Welcome to

DS595 Reinforcement Learning

Prof. Yanhua Li

Time: 6:00pm –8:50pm W
Location: Zoom
Spring 2022

Quiz 2 Wed Week #6 (2/23 W)

- Model-Free Policy Evaluation
  - MC First Visit
  - MC Every Visit
  - TD
  - 30 min at the beginning

Quiz 3 Wed in Week 7 (3/2 W)

- Model-free Control（30 mins）
  - Model-free control
    - SARSA
    - Q-Learning

Last Lecture

- Model Free Control
  - Generalized policy iteration
  - Control with Exploration
  - Monte Carlo (MC) Policy Iteration
  - Temporal-Difference (TD) Policy Iteration
    - SARSA
    - Q-Learning
  - Project 2 description.

This Lecture

- Value Function Approximation VFA
  - Introduction
  - VFA for Policy Evaluation
  - VFA for Control

This Lecture

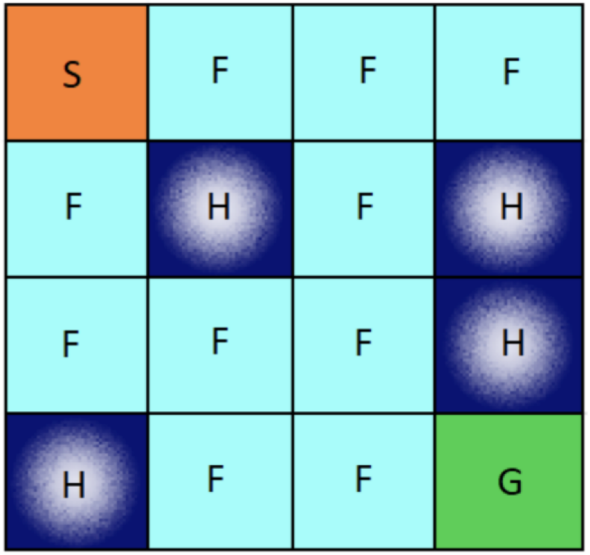
- Value Function Approximation VFA
  - Introduction
  - VFA for Policy Evaluation
  - VFA for Control

RL algorithms

Tabular Representation Value Function		Function Representation Value Function
Model-based control ▪ Policy evaluation (DP) ▪ Policy iteration ▪ Value iteration	Model-Free Control ▪ Policy evaluation MC (First/every visit) and TD ▪ Value/Policy Iteration • MC Iteration • TD Iteration – SARSA – Q-Learning – Double Q-Learning	Policy function approximation Value Function approximation (Asynchronous) Advantage Actor Critic: A2C A3C

