Lecture 6: Model-free RL with Value Function Approximation Continued [1]

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2023

[1] With some slides based on slides for DQN from David Silver

Class Structure

- Last time: Model-free value function approximation control and Deep Q-learning
- This time: Model-free value function approximation and more DQN
- Next time: Policy search in large spaces / policy gradient methods

Refresh Your Understanding: Modified AB Example: (Ex. 6.4, Sutton & Barto, 2018)

- Two states A, B with $\gamma = 1$
- Given 8 episodes of experience:
  - $A, 1, B, 0$ (observed 2 times)
  - $B, 1$ (observed 4 times)
  - $B, 0$ (observed 2 times)
- Imagine run TD updates over data infinite number of times, and (separately) MC over data an infinite number of times?
- What is $V^{TD}(B)$ and $V^{TD}(A)$? What is $V^{MC}(B)$ and $V^{MC}(A)$?

Refresh Your Understanding: Modified AB Example: (Ex. 6.4, Sutton & Barto, 2018). Solution

- Two states A, B with $\gamma = 1$
- Given 8 episodes of experience:
  - $A, 1, B, 0$ (observed 2 times)
  - $B, 1$ (observed 4 times)
  - $B, 0$ (observed 2 times)
- Imagine run TD updates over data infinite number of times, and (separately) MC over data an infinite number of times?
- What is $V^{TD}(B)$ and $V^{TD}(A)$? What is $V^{MC}(B)$ and $V^{MC}(A)$?

Table of Contents

Table of Contents

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation

- Define the mean squared error of a linear value function approximation for a particular policy π relative to the true value as

$$MSVE_{\mu}(\boldsymbol{w}) = \sum_{s \in S} \mu(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \boldsymbol{w}))^2$$

- where
  - $\mu(s)$: probability of visiting state s under policy π . Note $\sum_s \mu(s) = 1$
  - $\hat{V}^{\pi}(s; \boldsymbol{w}) = \boldsymbol{x}(s)^T \boldsymbol{w}$, a linear value function approximation

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation

- Define the mean squared error of a linear value function approximation for a particular policy π relative to the true value as

$$MSVE_{\mu}(\mathbf{w}) = \sum_{s \in S} \mu(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}))^2$$

- where
  - $\mu(s)$: probability of visiting state s under policy π . Note $\sum_s \mu(s) = 1$
  - $\hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w}$, a linear value function approximation
- Monte Carlo policy evaluation with VFA converges to the weights $\mathbf{w}_{MC}$ which has the minimum mean squared error possible with respect to the distribution μ:

$$MSVE_{\mu}(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_{s \in S} \mu(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}))^2$$

Convergence Guarantees for TD Linear VFA for Policy Evaluation: Preliminaries

- For infinite horizon, the Markov Chain defined by a MDP with a particular policy will eventually converge to a probability distribution over states $d(s)$
- $d(s)$ is called the stationary distribution over states of π
- $\sum_s d(s) = 1$
- $d(s)$ satisfies the following balance equation:

$$d(s') = \sum_s \sum_a \pi(a|s) p(s'|s, a) d(s)$$

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation

- Define the mean squared error of a linear value function approximation for a particular policy π relative to the true value given the distribution d as

$$MSVE_d(\boldsymbol{w}) = \sum_{s \in S} d(s)(V^\pi(s) - \hat{V}^\pi(s;\boldsymbol{w}))^2$$

- where
  - $d(s)$: stationary distribution of π in the true decision process
  - $\hat{V}^\pi(s;\boldsymbol{w}) = \boldsymbol{x}(s)^T\boldsymbol{w}$, a linear value function approximation
- TD(0) policy evaluation with VFA converges to weights $\boldsymbol{w}_{TD}$ which is within a constant factor of the min mean squared error possible given distribution d:

$$MSVE_d(\boldsymbol{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{w}} \sum_{s \in S} d(s)(V^\pi(s) - \hat{V}^\pi(s;\boldsymbol{w}))^2$$

Check Your Understanding L5N1: Poll

- TD(0) policy evaluation with VFA converges to weights $\boldsymbol{w}_{TD}$ which is within a constant factor of the min mean squared error possible for distribution d:

$$MSVE_d(\boldsymbol{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{w}} \sum_{s \in S} d(s)(V^\pi(s) - \hat{V}^\pi(s; \boldsymbol{w}))^2$$

- If the VFA is a tabular representation (one feature for each state), what is the $MSVE_d$ for TD?

