, & \mathit{otherwise} \end{cases}$$

ϵ-greedy(Q)

$$\pi_{i+1}(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon\,, & \text{if } a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, Q^{\pi_i}(s, a') \\ \epsilon/m\,, & \mathit{otherwise} \end{cases}$$



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement ϵ-greedy policy improvement

Greedy in the Limit of Infinite Exploration (GLIE)

Definition of GLIE

- All state-action pairs are visited an infinite number of times

$$\lim_{i \to \infty} N_i(s, a) \to \infty$$

- Behavior policy (policy used to act in the world) converges to greedy policy
  $\lim_{i \to \infty} \pi(a|s) \to \arg\max_a Q(s, a)$ with probability 1

- A simple GLIE strategy is ϵ-greedy where ϵ is reduced to 0 with the following rate: $\epsilon_i = 1/i$

Monte Carlo Online Control/On Policy Improvement

Initialize $Q(s,a) = 0, N(s,a) = 0\ \forall(s,a)$, Set $\epsilon = 1,\ k = 1$

$\pi_k = \epsilon$-greedy(Q) // Create initial ϵ-greedy policy

loop

  Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,T})$ given π_k

  $G_{k,t} = r_{k,t} + \gamma r_{k,t+1} + \gamma^2 r_{k,t+2} + \cdots \gamma^{T_i - 1} r_{k,T_i}$

  for $t = 1, \ldots, T$ **do**

    if First visit to (s,a) in episode k **then**

      $N(s,a) = N(s,a) + 1$

      $Q(s_t, a_t) = Q(s_t, a_t) + \frac{1}{N(s,a)}(G_{k,t} - Q(s_t, a_t))$

    end if

  end for

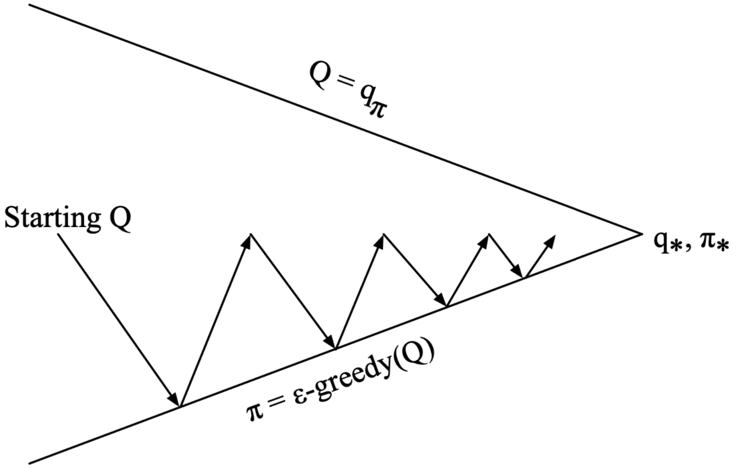
  $k = k + 1,\ \epsilon = 1/k$

  $\pi_k = \epsilon$-greedy(Q) // Policy improvement

end loop

SARSA For On-Policy Control

Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
Take $a_t \sim \pi(s_t)$ // Sample action from policy
Observe (r_t, s_{t+1})
loop
    Take action $a_{t+1} \sim \pi(s_{t+1})$
    Observe (r_{t+1}, s_{t+2})
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
    $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
    $t = t + 1$
end loop



Every time-step:

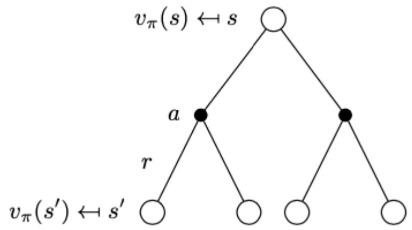
Policy evaluation Sarsa, $Q \approx q_\pi$

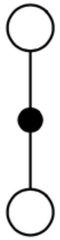
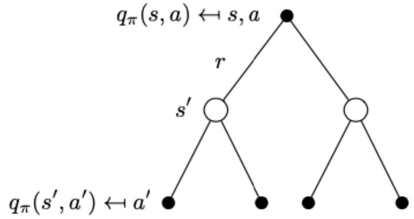
Policy improvement ϵ-greedy policy improvement