Value function representations

- Tabular representation



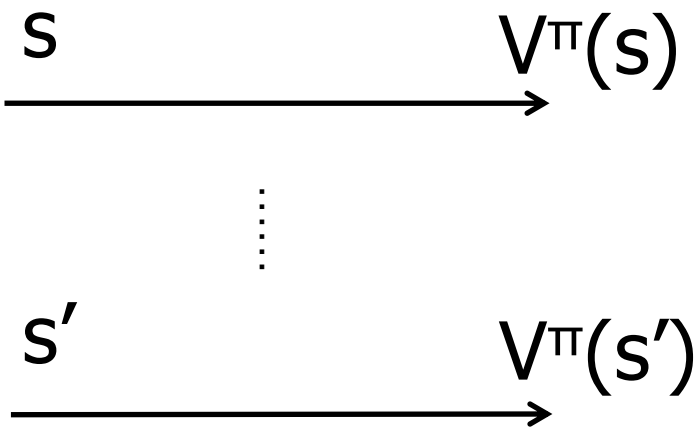
- V^{π} can be viewed as a vector

- Value function

- enormous state and/or action spaces
- Tabular representation is insufficient

Value function representations

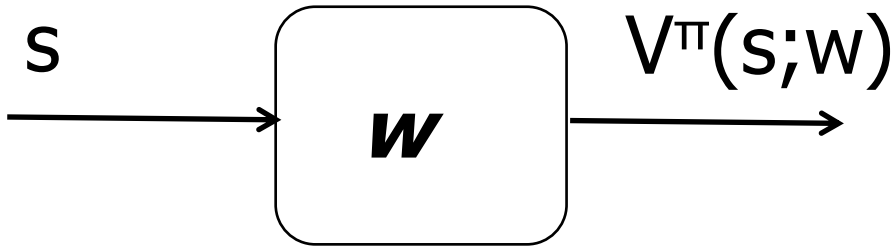
- Tabular representation



$S \rightarrow V^\pi$ *of* $|S|$ *dimensions*

- V^π can be viewed as a vector

- Value function

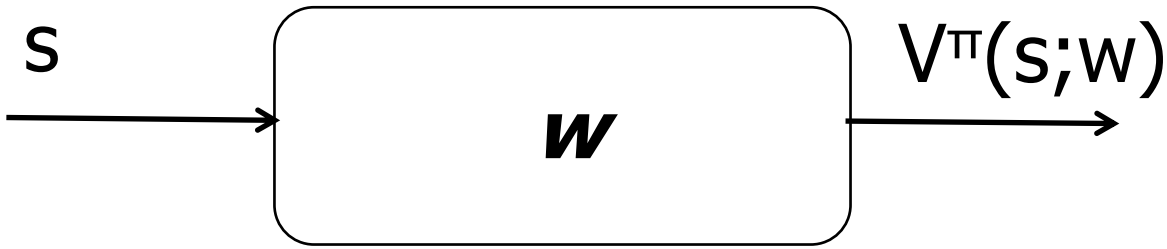


w *with k dimensions,* $k << |S|$

- enormous state and/or action spaces
- Tabular representation is insufficient

Value Function Approximation (VFA)

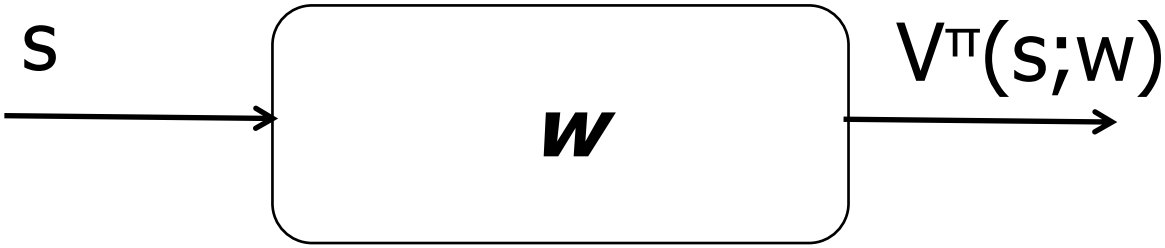
- Represent a (state-action/state) value function with a parameterized function instead of a table
- $V^{\pi}(s)$



- $Q^{\pi}(s,a)$

Why VFA? Benefits of VFA?

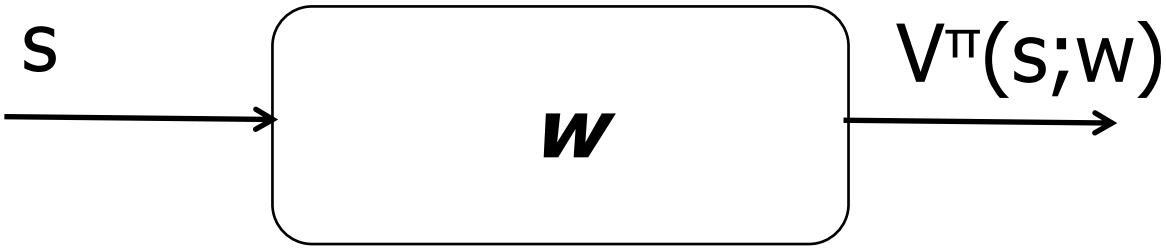
- $V^{\pi}(s)$



- $Q^{\pi}(s,a)$

Why VFA? Benefits of VFA?

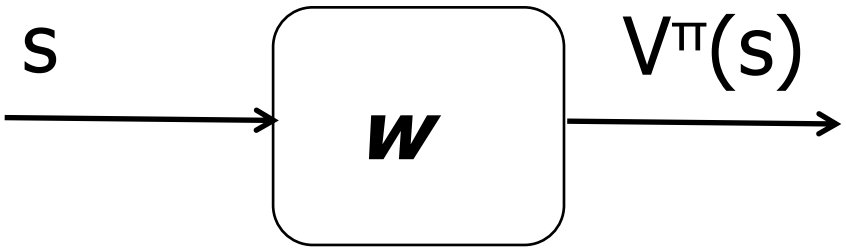
- Huge state and/or action space, thus impossible by tabular representation
- Want more compact representation that generalizes across state or states and actions
- $V^{\pi}(s)$