1. Depends on the problem
2. MSVE $=$ 0 for TD
3. Not sure

Check Your Understanding L5N1 : Poll

- TD(0) policy evaluation with VFA converges to weights $\boldsymbol{w}_{TD}$ which is within a constant factor of the min mean squared error possible for distribution d:

$$MSVE_d(\boldsymbol{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{w}} \sum_{s \in S} d(s)(V^\pi(s) - \hat{V}^\pi(s;\boldsymbol{w}))^2$$

- If the VFA is a tabular representation (one feature for each state), what is the $MSVE_d$ for TD?

Table of Contents

Recall 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

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

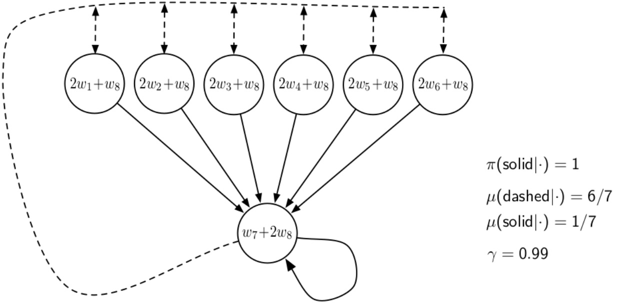
- For SARSA instead use a TD target $r + \gamma \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the current function approximation value

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

- 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

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

Challenges of Off Policy Control: Baird Example [1]



- Behavior policy and target policy are not identical
- Value can diverge

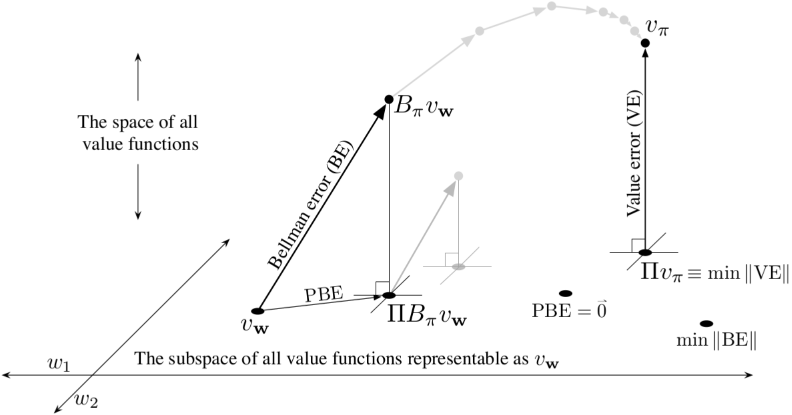
Convergence of Policy Evaluation and Control Methods with VFA

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

Active Area: Off Policy Learning with Function Approximation

- Extensive work in better TD-style algorithms with value function approximation, some with convergence guarantees: see Chp 11 SB
- Will come up further later in this course

Value Function Approximation[1]



[1] Figure from Sutton and Barto 2018

Table of Contents

Table of Contents

Maximization Bias[2]

- Consider single-state MDP ($|S| = 1$) with 2 actions, and both actions have 0-mean **random** rewards, ($\mathbb{E}(r|a = a_1) = \mathbb{E}(r|a = a_2) = 0$).
- Then $Q(s, a_1) = Q(s, a_2) = 0 = V(s)$
- Assume there are prior samples of taking action a_1 and a_2
- Let $\hat{Q}(s, a_1), \hat{Q}(s, a_2)$ be the finite sample estimate of Q
- Use an unbiased estimator for Q: e.g. $\hat{Q}(s, a_1) = \frac{1}{n(s,a_1)} \sum_{i=1}^{n(s,a_1)} r_i(s, a_1)$
- Let $\hat{\pi} = \arg\max_a \hat{Q}(s, a)$ be the greedy policy w.r.t. the estimated $\hat{Q}$