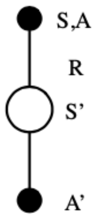
Q-Learning Algorithm

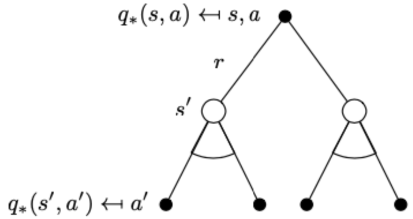
Initialize $Q(s,a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
Set π_b to be ϵ-greedy w.r.t. Q
loop
  Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
  Observe (r_t, s_{t+1})
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
  $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
  $t = t + 1$
end loop

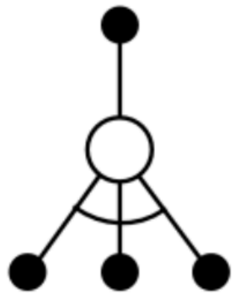
Relationship Between DP (Known) and TD (Unknown)

<table>
<tr><th></th><th>Full Backup (DP)</th><th>Sample Backup (TD)</th></tr>
<tr><td>Bellman Expectation Equation for $v_\pi(s)$</td><td>

Iterative Policy Evaluation</td><td>
TD Learning</td></tr>
<tr><td>Bellman Expectation Equation for $q_\pi(s,a)$</td><td>

Q-Policy Iteration</td><td>

Sarsa</td></tr>
<tr><td>Bellman Optimality Equation for $q_*(s,a)$</td><td>

Q-Value Iteration</td><td>
Q-Learning</td></tr>
</table>

Today's Agenda

1. Review of Previous Lectures

2. **Value Function Approximation Policy Evaluation**

Large-Scale Reinforcement Learning

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space
- How can we scale up the model-free methods for prediction and control from the last two lectures?

Value Function Approximation

- So far we have represented value function by a lookup table
  - Every state s has an entry $v(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$

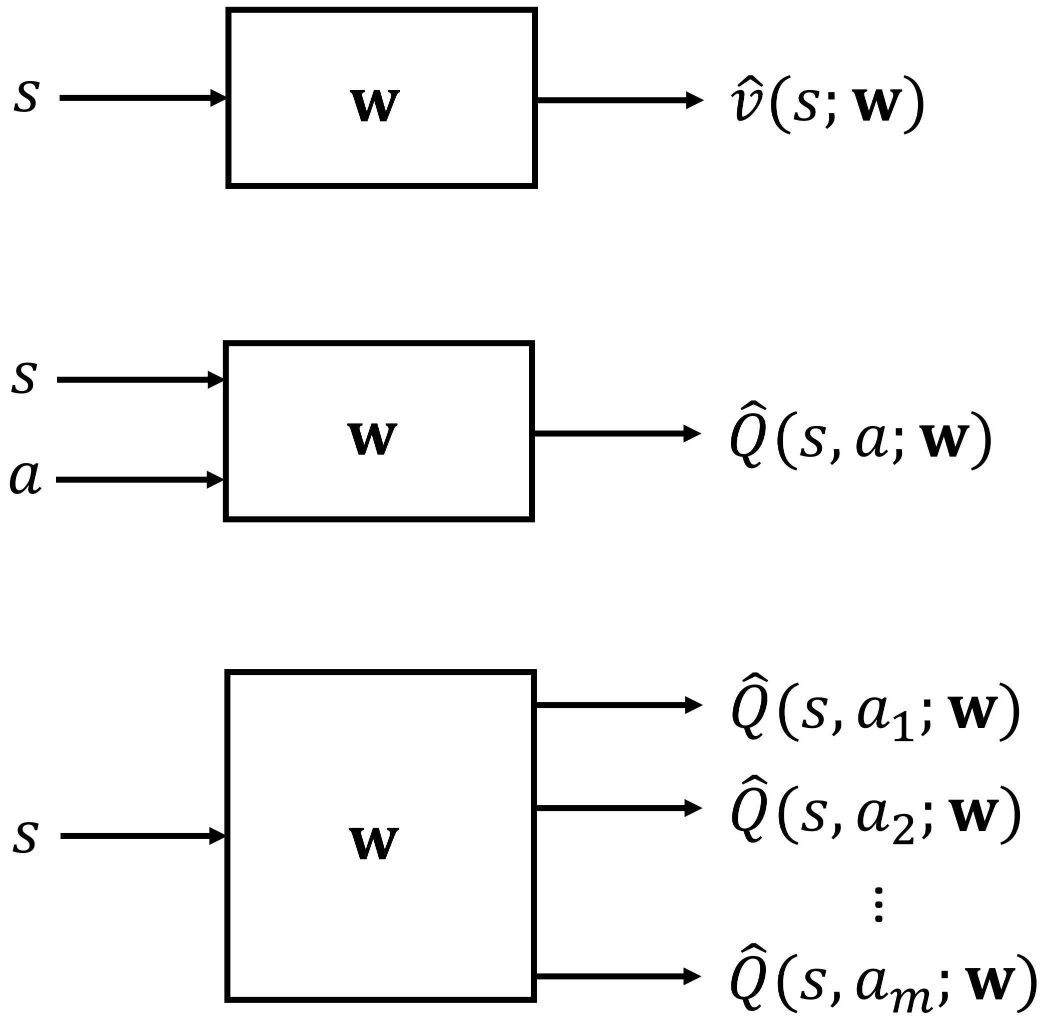
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually

- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$v^{\pi}(s) \approx \hat{v}(s; \mathbf{w})$$
$$\text{or } Q^{\pi} \approx \hat{Q}(s, a; \mathbf{w})$$

  - *Generalize* from seen states to unseen states
  - Update parameter $\mathbf{w}$ using MC or TD learning

Types of Value Function Approximation



Function Approximators

- Many possible function approximators including:
  - Linear combinations of features
  - Neural networks
  - Decision trees
  - Nearest neighbors
  - Fourier/ wavelet bases

- In this class we will focus on function approximators that are differentiable (Why?)

- Two very popular classes of differentiable function approximators
  - Linear feature representations (2 lectures)
  - Neural networks (Next 2 lectures)

- We require a training method that is suitable for <u>non-stationary</u>, <u>non-iid data</u>

Gradient Descent

- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_n} \end{pmatrix}$$

- To find a *local minimum* of $J(\mathbf{w})$:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

Value Function Approximation with an Oracle

- First assume we could query any state s and an oracle would return the true value for $v^{\pi}(s)$

- Therefore, we could have a set $\{(s_1, v^{\pi}(s_1)), (s_2, v^{\pi}(s_2)), \ldots\}$ of data

- Goal: Find the parameter vector $\mathbf{w}$ that minimizes the loss between a true value function $v^{\pi}(s)$ and its approximation $\hat{v}(s; w)$ as represented with a particular function/model class parameterized by $\mathbf{w}$.

- What does this remind you off?

How do we learn?

Batch Gradient Descent:

- Expensive to compute gradient for large dataset
- Computational and space complexity high

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Stochastic Gradient Descent:

