Lecture 4: Model Free Control and Function Approximation

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2023

- Structure and content drawn in part from David Silver's Lecture 5 and Lecture 6. For additional reading please see SB Sections 5.2-5.4, 6.4, 6.5, 6.7

Check Your Understanding L4N1: Model-free Generalized Policy Improvement

- Consider policy iteration
- Repeat:
  - Policy evaluation: compute Q^{π}
  - Policy improvement $\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s, a)$
- Question: is this π_{i+1} deterministic or stochastic?
- Answer: Deterministic, Stochastic, Not Sure
- Now consider evaluating the policy of this new π_{i+1}. Recall in model-free policy evaluation, we estimated V^{π}, using π to generate new trajectories
- Question: Can we compute $Q^{\pi_{i+1}}(s, a)\ \forall s, a$ by using this π_{i+1} to generate new trajectories?
- Answer: True, False, Not Sure

Check Your Understanding L4N1: Model-free Generalized Policy Improvement

- Consider policy iteration
- Repeat:
  - Policy evaluation: compute Q^{π}
  - Policy improvement $\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s, a)$
- Question: is this π_{i+1} deterministic or stochastic?
  Answer: Deterministic

- Now consider evaluating the policy of this new π_{i+1}. Recall in model-free policy evaluation, we estimated V^{π}, using π to generate new trajectories
- Question: Can we compute $Q^{\pi_{i+1}}(s, a)\ \forall s, a$ by using this π_{i+1} to generate new trajectories?
  Answer: No.

Class Structure

- Last time: Policy evaluation with no knowledge of how the world works (MDP model not given)
- Control (making decisions) without a model of how the world works
- Generalization – Value function approximation

Todays Lecture

1. Model-Free Control with a Tabular Representation
   - Generalized Policy Improvement
   - Monte-Carlo Control with Tabular Representations
   - Temporal Difference Methods for Control
2. Value Function Approximation
   - Model Free Value Function Approximation Policy Evaluation
   - Monte Carlo Value Function Approximation Policy Evaluation
   - Temporal Difference (TD(0)) Value Function Approximation Policy Evaluation
   - Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation
3. Control using Value Function Approximation

Table of Contents

Today: Model-free Control

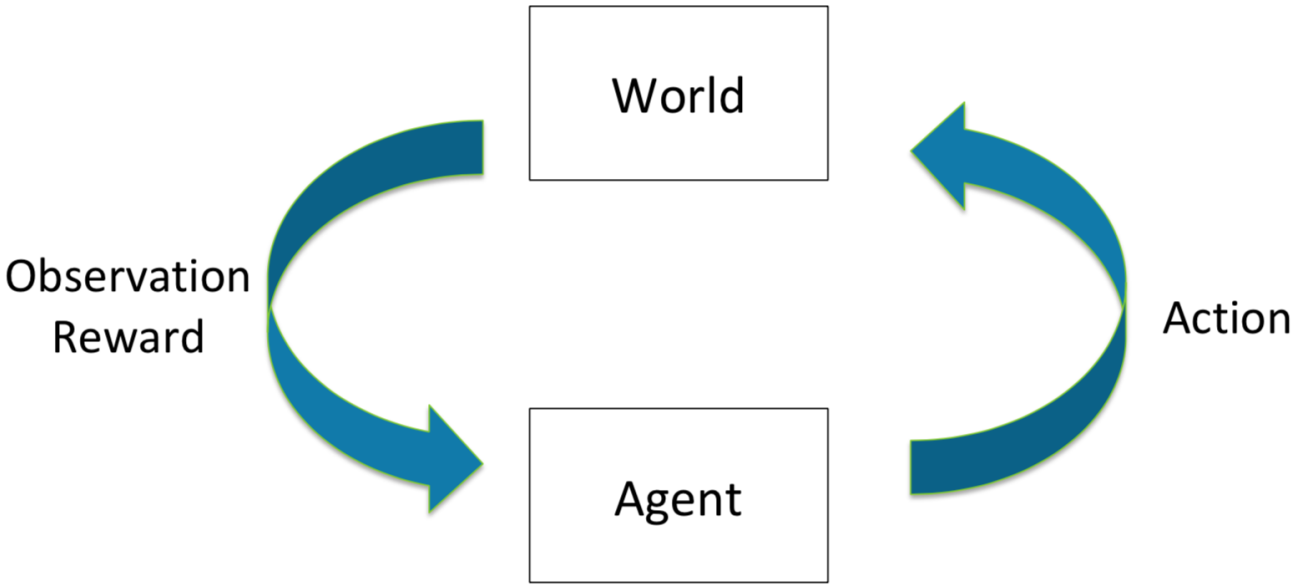
- **Generalized policy improvement**
- **Importance of exploration**
- Monte Carlo control
- Model-free control with temporal difference (SARSA, Q-learning)
- Maximization bias