[2] Example from Mannor, Simester, Sun and Tsitsiklis. Bias and Variance Approximation in Value Function Estimates. Management Science 2007

Maximization Bias[3] Proof

- Consider single-state MDP ($|S| = 1$) with 2 actions, and both actions have 0-mean random rewards, ($\mathbb{E}(r|a = a_1) = \mathbb{E}(r|a = a_2) = 0$).
- Then $Q(s, a_1) = Q(s, a_2) = 0 = V(s)$
- Assume there are prior samples of taking action a_1 and a_2
- Let $\hat{Q}(s, a_1), \hat{Q}(s, a_2)$ be the finite sample estimate of Q
- Use an unbiased estimator for Q: e.g. $\hat{Q}(s, a_1) = \frac{1}{n(s,a_1)} \sum_{i=1}^{n(s,a_1)} r_i(s, a_1)$
- Let $\hat{\pi} = \arg\max_a \hat{Q}(s, a)$ be the greedy policy w.r.t. the estimated $\hat{Q}$
- *Even though each estimate of the state-action values is unbiased*, the estimate of $\hat{\pi}$'s value $\hat{V}^{\hat{\pi}}$ can be biased:

[3] Example from Mannor, Simester, Sun and Tsitsiklis. Bias and Variance Approximation in Value Function Estimates. Management Science 2007

Table of Contents

Double Q-Learning

- The greedy policy w.r.t. estimated Q values can yield a maximization bias during finite-sample learning
- Avoid using max of estimates as estimate of max of true values
- Instead split samples and use to create two independent unbiased estimates of $Q_1(s_1, a_i)$ and $Q_2(s_1, a_i)$ $\forall a$.
  - Use one estimate to select max action: $a^* = \arg\max_a Q_1(s_1, a)$
  - Use other estimate to estimate value of a^*: $Q_2(s, a^*)$
  - Yields unbiased estimate: $\mathbb{E}(Q_2(s, a^*)) = Q(s, a^*)$

Double Q-Learning

- The greedy policy w.r.t. estimated Q values can yield a maximization bias during finite-sample learning
- Avoid using max of estimates as estimate of max of true values
- Instead split samples and use to create two independent unbiased estimates of $Q_1(s_1, a_i)$ and $Q_2(s_1, a_i)$ $\forall a$.
  - Use one estimate to select max action: $a^* = \arg\max_a Q_1(s_1, a)$
  - Use other estimate to estimate value of a^*: $Q_2(s, a^*)$
  - Yields unbiased estimate: $\mathbb{E}(Q_2(s, a^*)) = Q(s, a^*)$
- Why does this yield an unbiased estimate of the max state-action value?
- If acting online, can alternate samples used to update Q_1 and Q_2, using the other to select the action chosen
- Next slides extend to full MDP case (with more than 1 state)

Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
loop
    Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t, a) + Q_2(s_t, a)$
    Observe (r_t, s_{t+1})
    if (with 0.5 probability) **then**
        $Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(r_t + \gamma Q_2(s_{t+1}, \arg\max_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t))$
    else
        $Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(r_t + \gamma Q_1(s_{t+1}, \arg\max_a Q_2(s_{t+1}, a)) - Q_2(s_t, a_t))$
    end if
    $t = t + 1$
end loop

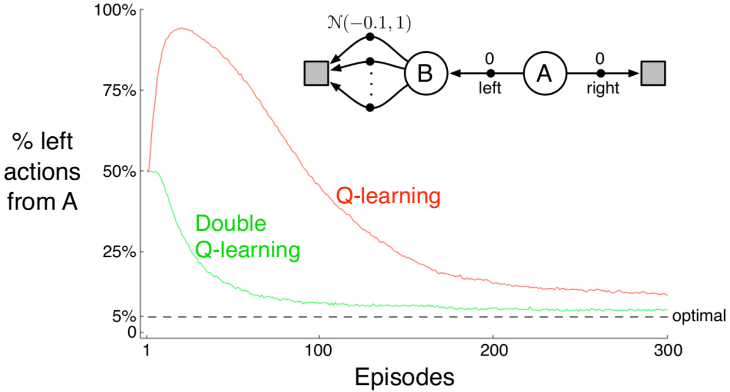
Compared to Q-learning, how does this change the: memory requirements, computation requirements per step, amount of data required?

Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
loop
    Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t, a) + Q_2(s_t, a)$
    Observe (r_t, s_{t+1})
    if (with 0.5 probability) **then**
        $Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(r_t + \gamma Q_2(s_{t+1}, \arg\max_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t))$
    else
        $Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(r_t + \gamma Q_1(s_{t+1}, \arg\max_a Q_2(s_{t+1}, a)) - Q_2(s_t, a_t))$
    end if
    $t = t + 1$
end loop

Compared to Q-learning, how does this change the: memory requirements, computation requirements per step, amount of data required?

Double Q-Learning (Figure 6.7 in Sutton and Barto 2018)



Due to the maximization bias, Q-learning spends much more time selecting suboptimal actions than double Q-learning.

Table of Contents

Recall DQN

- Deep Q-learning (DQN): Q-learning with deep neural networks **and**
  - Experience replay
  - Fixed Q-targets

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

Recall DQN Pseudocode

Input $\boldsymbol{C}$, $\boldsymbol{\alpha}$, $D = \{\}$, Initialize $\boldsymbol{w}$, $\boldsymbol{w}^- = \boldsymbol{w}$, $t = 0$
Get initial state s_0
loop
  Sample action a_t given ϵ-greedy policy for current $\hat{Q}(s_t, a; \boldsymbol{w})$
  Observe reward r_t and next state s_{t+1}
  Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer D
  Sample random minibatch of tuples (s_i, a_i, r_i, s_{i+1}) from D
  for j in minibatch **do**
    if episode terminated at step $i + 1$ **then**
      $y_i = r_i$
    else
      $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \boldsymbol{w}^-)$
    end if
    Do gradient descent step on $(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))^2$ for parameters $\boldsymbol{w}$: $\Delta\boldsymbol{w} = \alpha(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{Q}(s_i, a_i; \boldsymbol{w})$
  end for
  $t = t + 1$
  if mod(t,C) == 0 **then**
    $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$
  end if
end loop

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\mathbf{w}$ is used to select actions
- Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$\Delta \mathbf{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \mathbf{w})}_{\text{Action selection: } \mathbf{w}}; \mathbf{w}^-)}^{\text{Action evaluation: } \mathbf{w}^-} - \hat{Q}(s, a; \mathbf{w}))$$

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}; \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{Q}(s, a; \boldsymbol{w}))$$

- How is this different from fixed target network update used in DQN?

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w}); \boldsymbol{w}^-}_{\text{Action selection: } \boldsymbol{w}})}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{Q}(s, a; \boldsymbol{w}))$$

- How is this different from fixed target network update used in DQN?

Double DQN

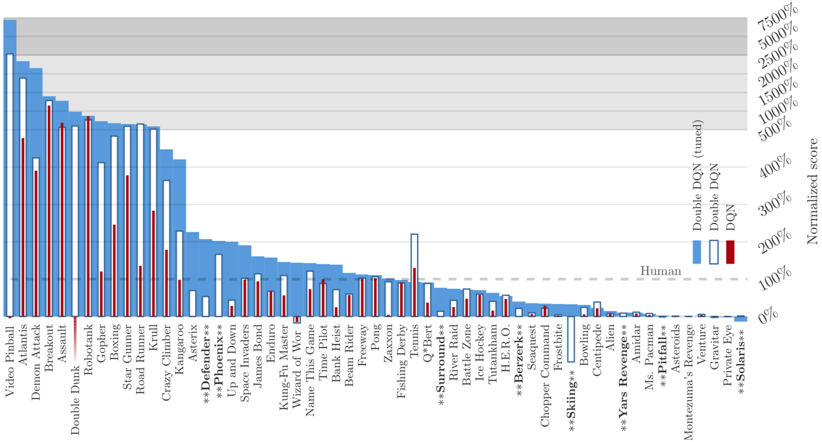


Figure: van Hasselt, Guez, Silver, 2015

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}; \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{Q}(s, a; \boldsymbol{w}))$$