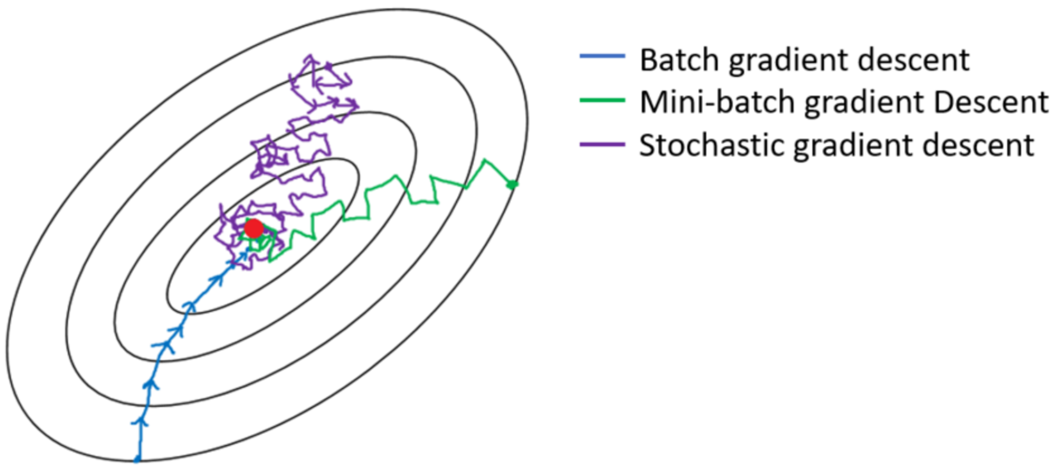
- Lots of random motion (slow to converge)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Mini-batch Gradient Descent (Mini-batch of size n):

- Hybrid between the two, still stochastic but less random

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$



Value Function Approx. By Stochastic Gradient Descent

- Goal: Find the parameter vector $\mathbf{w}$ that minimizes the loss between a true value function $v^{\pi}(s)$ and its approximation $\hat{v}(s; w)$ as represented with a particular function/model class parameterized by $\mathbf{w}$.
- Generally, use mean squared error and define the loss as

$$J(\mathbf{w}) = \mathbb{E}_{\pi}\left[\left(v^{\pi}(s) - \hat{v}(s; \mathbf{w})\right)^2\right]$$

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbb{E}_{\pi}\left[\left(v^{\pi}(s) - \hat{v}(s; \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{v}(s; \mathbf{w})\right]$$

- Stochastic gradient descent (SGD) *samples* the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(v^{\pi}(s) - \hat{v}(s; \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{v}(s; \mathbf{w})$$

- Expected update is equal to full gradient update

Model Free VFA Policy Evaluation

- Don't actually have access to an oracle to tell true $v^{\pi}(s)$ for any state s
- Now consider how to do model-free value function approximation for prediction / evaluation / policy evaluation without a model

What we did before

- Recall model-free policy evaluation from prior lectures
  - Following a fixed policy π for sampling
  - Goal is to estimate v^{π} and/or Q^{π}

- We did this by maintaining *a look-up table* to store estimates of v^{π} and/or Q^{π}
  - Update the look-up table estimate after each episode (Monte Carlo)
  - After each step (temporal difference)

- *Now*: in value function approximation, change the estimate update step to include fitting the function approximator

Feature Vectors

- The state is represented by a *feature vector*

$$\mathbf{x}(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

- For example:
  - Distance of a robot from "landmarks"
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation With Oracle

- Represent a value function (or state-action value function) for a particular policy with a weighted linear combination of features:

$$\hat{v}(s; \mathbf{w}) = \sum_{j=0}^{n} x_j(s) w_j = \mathbf{x}(s)^T \mathbf{w}$$

- Objective function is:

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v^\pi(s) - \hat{v}(s; \mathbf{w})\right)^2\right]$$

- Update rule is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(v^\pi(s) - \hat{v}(s; \mathbf{w})\right)\nabla_{\mathbf{w}}\hat{v}(s; \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(v^\pi(s) - \hat{v}(s; \mathbf{w})\right)\mathbf{x}(s)$$

Table Lookup Features

- Table lookup is a special case of linear value function approximation
- Using *table lookup features*

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

- Parameter vector $\mathbf{w}$ gives value of each individual state

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

Monte Carlo Value Function Approximation

- Return G_t is an unbiased but noisy sample of the true expected return $v^{\pi}(s_t)$
- Therefore, can reduce MC VFA to doing supervised learning on a set of (state,return) pairs: $\langle s_1, G_1\rangle, \langle s_2, G_2\rangle, \ldots, \langle s_T\ , G_T\rangle$
  - Substitute G_t for the true $v^{\pi}(s_t)$ when fit function approximator
- Concretely when using linear VFA for policy evaluation