Table of Contents

Model-free Policy Iteration

- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q^{π}
  - Policy improvement: update π given Q^{π}
- May need to modify policy evaluation:
  - If π is deterministic, can't compute $Q(s, a)$ for any $a \neq \pi(s)$
- How to interleave policy evaluation and improvement?
  - Policy improvement is now using an estimated Q

The Problem of Exploration



- Goal: Learn to select actions to maximize total expected future reward
- Problem: Can't learn about actions without trying them (need to *explore*
- Problem: But if we try new actions, spending less time taking actions that our past experience suggests will yield high reward (need to *exploit* knowledge of domain to achieve high rewards)

ϵ-greedy Policies

- Simple idea to balance exploration and achieving rewards
- Let $|A|$ be the number of actions
- Then an ϵ-greedy policy w.r.t. a state-action value $Q(s, a)$ is $\pi(a|s) =$

ϵ-greedy Policies

- Simple idea to balance exploration and achieving rewards
- Let $|A|$ be the number of actions
- Then an ϵ-greedy policy w.r.t. a state-action value $Q(s, a)$ is $\pi(a|s) =$
  - $\arg\max_a Q(s, a)$, w. prob $1 - \epsilon + \frac{\epsilon}{|A|}$
  - $a' \neq \arg\max Q(s, a)$ w. prob $\frac{\epsilon}{|A|}$

Policy Improvement with ϵ-greedy policies

- Recall we proved that policy iteration using given dynamics and reward models, was guaranteed to monotonically improve
- That proof assumed policy improvement output a deterministic policy
- Same property holds for ϵ-greedy policies

Monotonic ϵ-greedy Policy Improvement

Theorem

For any ϵ-greedy policy π_i, the ϵ-greedy policy w.r.t. Q^{π_i}, π_{i+1} is a monotonic improvement $V^{\pi_{i+1}} \geq V^{\pi_i}$

$$
\begin{aligned}
Q^{\pi_i}(s, \pi_{i+1}(s)) &= \sum_{a \in A} \pi_{i+1}(a|s) Q^{\pi_i}(s, a) \\
&= (\epsilon/|A|) \left[\sum_{a \in A} Q^{\pi_i}(s, a) \right] + (1 - \epsilon) \max_a Q^{\pi_i}(s, a)
\end{aligned}
$$

Today: Model-free Control

- Generalized policy improvement
- Importance of exploration
- **Monte Carlo control**
- Model-free control with temporal difference (SARSA, Q-learning)

Table of Contents

Recall Monte Carlo Policy Evaluation, **Now for Q**

1: Initialize $Q(s,a) = 0, N(s,a) = 0\ \forall (s,a),\ k = 1$, Input $\epsilon = 1,\ \pi$
2: **loop**
3: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,T})$ given π
3: Compute $G_{k,t} = r_{k,t} + \gamma r_{k,t+1} + \gamma^2 r_{k,t+2} + \cdots \gamma^{T_i - 1} r_{k,T_i}\ \forall t$
4: **for** $t = 1, \ldots, T$ **do**
5: **if** First visit to **(s,a)** in episode k **then**
6: $N(s,a) = N(s,a) + 1$
7: $Q(s_t, a_t) = Q(s_t, a_t) + \frac{1}{N(s,a)}(G_{k,t} - Q(s_t, a_t))$
8: **end if**
9: **end for**
10: $k = k + 1$
11: **end loop**

Monte Carlo Online Control / On Policy Improvement

1: Initialize $Q(s,a) = 0, N(s,a) = 0\ \forall (s,a)$, Set $\epsilon = 1$, $k = 1$
2: $\pi_k = \epsilon$-greedy(Q) // Create initial ϵ-greedy policy
3: **loop**
4: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,T})$ given π_k
4: $G_{k,t} = r_{k,t} + \gamma r_{k,t+1} + \gamma^2 r_{k,t+2} + \cdots \gamma^{T_i - 1} r_{k,T_i}$
5: **for** $t = 1, \ldots, T$ **do**
6: **if** First visit to (s,a) in episode k **then**
7: $N(s,a) = N(s,a) + 1$
8: $Q(s_t, a_t) = Q(s_t, a_t) + \frac{1}{N(s,a)}(G_{k,t} - Q(s_t, a_t))$
9: **end if**
10: **end for**
11: $k = k + 1$, $\epsilon = 1/k$
12: $\pi_k = \epsilon$-greedy(Q) // Policy improvement
13: **end loop**