- $Q^{\pi}(s,a)$

Benefits of Generalization via VFA

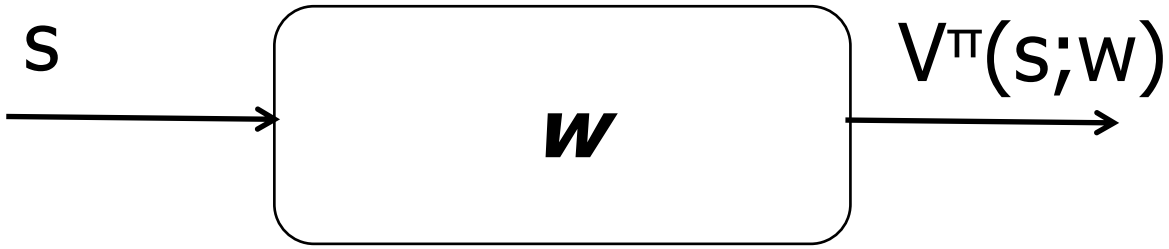
- Huge state and/or action space,
  - Reduce the memory needed
- More compact representation that generalizes across state or states and actions
  - Generalization across states/state-action-pairs
  - Advantages of tabular: Exact value of s, or s,a
- A trade-off
  - Capacity vs (computational and space) efficiency



w *with k dimensions,*
k<<|S|

What function for VFA?

- $V^{\pi}(s)$



- $Q^{\pi}(s,a)$

- What function approximator?

What function for VFA?

- Many possible function approximators including
  - Linear combinations of features
  - Neural networks
  - Decision trees
  - Nearest neighbors, and more.
- In this class we will focus on function approximators that are differentiable (Why?)

- Two very popular classes of differentiable function approximators
  - Linear feature representations;
  - Neural networks (Deep Reinforcement Learning).

This Lecture

- Value Function Approximation VFA
  - Introduction
  - VFA for Policy Evaluation
  - VFA for Control

Review: Gradient Descent

- Consider a function J(**w**) that is a differentiable function of a parameter vector **w**

- Goal is to find parameter **w** that minimizes J
- The gradient of J(w) is

Review: Gradient Descent

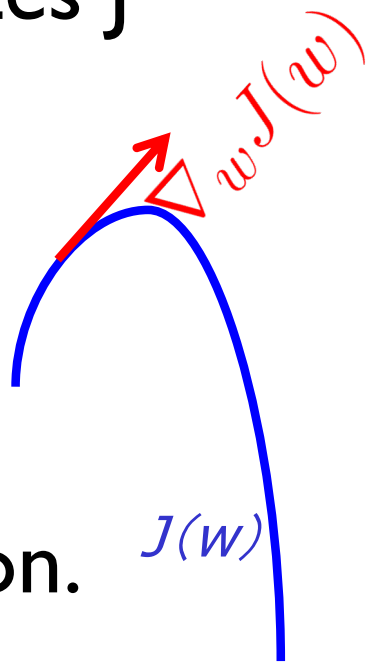
- Consider a function J(**w**) that is a differentiable function of a parameter vector **w**

- Goal is to find parameter **w** that minimizes J
- The gradient of J(w) is

$$\nabla_w J(w) = [\frac{\partial J(w)}{\partial w_1}, \cdots, \frac{\partial J(w)}{\partial w_N}]$$

$$\Delta w = \alpha \nabla_w J(w), \quad \text{with } \alpha > 0$$

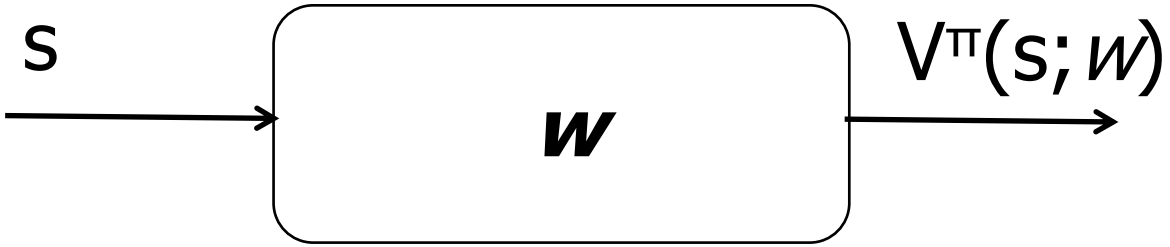
$$w = w - \Delta w$$



- Gradient vector points the uphill direction.
- To minimize J(w), we remove α-weighted gradient vector from w in each iteration.