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(G_t - \hat{v}(s; \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{v}(s; \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(G_t - \hat{v}(s; \mathbf{w})\big)\mathbf{x}(s)$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(G_t - \mathbf{x}(s)^T\mathbf{w})\mathbf{x}(s)$$

- Note: G_t may be a very noisy estimate of true return

MC Linear VFA for Policy Evaluation

Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$

loop

  Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,L_k})$ given π

  for $t = 1, \ldots, L_k$ **do**

    if First visit to (s) in episode k **then**

      $G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$

      Update weights:

    end if

  end for

  $k = k + 1$

end loop

MC Linear VFA for Policy Evaluation Example

Recall MC update: $\alpha(G_t - \mathbf{x}(s)^T\mathbf{w})\mathbf{x}(s)$

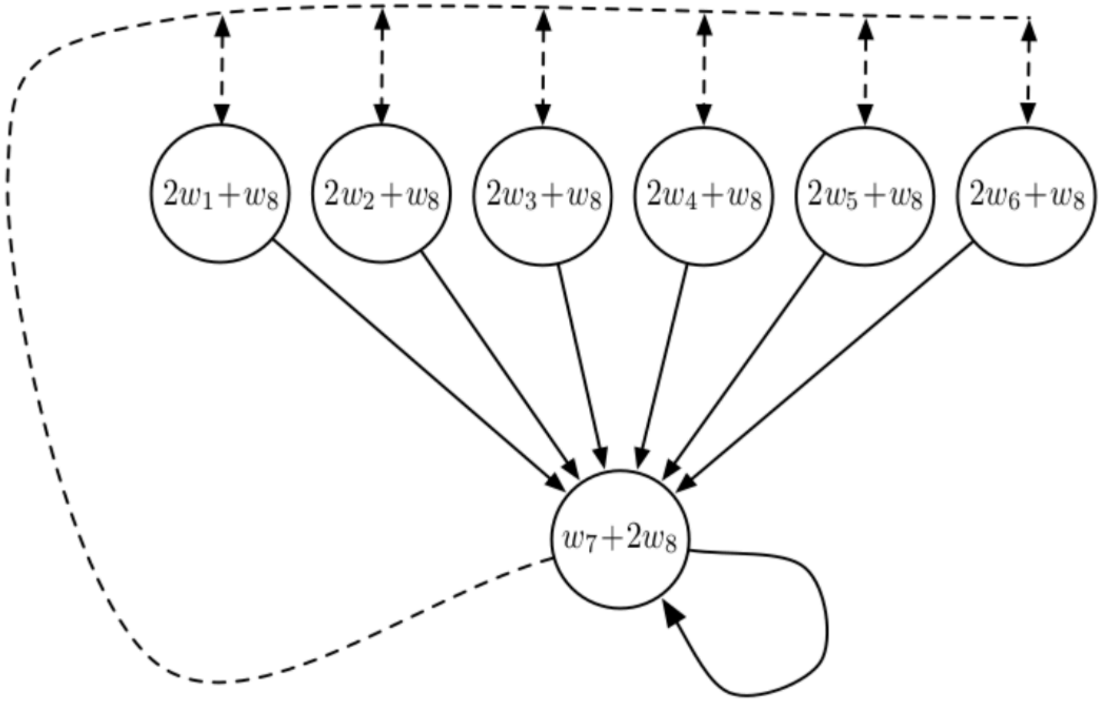
Two actions:

- a_1 goes to 7 (solid line)
- a_2 goes to 1-6 with 1/6 probability (dashed line)

Observe $s_1, a_1, 0, s_7, a_1, 0, s_7, a_1, 0, terminate$

Assume $\mathbf{w}_0 = [1,1,1,1,1,1,1,1]$,$\alpha = 0.5, \gamma$ =0.9

What is $\mathbf{w}_1$ after the first SGD update?