MC for On Policy Control

- Mars rover with new actions:
  - $r(-, a_1) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +10]$, $r(-, a_2) = [\ 0\ 0\ 0\ 0\ 0\ 0\ +5]$, $\gamma = 1$.
- Assume current greedy $\pi(s) = a_1\ \forall s$, ϵ=.5. $Q(s, a) = 0$ for all (s, a)
- Sample trajectory from ϵ-greedy policy
- Trajectory = (s_3, a_1, 0, s_2, a_2, 0, s_3, a_1, 0, s_2, a_2, 0, s_1, a_1, 1, terminal)
- First visit MC estimate of Q of each (s, a) pair?
- $Q^{\epsilon-\pi}(-, a_1) = [1\ 0\ 1\ 0\ 0\ 0\ 0]$

After this trajectory (Select all)

- $Q^{\epsilon-\pi}(-, a_2) = [0\ 0\ 0\ 0\ 0\ 0\ 0]$
- The new **greedy** policy would be: $\pi = [1$ tie 1 tie tie tie tie$]$
- The new **greedy** policy would be: $\pi = [1\ 2\ 1$ tie tie tie tie$]$
- If $\epsilon = 1/3$, prob of selecting a_1 in s_1 in the new ϵ-greedy policy is $1/9$.
- If $\epsilon = 1/3$, prob of selecting a_1 in s_1 in the new ϵ-greedy policy is $2/3$.
- If $\epsilon = 1/3$, prob of selecting a_1 in s_1 in the new ϵ-greedy policy is $5/6$.
- Not sure

Properties of MC control with ϵ-greedy policies

- Computational complexity?
- Converge to optimal Q^* function?
- Empirical performance?

L4N2 Check Your Understanding: 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

- Is Q and estimate of Q^{π_k}? When might this procedure fail to compute the optimal Q^*?

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

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: $\epsilon_i = 1/i$

GLIE Monte-Carlo Control

Theorem

GLIE Monte-Carlo control converges to the optimal state-action value function $Q(s, a) \rightarrow Q^*(s, a)$

Table of Contents

Model-free Policy Iteration with TD Methods

- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q^{π} using temporal difference updating with ϵ-greedy policy
  - Policy improvement: Same as Monte carlo policy improvement, set π to ϵ-greedy (Q^{π})
- First consider SARSA, which is an on-policy algorithm.
- On policy: SARSA is trying to compute an estimate Q of the policy being followed.

General Form of SARSA Algorithm

1: Set initial ϵ-greedy policy π randomly, $t = 0$, initial state $s_t = s_0$
2: Take $a_t \sim \pi(s_t)$
3: Observe (r_t, s_{t+1})
4: **loop**
5: Take action $a_{t+1} \sim \pi(s_{t+1})$ // Sample action from policy
6: Observe (r_{t+1}, s_{t+2})
7: Update Q given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$:
8: Perform policy improvement:
9: $t = t + 1$
10: **end loop**

General Form of SARSA Algorithm

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

Worked Example: SARSA for Mars Rover

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

- Initialize $\epsilon = 1/k$, $k = 1$, and $\alpha = 0.5$, $Q(-, a_1) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +10]$, $Q(-, a_2) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +5]$, $\gamma = 1$
- Assume starting state is s_6 and sample a_1

Worked Example: SARSA for Mars Rover

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

- Initialize $\epsilon = 1/k$, $k = 1$, and $\alpha = 0.5$, $Q(-, a_1) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +10]$, $Q(-, a_2) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +5]$, $\gamma = 1$
- Tuple: $(s_6, a_1, 0, s_7, a_2, 5, s_7)$.
- $Q(s_6, a_1) = .5 * 0 + .5 * (0 + \gamma Q(s_7, a_2)) = 2.5$

Properties of SARSA with ϵ-greedy policies

- Computational complexity?
- Converge to optimal Q^* function? Recall:
  - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
  - $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
  - *Q is an estimate of the performance of a policy that may be changing at each time step*
- Empirical performance?

Convergence Properties of SARSA

Theorem

SARSA for finite-state and finite-action MDPs converges to the optimal action-value, $Q(s,a) \rightarrow Q^*(s,a)$, under the following conditions:

1. The policy sequence $\pi_t(a|s)$ satisfies the condition of GLIE
2. The step-sizes α_t satisfy the Robbins-Munro sequence such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$
$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

- For ex. $\alpha_t = \frac{1}{t}$ satisfies the above condition.