VFA problem

- Consider an oracle function exists, that takes s as input, and outputs a $V^{\pi}(s)$.
  - The oracle may not be accessible in practice (that is the model-free problem setting).



- The objective is to find the best approximate representation of $V^{\pi}(s)$, given a particular parameterized function $V'^{\pi}(s, \mathbf{w})$

Stochastic Gradient Descent

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

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(V^{\pi}(s) - \hat{V}(s;\boldsymbol{w}))^2]$$

- Can use gradient descent to find a local minimum

$$\Delta \boldsymbol{w} = \frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

Without loss of generality, a constant parameter ½ was added.

- Stochastic gradient descent (SGD) samples the gradient:

- Expected SGD is the same as the full gradient update

Stochastic Gradient Descent

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

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(V^{\pi}(s) - \hat{V}(s;\boldsymbol{w}))^2]$$

- Can use gradient descent to find a local minimum

$$\Delta \boldsymbol{w} = \frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

Without loss of generality, a constant parameter ½ was added.

- Stochastic gradient descent (SGD) samples the gradient:

From full gradient to Stochastic gradient

$$\nabla_w J(w) = -\mathbb{E}_{\pi}[2(V^{\pi}(s) - \hat{V}^{\pi}(s;w))\nabla_w \hat{V}^{\pi}(s;w)]$$

$$\Delta w = -\alpha(V^{\pi}(s) - \hat{V}^{\pi}(s;w))\nabla_w \hat{V}^{\pi}(s;w)$$

- Expected SGD is the same as the full gradient update

$$w = w - \Delta w = w + \alpha(V^{\pi}(s) - \hat{V}^{\pi}(s;w))\nabla_w \hat{V}^{\pi}(s;w)$$

Model-free Policy Evaluation From tabular Representation to VFA

- Following a fixed policy π (or had access to prior data) Goal is to estimate V^{π} and/or Q^{π}

- Maintained a *look up table* to store estimates V^{π} and/or Q^{π}
- Updated *these tabular estimates*
  - after each episode (Monte Carlo methods)

  or
  - after each step (TD methods)

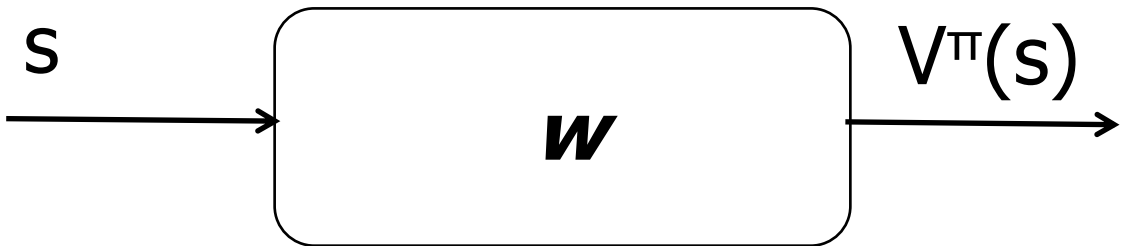
V(1)	V(2)	V(3)	V(4)	V(5)

Model-free Policy Evaluation From tabular Representation to VFA

- Following a fixed policy π (or had access to prior data) Goal is to estimate V^{π} and/or Q^{π}

- Maintained a *function parameter vector* ***w*** to store estimates V^{π} and/or Q^{π}
- Updated *the function parameter vector* ***w***
  - after each episode (Monte Carlo methods)

  or
  - after each step (TD methods)



Feature Vectors

- Use a feature vector to represent a state s

$$\boldsymbol{x}(s) = \begin{pmatrix} x_1(s) \\ x_2(s) \\ \dots \\ x_n(s) \end{pmatrix}$$

Linear Value Function Approximation for Prediction With An Oracle

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

$$\hat{V}(s; \boldsymbol{w}) = \sum_{j=1}^{n} x_j(s) w_j = \boldsymbol{x}(s)^T \boldsymbol{w}$$

- Objective function is

$$J(\boldsymbol{w}) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}(s; \boldsymbol{w}))^2]$$

- Recall weight update is

$$\Delta \boldsymbol{w} = \quad \frac{1}{2} \alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

- Update is:

- Update = step-size $\times$ prediction error $\times$ feature value

Linear Value Function Approximation for Prediction With An Oracle

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

$$\hat{V}(s; \boldsymbol{w}) = \sum_{j=1}^{n} x_j(s) w_j = \boldsymbol{x}(s)^T \boldsymbol{w}$$

- Objective function is

$$J(\boldsymbol{w}) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}(s; \boldsymbol{w}))^2]$$