MC Linear VFA for Policy Evaluation Example

Recall MC update: $\alpha(G_t - \mathbf{x}(s)^T\mathbf{w})\mathbf{x}(s)$

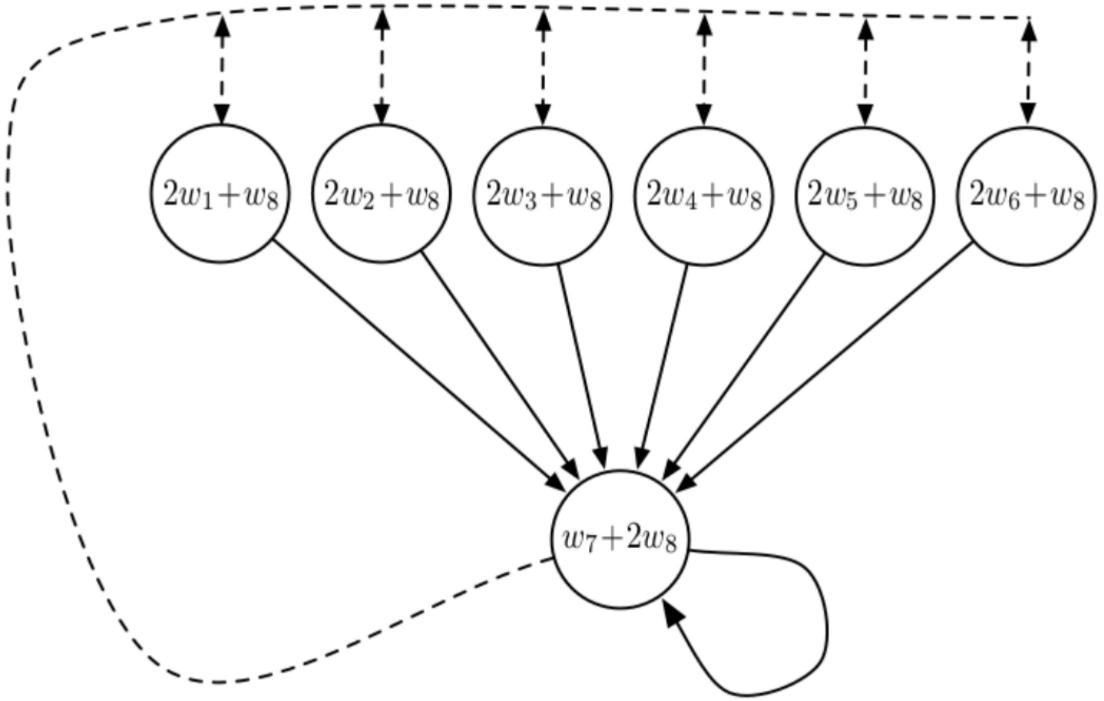
Two actions:

- a_1 goes to 7 (solid line)
- a_2 goes to 1-6 with 1/6 probability (dashed line)

Observe $s_1, a_1, 0, s_7, a_1, 0, s_7, a_1, 0, terminate$

Assume $\mathbf{w}_0 = [1,1,1,1,1,1,1,1], \alpha = 0.5, \gamma = 0.9$

What is $\mathbf{w}_1$ after the first SGD update?



Recall: Temporal Difference Learning w/ Lookup Table

- Uses bootstrapping and sampling to approximate $v^{\pi}(s)$
- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $v^{\pi}(s_t)$ toward estimated return $r_t + \gamma v^{\pi}(s_{t+1})$

$$v^{\pi}(s_t) = v^{\pi}(s_t) + \alpha([r_t + \gamma v^{\pi}(s_{t+1})] - v^{\pi}(s_t))$$

- $r_t + \gamma v^{\pi}(s_{t+1})$ is called the *TD target*
- $\delta_t = r_t + \gamma v^{\pi}(s_{t+1}) - v^{\pi}(s_t)$ is called the *TD error*
- a biased estimate of the true value $v^{\pi}(s)$
- Represent value for each state with a separate table entry