Properties of SARSA with ϵ-greedy policies

- Result builds on stochastic approximation
- Relies on step sizes decreasing at the right rate
- Relies on Bellman backup contraction property
- Relies on bounded rewards and value function

Stochastic Approximation

On and Off-Policy Learning

- On-policy learning
  - Direct experience
  - Learn to estimate and evaluate a policy from experience obtained from following that policy
- Off-policy learning
  - Learn to estimate and evaluate a policy using experience gathered from following a different policy

Q-Learning: Learning the Optimal State-Action Value

- SARSA is an **on-policy** learning algorithm
- SARSA estimates the value of the current **behavior** policy (policy using to take actions in the world)
- And then updates that (behavior) policy
- Alternatively, can we directly estimate the value of π^* while acting with another behavior policy π_b?
- Yes! Q-learning, an **off-policy** RL algorithm

Q-Learning: Learning the Optimal State-Action Value

- SARSA is an **on-policy** learning algorithm
- SARSA estimates the value of the current **behavior** policy (policy using to take actions in the world)
- And then updates the policy trying to estimate
- Alternatively, can we directly estimate the value of π^* while acting with another behavior policy π_b?
- Yes! Q-learning, an **off-policy** RL algorithm
- Key idea: Maintain state-action Q estimates and use to bootstrap– use the value of the best future action
- Recall SARSA

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$$

- Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

Q-Learning with ϵ-greedy Exploration

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: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
7: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$
9: **end loop**

Worked Example: ϵ-greedy Q-Learning Mars

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: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
7: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$
9: **end loop**

- Initialize $\epsilon = 1/k$, $k = 1$, and $\alpha = 0.5$, $Q(-, a_1) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +10]$, $Q(-, a_2) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +5]$, $\gamma = 1$
- Like in SARSA example, start in s_6 and take a_1.

Worked Example: ϵ-greedy Q-Learning Mars

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: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
7: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$
9: **end loop**

- Initialize $\epsilon = 1/k$, $k = 1$, and $\alpha = 0.5$, $Q(-, a_1) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +10]$, $Q(-, a_2) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +5]$, $\gamma = 1$
- Tuple: $(s_6, a_1, 0, s_7)$.
- $Q(s_6, a_1) = 0 + .5 * (0 + \gamma \max_{a'} Q(s_7, a') - 0) = .5*10 = 5$
- Recall that in the SARSA update we saw $Q(s_6, a_1) = 2.5$ because we used the actual action taken at s_7 instead of the max
- Does how Q is initialized matter (initially? asymptotically?)? Asymptotically no, under mild condiditions, but at the beginning, yes

Check Your Understanding L4N3: SARSA and Q-Learning

- SARSA: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
- Q-Learning:
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$

Select all that are true

1. Both SARSA and Q-learning may update their policy after every step
2. If $\epsilon = 0$ for all time steps, and Q is initialized randomly, a SARSA Q state update will be the same as a Q-learning Q state update
3. Not sure

Check Your Understanding L4N3: SARSA and Q-Learning

- SARSA: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
- Q-Learning:
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$

Select all that are true

1. Both SARSA and Q-learning may update their policy after every step
2. If $\epsilon = 0$ for all time steps, and Q is initialized randomly, a SARSA Q state update will be the same as a Q-learning Q state update
3. Not sure

Both are true.

Q-Learning with ϵ-greedy Exploration

- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal Q^*?
  Visit all (s, a) pairs infinitely often, and the step-sizes α_t satisfy the Robbins-Munro sequence. Note: the algorithm does not have to be greedy in the limit of infinite exploration (GLIE) to satisfy this (could keep ϵ large).

- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal π^*?
  The algorithm is GLIE, along with the above requirement to ensure the Q value estimates converge to the optimal Q.

Table of Contents

Motivation for Function Approximation

- Don't want to have to explicitly store or learn for every single state a
  - Dynamics or reward model
  - Value
  - State-action value
  - Policy
- Want more compact representation that generalizes across state or states and actions

Benefits of Function Approximation

- Reduce memory needed to store $(P, R)/V/Q/\pi$
- Reduce computation needed to compute $(P, R)/V/Q/\pi$
- Reduce experience needed to find a good $P, R/V/Q/\pi$

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 (Today)
  - Neural networks (Next lecture)

Review: Gradient Descent

- Consider a function $J(\boldsymbol{w})$ that is a differentiable function of a parameter vector $\boldsymbol{w}$
- Goal is to find parameter $\boldsymbol{w}$ that minimizes J
- The gradient of $J(\boldsymbol{w})$ is