- **Very small code change, often can lead to significantly improved results**

Table of Contents

Rainbow: Combining Improvements in Deep Reinforcement Learning. Hessel et al. 2018 (DeepMind)

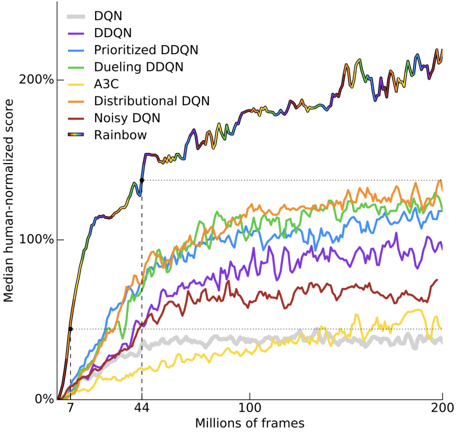


Figure: Median human-normalized performance across 57 Atari games. Curves smoothed with a moving avg over 5 points.

Many new methods

- One (of many) significant ideas: use additional objectives

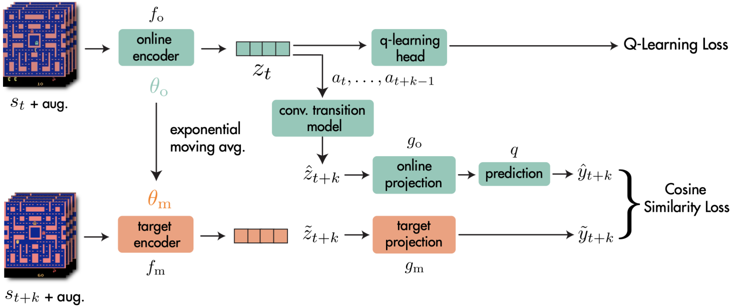


Figure: Data-efficient reinforcement learning with self-predictive representations. Schwarzer et al. ICLR 2021.

What is Enabling Progress?

- Benchmark tasks. Atari, Atari 100k, Mujoco, ...
- Standing on the shoulders of giants... : building on past algorithms
    - and code bases for said algorithms

Model-free value function approximation RL: What You Should Know

- Be able to derive weight update for generic function approximation for Q/V^π
- Understand various (MC/SARSA/Q-learning) targets used when updating Q function
- Know what TD vs MC converge to for policy evaluation with a linear function approximator
- Be able to implement DQN
- Define the maximization bias and give one tool for alleviating it

Class Structure

- Last time: Model-free value function approximation control and Deep Q-learning
- This time: Model-free value function approximation and more DQN
- Next time: Policy search in large spaces / policy gradient methods

Lecture 6: Refresh Your Knowledge

- In TD learning with linear VFA (select all):
  1. $\mathbf{w} = \mathbf{w} + \alpha(r(s_t) + \gamma \mathbf{x}(s_{t+1})^T \mathbf{w} - \mathbf{x}(s_t)^T \mathbf{w})\mathbf{x}(s_t)$
  2. $V(s) = \mathbf{w}(s)\mathbf{x}(s)$
  3. Asymptotic convergence to the true best minimum MSE linear representable $V(s)$ is guaranteed for $\alpha \in (0, 1)$, $\gamma < 1$.
  4. Not sure

Lecture 6: Refresh Your Knowledge **Solutions**

- In TD learning with linear VFA (select all):
  1. $\boldsymbol{w} = \boldsymbol{w} + \alpha(r(s_t) + \gamma \boldsymbol{x}(s_{t+1})^T \boldsymbol{w} - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$
  2. $V(s) = \boldsymbol{w}(s)\boldsymbol{x}(s)$
  3. Asymptotic convergence to the true best minimum MSE linear representable $V(s)$ is guaranteed for $\alpha \in (0, 1)$, $\gamma < 1$.
  4. Not sure