Temporal Difference (TD(0)) Learning with VFA

- In value function approximation, the target is $r + \gamma \hat{v}^{\pi}(s'; \mathbf{w})$, a biased and approximated estimate of the true value $v^{\pi}(s)$

- Can reduce doing TD(0) learning with value function approximation to supervised learning on a set of data pairs:

$$\langle s_1, r_1 + \gamma \hat{v}^{\pi}(s_2; \mathbf{w}) \rangle, \langle s_2, r_2 + \gamma \hat{v}^{\pi}(s_3; \mathbf{w}) \rangle, \ldots$$

- Find weights to minimize mean squared error:

$$J(\mathbf{w}) = \mathbb{E}_{\pi}\left[\left(r_j + \gamma \hat{v}^{\pi}(s_{j+1}; \mathbf{w}) - \hat{v}(s_j; \mathbf{w})\right)^2\right]$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(r_j + \gamma \mathbf{x}(s_{j+1})^T \mathbf{w} - \mathbf{x}(s_j)^T \mathbf{w}\right) \mathbf{x}(s_j)$$

- Therefore, we are suing 3 forms of approximation, what are they?

TD(0) Linear VFA for Policy Evaluation

1: Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$
2: **loop**
3: Sample tuple (s_k, a_k, r_k, s_{k+1}) given π
4: Update weights:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$$

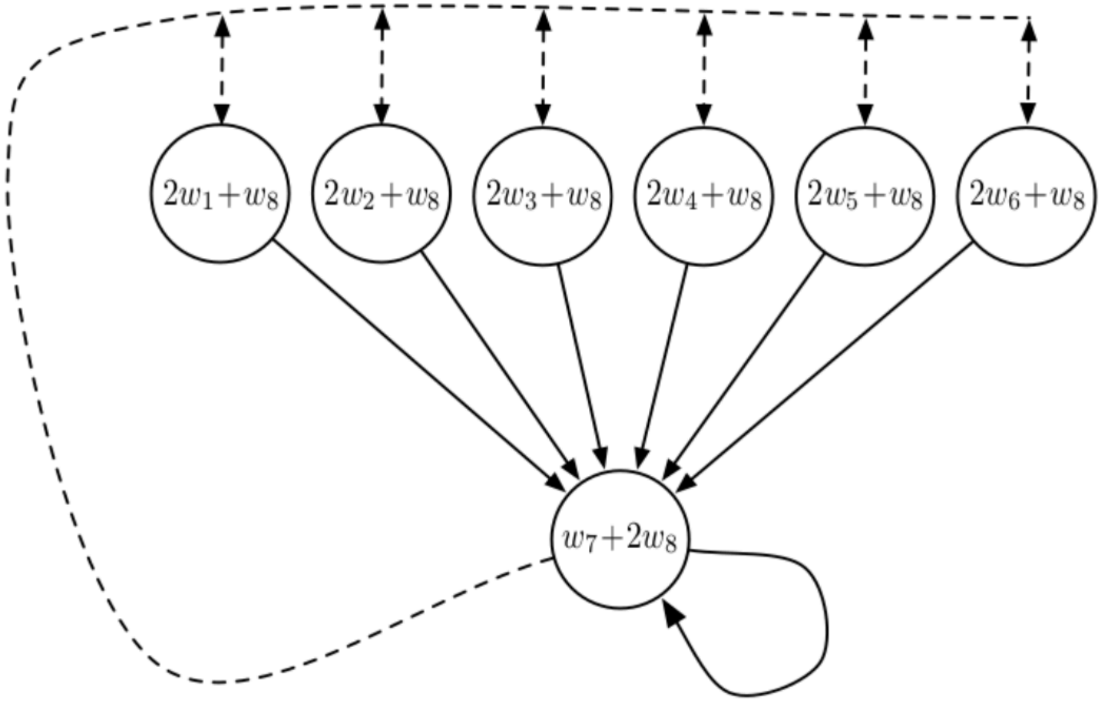
5: $k = k + 1$
6: **end loop**

TD(0) Linear VFA for Policy Evaluation Example

Recall MC update: $\alpha(G_t - \mathbf{x}(s)^T \mathbf{w})\mathbf{x}(s)$
Two actions:

- a_1 goes to 7 (solid line)
- a_2 goes to 1-6 with 1/6 probability (dashed line)

Observe $(s_1, a_1, 0, s_7)$
Assume $\mathbf{w}_0 = [1,1,1,1,1,1,1,1], \alpha = 0.5, \gamma = 0.9$
What is $\mathbf{w}_1$ after the first SGD update?