Value Function Approximation for Policy Evaluation with an Oracle

- First assume we could query any state s and an oracle would return the true value for $V^{\pi}(s)$
- Similar to supervised learning: assume given $(s, V^{\pi}(s))$ pairs
- The objective is to find the best approximate representation of V^{π} given a particular parameterized function $\hat{V}(s; w)$

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

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

- Stochastic gradient descent (SGD) uses a finite number of (often one) samples to compute an approximate 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

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

- Stochastic gradient descent (SGD) uses a finite number of (often one) samples to compute an approximate gradient:

$$\begin{aligned} \Delta_w J(w) &= \Delta_w E_{\pi}[V^{\pi}(s) - \hat{V}(s; w)]^2 \\ &= E_{\pi}[2(V^{\pi}(s) - \hat{V}(s; w)]\Delta_w \hat{V}(s, w) \end{aligned}$$

- Expected SGD is the same as the full gradient update

Table of Contents

Model Free VFA Policy Evaluation

- No oracle to tell true $V^{\pi}(s)$ for any state s
- Use model-free value function approximation

Model Free VFA Prediction / Policy Evaluation

- Recall model-free policy evaluation (Lecture 3)
  - Following a fixed policy π (or had access to prior data)
  - Goal is to estimate V^π and/or Q^π
- Maintained a lookup table to store estimates V^π and/or Q^π
- Updated these estimates after each episode (Monte Carlo methods) or after each step (TD methods)
- **Now: in value function approximation, change the estimate update step to include fitting the function approximator**

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

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

- Update is:

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: $\Delta \boldsymbol{w} = -\frac{1}{2}\alpha(V^\pi(s) - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}$
- Update = step-size $\times$ prediction error $\times$ feature value

Linear Value Function Approximation for Policy Evaluation: Plug In Estimate for $V^{\pi}(s)$

- 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: $\Delta \boldsymbol{w} = -\frac{1}{2} \alpha (V^{\pi}(s) - \boldsymbol{x}(s)^T \boldsymbol{w}) \boldsymbol{x}$

Table of Contents

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

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

MC Linear Value Function Approximation for Policy Evaluation

Initialize w $= 0$, $k = 1$
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

Break

Linear Value Function Approximation for Policy Evaluation: Plug In Estimate for $V^{\pi}(s)$

- 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: $\Delta \boldsymbol{w} = -\frac{1}{2} \alpha (V^{\pi}(s) - \boldsymbol{x}(s)^T \boldsymbol{w}) \boldsymbol{x}$

Table of Contents

Recall: Temporal Difference Learning w/ Lookup Table

- 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

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

- Uses bootstrapping and sampling to approximate true V^π
- Updates estimate $V^\pi(s)$ 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 $V^\pi(s)$
- In value function approximation, target is $r + \gamma \hat{V}^\pi(s'; \boldsymbol{w})$, a biased and approximated estimate of the true value $V^\pi(s)$
- 3 forms of approximation:
  1. Sampling
  2. Bootstrapping
  3. Value function approximation

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)

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

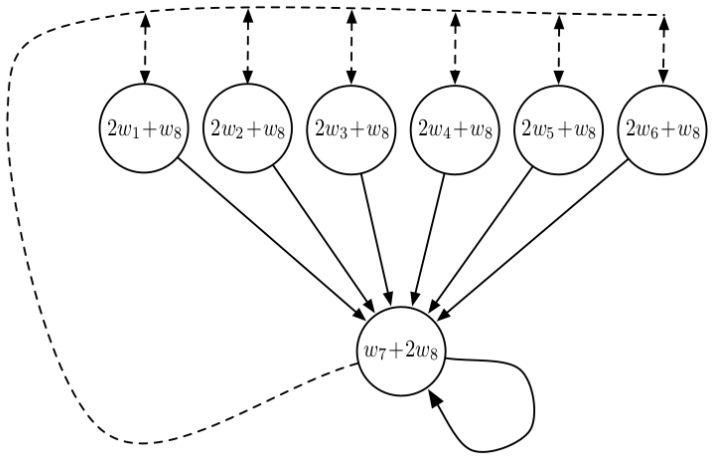
TD(0) Linear Value Function Approximation for Policy Evaluation

1: Initialize w $= 0$, $k = 1$
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**

Baird Example with TD(0) On Policy Evaluation [1]