- Recall weight update is

$$\Delta \boldsymbol{w} = \frac{1}{2} \alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

- Update is:

$$\Delta \mathbf{w} = -\alpha(V^\pi(s) - \hat{V}^\pi(s; \mathbf{w}))\mathbf{x}(s)$$

- Update = step-size $\times$ prediction error $\times$ feature value

- From updating initial V over iterations
  - MC $$V^{\pi}(s) = V^{\pi}(s) + \alpha(G_{i,t} - V^{\pi}(s))$$
  - TD $$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha(\underbrace{[r_t + \gamma V^{\pi}(s_{t+1})]}_{\text{TD target}} - V^{\pi}(s_t))$$

- To update initial w over iterations

$$\Delta \mathbf{w} = -\alpha(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}))\mathbf{x}(s)$$

$$w = w - \Delta w$$

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

$$\begin{aligned} \Delta \boldsymbol{w} &= -\alpha(G_t - \hat{V}(s_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{V}(s_t; \boldsymbol{w}) \\ &= -\alpha(G_t - \hat{V}(s_t; \boldsymbol{w}))\boldsymbol{x}(s_t) \\ &= -\alpha(G_t - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t) \end{aligned}$$

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

$$\hat{V}(s; \boldsymbol{w}) = \sum_{j=1}^{n} x_j(s) w_j = \boldsymbol{x}(s)^T \boldsymbol{w}$$

MC Linear Value Function Approximation for Policy Evaluation

1: Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$
2: **loop**
3: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,L_k})$ given π
4: **for** $t = 1, \ldots, L_k$ **do**
5: **if** First visit to (s) in episode k **then**
6: $G_t(s) = \sum_{j=t}^{L_k} r_{k,j} \ \gamma^{j-t}$
7: Update weights:
8: **end if**
9: **end for**
10: $k = k + 1$
11: **end loop**

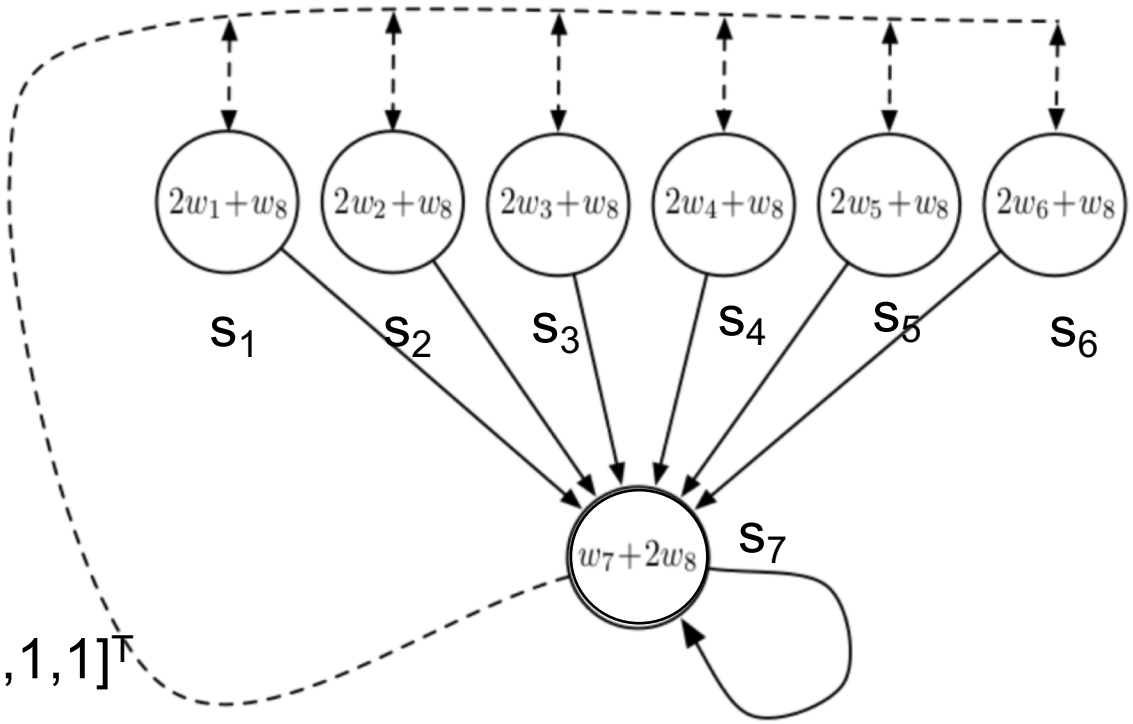
MC Linear Value Function Approximation for Policy Evaluation