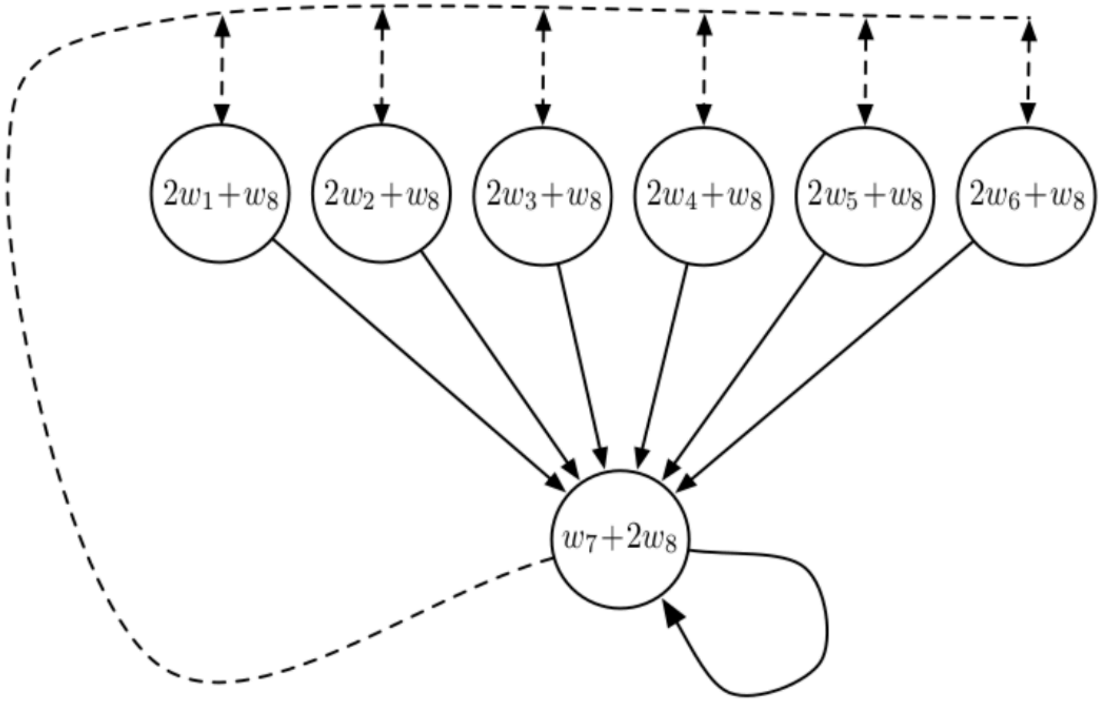


TD(0) Linear VFA for Policy Evaluation Example

Recall MC update: $\alpha(G_t - \mathbf{x}(s)^T\mathbf{w})\mathbf{x}(s)$
Two actions:

- a_1 goes to 7 (solid line)
- a_2 goes to 1-6 with 1/6 probability (dashed line)

Observe $(s_1, a_1, 0, s_7)$
Assume $\mathbf{w}_0 = [1,1,1,1,1,1,1,1], \alpha = 0.5, \gamma$ =0.9
What is $\mathbf{w}_1$ after the first SGD update?



Convergence Rates for Linear Value Function Approximation for Policy Evaluation

- Does TD or MC converge faster to a fixed point?
- Not (to my knowledge) definitively understood
- Practically TD learning often converges faster to its fixed value function approximation point