- $\boldsymbol{x}(s_1) = [2\ 0\ 0\ 0\ 0\ 0\ 0\ 1]$ $\boldsymbol{x}(s_2) = [0\ 2\ 0\ 0\ 0\ 0\ 0\ 1] \ldots \boldsymbol{x}(s_6) = [0\ 0\ 0\ 0\ 0\ 2\ 0\ 1]$ $\boldsymbol{x}(s_7) = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 2]$ $r(s) = 0\ \forall s$ 2 actions a_1 solid line, a_2 dotted
- Small prob s_7 goes to terminal state s_T
- Consider tuple $(s_1, a_1, 0, s_7)$.
- Let w_0 =[1 1 1 1 1 1 1 1]. TD update: $\Delta \boldsymbol{w} = \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$
- TD target is $r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w}$. $r = 0$ $\boldsymbol{x}(s')^T \boldsymbol{w} = 3$.
- $\boldsymbol{x}(s)^T \boldsymbol{w} = 3$
- $\Delta \boldsymbol{w} = \alpha(3\gamma - 3)\boldsymbol{x}(s_1)$

[1] Figure from Sutton and Barto 2018

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

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

- 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
- Monte Carlo policy evaluation with VFA converges to the weights $\boldsymbol{w}_{MC}$ which has the minimum mean squared error possible with respect to the distribution μ:

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

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

- 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

- 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?

MSVE $= 0$ for TD

Table of Contents

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
  - **Off-policy learning**

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

$$\begin{aligned} -\frac{1}{2}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) &= \mathbb{E}\left[(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}^{\pi}(s, a; \boldsymbol{w})\right] \\ \Delta(\boldsymbol{w}) &= -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \end{aligned}$$

- Stochastic gradient descent (SGD) samples the gradient

Check Your Understanding L5N2: Predict Control Updates

- The weight update for control for MC and TD-style methods will be near identical to the policy evaluation steps. Try to see if you can predict which are the right weight update equations for the different methods (select all that are true)
- (1) is the SARSA control update
- (2) is the MC control update
- (3) is the Q-learning control update
- (4) is the MC control update
- (5) is the Q-learning control update

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) (1)$$

$$\Delta \boldsymbol{w} = \alpha(G_t + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) (2)$$

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) (3)$$

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}) (4)$$

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{s'} \hat{Q}(s', a; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) (5)$$

Check Your Understanding L5N2: Answers

- The weight update for control for MC and TD-style methods will be near identical to the policy evaluation steps. Try to see if you can predict which are the right weight update equations for the different methods.
- (1) is the SARSA control update
  $\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$
- (3) is the Q-learning control update
  $\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})(3)$
- (4) is the MC control update
  $\Delta \boldsymbol{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$

Linear State Action Value Function Approximation with an Oracle

- Use features to represent both the state and action

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

- 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

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

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

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

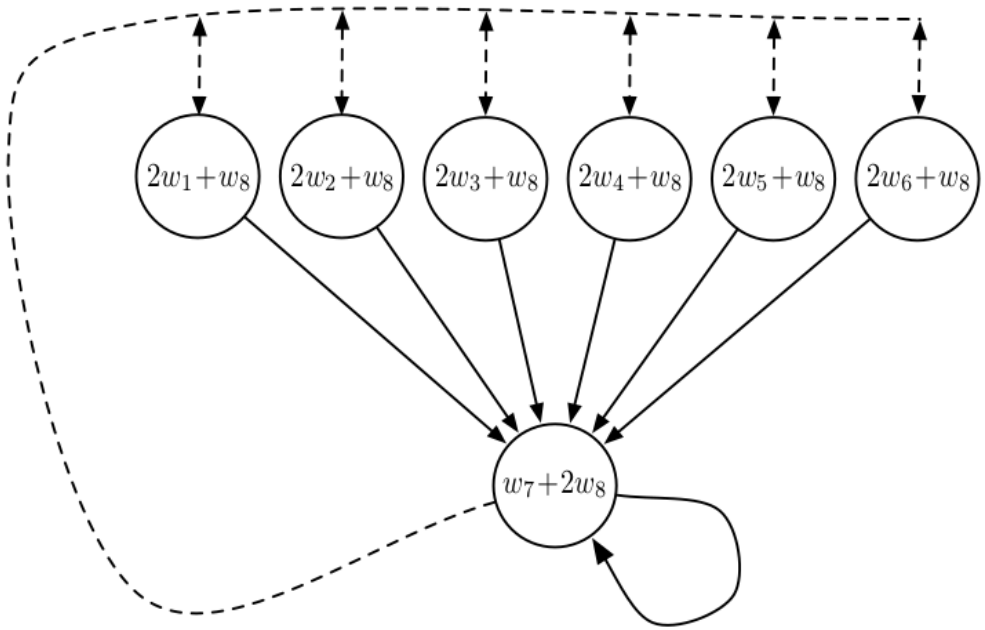
- For Q-learning instead use a TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the max of the current function approximation value