1: Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$
2: **loop**
3: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,L_k})$ given π
4: **for** $t = 1, \ldots, L_k$ **do**
5: **if** First visit to (s) in episode k **then**
6: $G_t(s) = \sum_{j=t}^{L_k} r_{k,j} \ \gamma^{j-t}$
7: Update weights:

$$\mathbf{w} = \mathbf{w} \ - \ \Delta\mathbf{w} = \mathbf{w} + \alpha(G_t(s) - \gamma\hat{V}(s;\mathbf{w}))\mathbf{x}(s_t)$$
$$= \mathbf{w} + \alpha(G_t(s) - \gamma\mathbf{x}(s_t)^\top\mathbf{w})\mathbf{x}(s_t)$$

8: **end if**
9: **end for**
10: $k = k + 1$
11: **end loop**

Baird (1995)-Like Example with MC Policy Evaluation[1]



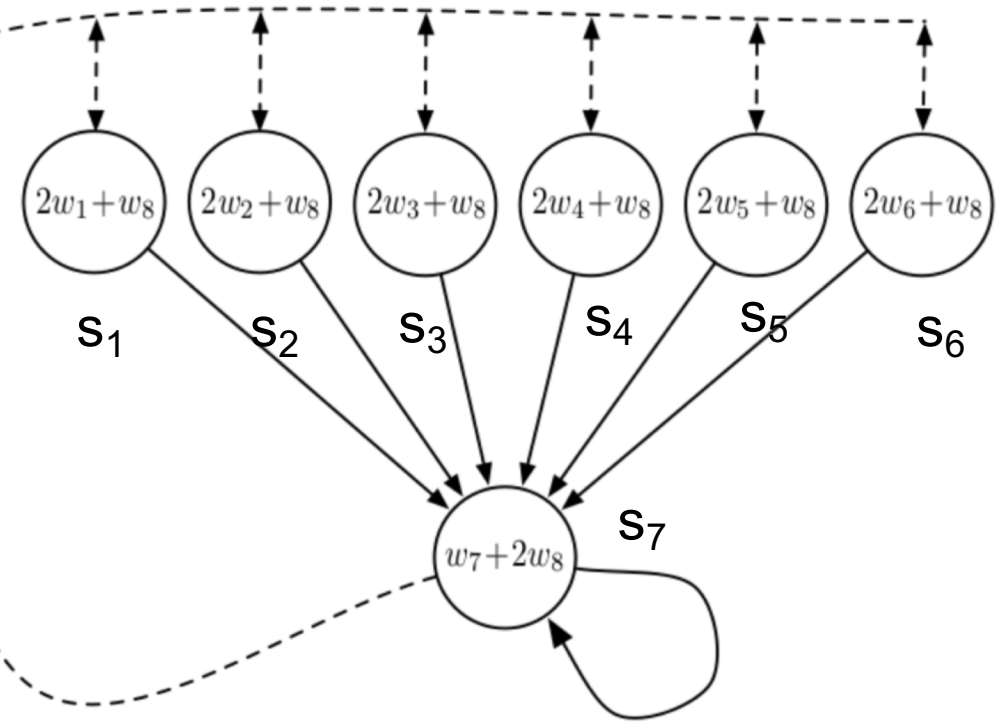
Init w_0=[1,1,1,1,1,1,1,1]^T

- MC update: $\Delta \boldsymbol{w} = \alpha(G_t - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$, α=0.5, γ=1
- Small prob s_7 goes to terminal state, $\boldsymbol{x}(s_7)^T = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 2]$

(s_1,a_1,0,s_7,a_1,0,s_7,a_1,0,T)
What is Δw *and* w_1 after update with the first visit of s_1?

Baird (1995)-Like Example with MC Policy Evaluation[1]

$x(s_1)=[2,0,0,0,0,0,0,1]^T$
$x(s_2)=[0,2,0,0,0,0,0,1]^T$
$x(s_3)=[0,0,2,0,0,0,0,1]^T$
$x(s_4)=[0,0,0,2,0,0,0,1]^T$
$x(s_5)=[0,0,0,0,2,0,0,1]^T$
$x(s_6)=[0,0,0,0,0,2,0,1]^T$
$x(s_7)=[0,0,0,0,0,0,1,2]^T$
$w_0=[1,1,1,1,1,1,1,1]^T$