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

Convergence of TD Methods with VFA

- Informally, updates involve doing an (approximate) Bellman backup followed by best trying to fit underlying value function to a particular feature representation
- Bellman operators are contractions, but value function approximation fitting can be an expansion

Challenges of Off Policy Control: Baird Example [1]



$\pi(\text{solid}|\cdot) = 1$

$\mu(\text{dashed}|\cdot) = 6/7$

$\mu(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$

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

What You Should Understand

- Be able to implement TD(0) and MC on policy evaluation with linear value function approximation
- Be able to define what TD(0) and MC on policy evaluation with linear VFA are converging to and when this solution has 0 error and non-zero error.
- Be able to implement Q-learning and SARSA and MC control algorithms
- List the 3 issues that can cause instability and describe the problems qualitatively: function approximation, bootstrapping and off policy learning

Class Structure

- Last time: Control (making decisions) without a model of how the world works
- This time: Value function approximation
- **Next time**: Deep reinforcement learning

Batch Monte Carlo Value Function Approximation

- May have a set of episodes from a policy π
- Can analytically solve for the best linear approximation that minimizes mean squared error on this data set
- Let $G(s_i)$ be an unbiased sample of the true expected return $V^{\pi}(s_i)$

$$\arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (G(s_i) - \boldsymbol{x}(s_i)^T \boldsymbol{w})^2$$

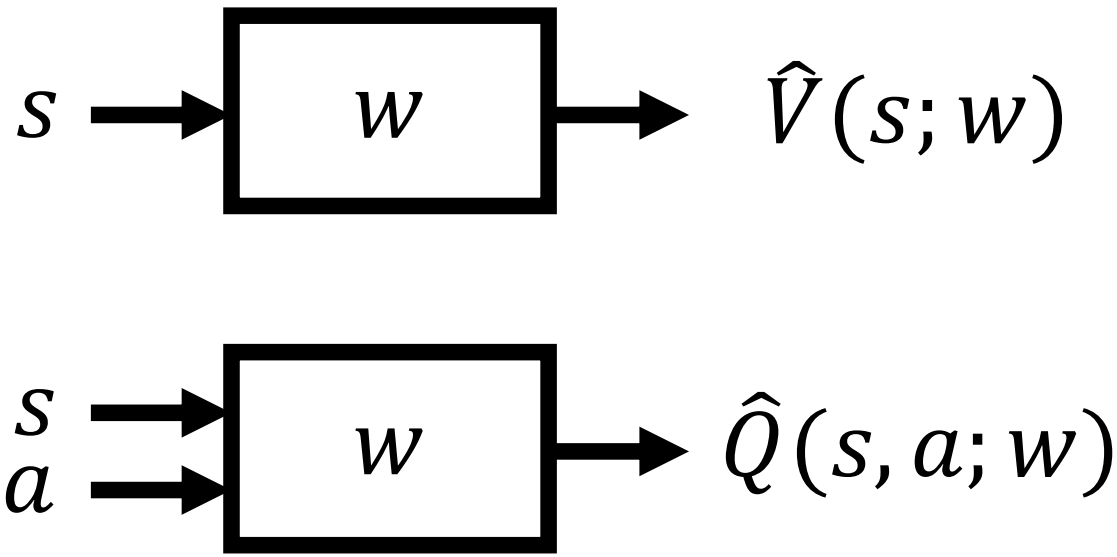
- Take the derivative and set to 0

$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{G}$$

- where $\boldsymbol{G}$ is a vector of all N returns, and X is a matrix of the features of each of the N states $\boldsymbol{x}(s_i)$
- Note: not making any Markov assumptions

Value Function Approximation (VFA)

- Represent a (state-action/state) value function with a parameterized function instead of a table



What You Should Know

- Be able to implement MC on policy control and SARSA and Q-learning
- Compare them according to properties of how quickly they update, (informally) bias and variance, computational cost
- Define conditions for these algorithms to converge to the optimal Q and optimal π and give at least one way to guarantee such conditions are met.

Class Structure

- Last time: Policy evaluation with no knowledge of how the world works (MDP model not given)
- This time: Control (making decisions) without a model of how the world works
- **Next time: Generalization – Value function approximation**

Refresh Your Knowledge L4. Polleverywhere Poll

- Which of the following equations express a TD update?
  1. $V(s_t) = r(s_t, a_t) + \gamma \sum_{s'} p(s'|s_t, a_t) V(s')$
  2. $V(s_t) = (1 - \alpha) V(s_t) + \alpha (r(s_t, a_t) + \gamma V(s_{t+1}))$
  3. $V(s_t) = (1 - \alpha) V(s_t) + \alpha \sum_{i=t}^{H} r(s_i, a_i)$
  4. $V(s_t) = (1 - \alpha) V(s_t) + \alpha \max_a (r(s_t, a) + \gamma V(s_{t+1}))$
  5. Not sure
- Bootstrapping is
  1. When samples of (s,a,s') transitions are used to approximate the true expectation over next states
  2. When an estimate of the next state value is used instead of the true next state value
  3. Used in Monte-Carlo policy evaluation
  4. Not sure

Refresh Your Knowledge L4. Polleverywhere Poll

- Which of the following equations express a TD update?
  True. $V(s_t) = (1 - \alpha)V(s_t) + \alpha(r(s_t, a_t) + \gamma V(s_{t+1}))$

- Bootstrapping is when:
  An estimate of the next state value is used instead of the true next state value

Monotonic ϵ-greedy Policy Improvement

Theorem

For any ϵ-greedy policy π_i, the ϵ-greedy policy w.r.t. Q^{π_i}, π_{i+1} is a monotonic improvement $V^{\pi_{i+1}} \geq V^{\pi_i}$

$$
\begin{aligned}
Q^{\pi_i}(s, \pi_{i+1}(s)) &= \sum_{a \in A} \pi_{i+1}(a|s) Q^{\pi_i}(s, a) \\
&= (\epsilon/|A|) \left[\sum_{a \in A} Q^{\pi_i}(s, a) \right] + (1 - \epsilon) \max_a Q^{\pi_i}(s, a) \\
&= (\epsilon/|A|) \left[\sum_{a \in A} Q^{\pi_i}(s, a) \right] + (1 - \epsilon) \max_a Q^{\pi_i}(s, a) \frac{1 - \epsilon}{1 - \epsilon} \\
&= (\epsilon/|A|) \left[\sum_{a \in A} Q^{\pi_i}(s, a) \right] + (1 - \epsilon) \max_a Q^{\pi_i}(s, a) \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1 - \epsilon} \\
&\geq \frac{\epsilon}{|A|} \left[\sum_{a \in A} Q^{\pi_i}(s, a) \right] + (1 - \epsilon) \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1 - \epsilon} Q^{\pi_i}(s, a) \\
&= \sum_{a \in A} \pi_i(a|s) Q^{\pi_i}(s, a) = V^{\pi_i}(s)
\end{aligned}
$$

Check Your Understanding L4N2: MC for On Policy Control

- Mars rover with new actions:
  - $r(-, a_1) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +10]$, $r(-, a_2) = [\ 0\ 0\ 0\ 0\ 0\ 0\ +5]$, $\gamma = 1$.
- Assume current greedy $\pi(s) = a_1\ \forall s$, ϵ=.5
- Sample trajectory from ϵ-greedy policy
- Trajectory = (s_3, a_1, 0, s_2, a_2, 0, s_3, a_1, 0, s_2, a_2, 0, s_1, a_1, 1, terminal)
- First visit MC estimate of Q of each (s, a) pair?
- $Q^{\epsilon-\pi}(-, a_1) = [1\ 0\ 1\ 0\ 0\ 0\ 0]$, $Q^{\epsilon-\pi}(-, a_2) = [0\ 1\ 0\ 0\ 0\ 0\ 0]$
- What is $\pi(s) = \arg\max_a Q^{\epsilon-\pi}(s, a)\ \forall s$?
  $\pi = [1\ 2\ 1$ tie tie tie tie$]$

- Under the new ϵ-greedy policy, if $k = 3$, $\epsilon = 1/k$
  With probability $2/3$ choose $\pi(s)$ else choose randomly. As an example, $\pi(s_1) = a_1$ with prob $(2/3)$ else randomly choose an action. So the prob of picking a_1 will be $2/3 + (1/3) * (1/2) = 5/6$

SARSA Initialization

- Mars rover with new actions:
  - $r(-, a_1) = [\ 1\ 0\ 0\ 0\ 0\ 0\ +10]$, $r(-, a_2) = [\ 0\ 0\ 0\ 0\ 0\ 0\ +5]$, $\gamma = 1$.
- Initialize $\epsilon = 1/k$, $k = 1$, and $\alpha = 0.5$, $Q(-, a_1) = r(-, a_1)$, $Q(-, a_2) = r(-, a_2)$
- SARSA: $(s_6, a_1, 0, s_7, a_2, 5, s_7)$.
- Does how Q is initialized matter (initially? asymptotically?)? Asymptotically no, under mild condiditions, but at the beginning, yes