- MC update: $\Delta \boldsymbol{w} = -\alpha(G_t - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$, α=0.5, γ=1
- Small prob s_7 goes to terminal state, $\boldsymbol{x}(s_7)^T = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 2]$

$(s_1,a_1,0,s_7,a_1,0,s_7,a_1,0,T)$

s_1: $G_{s1}=0$, $V(s_1)=x(s_1)^T w = 3$
α=0.5, $x(s_1)=[2,0,0,0,0,0,0,1]^T$
$\Delta w=-0.5*(0-3)\ [2,0,0,0,0,0,0,1]^T=[3,0,0,0,0,0,0,1.5]^T$
$w_1=w_0-\Delta w=[1,1,1,1,1,1,1,1]^T-[3,0,0,0,0,0,0,1.5]^T=[-2,1,1,1,1,1,1,-0.5]^T$

Tabular representation

- Uses bootstrapping and sampling to approximate V^π
- Updates $V^\pi(s)$ after each transition (s, a, r, s'):

$$V^\pi(s) \quad = \quad V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s))$$

- Target is $r + \gamma V^\pi(s')$, a biased estimate of the true value $V^\pi(s)$
- Represent value for each state with a separate table entry

Input: α
Initialize $V^\pi(s) = 0, \forall s \in S$
Loop

- Sample **tuple** (s_t, a_t, r_t, s_{t+1})
- $V^\pi(s_t) = V^\pi(s_t) + \alpha(\underbrace{[r_t + \gamma V^\pi(s_{t+1})]}_{\text{TD target}} - V^\pi(s_t))$

Temporal Difference (TD(0)) Learning with Value Function Approximation

- In value function approximation, target is $r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w})$, 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:
  - $\langle s_1, r_1 + \gamma \hat{V}^{\pi}(s_2; \boldsymbol{w}) \rangle, \langle s_2, r_2 + \gamma \hat{V}(s_3; \boldsymbol{w}) \rangle, \ldots$
- Find weights to minimize mean squared error

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(r_j + \gamma \hat{V}^{\pi}(s_{j+1}, \boldsymbol{w}) - \hat{V}(s_j; \boldsymbol{w}))^2]$$

Temporal Difference (TD(0)) Learning with Value Function Approximation

- In value function approximation, target is $r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w})$, a biased and approximated estimate of the true value $V^{\pi}(s)$
- Supervised learning on a different set of data pairs: $\langle s_1, r_1 + \gamma \hat{V}^{\pi}(s_2; \boldsymbol{w}) \rangle, \langle s_2, r_2 + \gamma \hat{V}(s_3; \boldsymbol{w}) \rangle, \ldots$
- In linear TD(0)

$$
\begin{aligned}
\Delta \boldsymbol{w} &= - \alpha (r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w}) - \hat{V}^{\pi}(s; \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{V}^{\pi}(s; \boldsymbol{w}) \\
&= - \alpha (r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w}) - \hat{V}^{\pi}(s; \boldsymbol{w})) \boldsymbol{x}(s) \\
&= - \alpha (r + \gamma \boldsymbol{x}(s')^{T} \boldsymbol{w} - \boldsymbol{x}(s)^{T} \boldsymbol{w}) \boldsymbol{x}(s)
\end{aligned}
$$

TD(0) Linear Value Function Approximation 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:

$$s = s_k, s' = s_{k+1}, r = r_k$$
$$\boldsymbol{w} = \boldsymbol{w} + \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$$

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

This Lecture

- Value Function Approximation VFA
  - Introduction
  - VFA for Policy Evaluation
  - VFA for Control

Recall: Tabular representation

1: Initialize $Q(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
2: Set π_b to be ϵ-greedy w.r.t. Q
3: **loop**
4: Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
5: Observe (r_t, s_{t+1})
6: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \quad \max_a Q(s_{t_1}, a) - Q(s_t, a_t))$
7: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$
9: **end loop**

Does how Q is initialized matter?
Asymptotically no, under mild condiditions, but at the beginning, yes

Control using Value Function Approximation

- Use value function approximation to represent state-action values $\hat{Q}^{\pi}(s, a; \boldsymbol{w}) \approx Q^{\pi}$
- Interleave
  - Approximate policy evaluation using value function approximation
  - Perform ϵ-greedy policy improvement
- Can be unstable. Generally involves intersection of the following:
  - Function approximation
  - Bootstrapping

Action-Value Function Approximation with an Oracle

- $\hat{Q}^{\pi}(s, a; \boldsymbol{w}) \approx Q^{\pi}$
- Minimize the mean-squared error between the true action-value function $Q^{\pi}(s, a)$ and the approximate action-value function:

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a; \boldsymbol{w}))^2]$$

- Use stochastic gradient descent to find a local minimum

$$\nabla_w J(w) = \mathbb{E}_{\pi}\left[-2(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a; w))\nabla_w \hat{Q}^{\pi}(s, a; w)\right]$$
$$\Delta w = \frac{1}{2}\alpha \nabla_w J(w) = -\alpha(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a; w))\nabla_w \hat{Q}^{\pi}(s, a; w)$$

- Stochastic gradient descent (SGD) samples the gradient

Linear State Action Value Function Approximation with an Oracle

- Use features to represent both the state and action

$$\boldsymbol{x}(s,a) = \begin{pmatrix} x_1(s,a) \\ x_2(s,a) \\ \cdots \\ x_n(s,a) \end{pmatrix}$$

- Represent state-action value function with a weighted linear combination of features

$$\hat{Q}(s,a;\boldsymbol{w}) = \boldsymbol{x}(s,a)^T \boldsymbol{w} = \sum_{j=1}^{n} x_j(s,a) w_j$$

- Stochastic gradient descent update:

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \mathbb{E}_{\pi}[(Q^{\pi}(s,a) - \hat{Q}^{\pi}(s,a;\boldsymbol{w}))^2]$$

Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- In Monte Carlo methods, use a return G_t as a substitute target

$$\Delta w = -\alpha(G_t - \hat{Q}(s_t, a_t; w))\nabla_w \hat{Q}(s_t, a_t; w)$$

- For SARSA instead use a TD target $r + \gamma \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the current function approximation value

$$\Delta w = -\alpha(r + \gamma \hat{Q}(s', a'; w) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s, a; w)$$

Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- In Monte Carlo methods, use a return G_t as a substitute target

$$\Delta w = -\alpha(G_t - \hat{Q}(s_t, a_t; w))\nabla_w \hat{Q}(s_t, a_t; w)$$

- For SARSA instead use a TD target $r + \gamma \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the current function approximation value

$$\Delta w = -\alpha(r + \gamma \hat{Q}(s', a'; w) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s, a; w)$$

- For Q-learning instead use a TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the max of the current function approximation value

$$\Delta w = -\alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s, a; w)$$

Recall: Tabular representation

1: Initialize $Q(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
2: Set π_b to be ϵ-greedy w.r.t. Q
3: **loop**
4: Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
5: Observe (r_t, s_{t+1})
6: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \arg\max_a Q(s_{t_1}, a) - Q(s_t, a_t))$
7: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$
9: **end loop**

Does how Q is initialized matter?
Asymptotically no, under mild condiditions, but at the beginning, yes

Model-Free Q-Learning Control Value Function Approximation (VFA)

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:

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

Model-Free Q-Learning Control Value Function Approximation (VFA)

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:

$$\Delta w = -\alpha(r_k + \gamma \max_{a_{k+1}} \hat{Q}(s_{k+1}, a_{k+1}; w) - \hat{Q}(s_k, a_k; w)) \nabla_w \hat{Q}(s_k, a_k; w)$$

$$w = w - \Delta w$$

$$\pi(s_k) = \arg\max_{a_k} \hat{Q}(s_k, a_k), \text{with prob } 1 - \epsilon, \text{else random.}$$

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

Convergence of Control Methods with VFA

Algorithm	Tabular	Linear VFA	Nonlinear VFA
Monte-Carlo Control			
Sarsa			
Q-learning			

Convergence of Control Methods with VFA

Algorithm	Tabular	Linear VFA	Nonlinear VFA
Monte-Carlo Control	V	(V)	X
Sarsa	V	(V)	X
Q-learning	V	X	X

See more details in Chapter 11 in Textbook by Sutton & Barto

RL algorithms

Tabular Representation Value Function		Function Representation Value Function
Model-based control	Model-Free Control	Policy function approximation
▪ Policy evaluation (DP)	▪ Policy evaluation MC (First/every visit) and TD	Value Function approximation
▪ Policy iteration ▪ Value iteration	▪ Value/Policy Iteration • MC Iteration • TD Iteration – SARSA – Q-Learning – Double Q-Learning	(Asynchronous) Advantage Actor Critic: A2C A3C

Next Lecture

- (Continue) Value Function Approximation
  - Linear Value Function
- Review of Deep Learning
- Deep Learning Implementation in Pytorch
  - (by TA Yingxue)

Questions?