Lecture 4: Model Free Control and Function Approximation

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2024

- 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? Assume for each state s there is a unique $\max_a Q^{\pi_i}(s, a)$.
- 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

  $\pi_{i+1}(s)$
- 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? Assume for each state s there is a unique $\max_a Q^{\pi_i}(s, a)$.
  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

Q-learning w/ DNN $\longrightarrow$ DQN

Today's Lecture

- Generalized Policy Improvement
- Monte-Carlo Control with Tabular Representations
- Greedy in the Limit of Infinite Exploration
- Temporal Difference Methods for Control

1. Model Free Value Function Approximation
    - Policy Evaluation
    - Monte Carlo Policy Evaluation
    - Temporal Difference TD(0) Policy Evaluation

2. Control using Value Function Approximation
    - Control using General Value Function Approximators
    - Deep Q-Learning

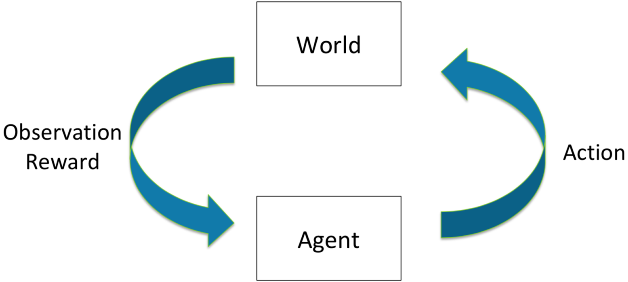
Table of Contents

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

because we will be estimating Q from data

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 $\pi(a|s) =$
  - $\arg\max_a Q(s,a)$, w. prob $1 - \epsilon + \frac{\epsilon}{|A|}$
  - $a' \neq \arg\max Q(s,a)$ w. prob $\frac{\epsilon}{|A|}$
- In words: select argmax action with probability $1-\epsilon$, else select action uniformly at random

$1-\epsilon$ greedy

ϵ randomly

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**

```
Initialize Q(s,a) = 0, N(s,a) = 0 ∀(s,a), k = 1, Input ε = 1, π
loop
    Sample k-th episode (s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, ..., s_{k,T}) given π
    Compute G_{k,t} = r_{k,t} + γ r_{k,t+1} + γ^2 r_{k,t+2} + ··· γ^{T_i - 1} r_{k,T_i} ∀t
    for t = 1, ..., T do
        if First visit to (s,a) in episode k then
            N(s,a) = N(s,a) + 1                                  target
            Q(s_t, a_t) = Q(s_t, a_t) + (1 / N(s,a)) (G_{k,t} - Q(s_t, a_t))
        end if
    end for
    k = k + 1
end loop
```

Where:

- Line 1: Initialize $Q(s,a) = 0, N(s,a) = 0 \ \forall (s,a)$, $k = 1$, Input $\epsilon = 1$, π
- Sample step: $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,T})$ given π
- Compute $G_{k,t} = r_{k,t} + \gamma r_{k,t+1} + \gamma^2 r_{k,t+2} + \cdots \gamma^{T_i - 1} r_{k,T_i} \ \forall t$
- $N(s,a) = N(s,a) + 1$
- $Q(s_t, a_t) = Q(s_t, a_t) + \frac{1}{N(s,a)}(G_{k,t} - Q(s_t, a_t))$
- $k = k + 1$

Monte Carlo Online Control / On Policy Improvement

Initialize $Q(s,a) = 0, N(s,a) = 0 \ \forall (s,a)$, Set $\epsilon = 1, \ k = 1$
$\pi_k = \epsilon\text{-greedy}(Q)$ // Create initial ϵ-greedy policy
loop episodes
    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
    $k = k + 1, \ \epsilon = 1/k$
    $\pi_k = \epsilon\text{-greedy}(Q)$ // Policy improvement
end loop

for each s
$\pi(s) = \arg\max_a Q(s,a)$ w/ prob $1-\epsilon$
else random

Optional Worked Example: 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 ∀(s,a), Set ε = 1, k = 1
π_k = ε-greedy(Q) // Create initial ε-greedy policy
loop
    Sample k-th episode (s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, ..., s_{k,T}) given π_k
    G_{k,t} = r_{k,t} + γ r_{k,t+1} + γ^2 r_{k,t+2} + ··· γ^{T_i - 1} r_{k,T_i}
    for t = 1, ..., 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) + (1/N(s,a)) (G_{k,t} − Q(s_t,a_t))
        end if
    end for
    k = k + 1, ε = 1/k
    π_k = ε-greedy(Q) // Policy improvement
end loop
```

policy eval

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

Table of Contents

Greedy in the Limit of Infinite Exploration (GLIE)

Definition of GLIE

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

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

$\forall s\ a$

- 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: $\epsilon_i = 1/i$

and visit all states

GLIE Monte-Carlo Control using Tabular Representations

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)
- Method 1: SARSA state action reward next state next action
- On policy: SARSA computes an estimate Q of policy used to act

General Form of SARSA Algorithm

Set initial ϵ-greedy policy π randomly, $t = 0$, initial state $s_t = s_0$
Take $a_t \sim \pi(s_t)$
Observe (r_t, s_{t+1})
loop
  Take action $a_{t+1} \sim \pi(s_{t+1})$ // Sample action from policy
  Observe (r_{t+1}, s_{t+2})
  Update Q given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left(\overbrace{r_t + \gamma Q(s_{t+1},a_{t+1})}^{\text{target}} - Q(s_t,a_t)\right)$$

  Perform policy improvement:

for all $\forall s \quad \pi(s) = \arg\max_a Q(s,a)$ w/prob $1-\epsilon$
random otherwise

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

if s_{t+2} is terminal

end loop

reset episode sample s

General Form of SARSA Algorithm

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
    $t = t + 1$ $\epsilon = 1/t$
end loop

- See worked example with Mars rover at end of slides

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, $Q(s,a) \to Q^*(s,a)$, 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

1992 1994
papers

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
  - Estimates the value of **behavior** policy (policy using to take actions in the world)
  - And then updates the behavior policy
- Q-learning
  - estimate the Q value of π^* while acting with another behavior policy π_b
- Key idea: Maintain Q estimates and bootstrap for best future value
- Recall SARSA

actual action

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

- Q-learning:

$\sum_{s'} p(s'|s,a) V^*(s')$

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

Q-Learning with ϵ-greedy Exploration

```
Initialize Q(s, a), ∀s ∈ S, a ∈ A t = 0, initial state s_t = s_0
Set π_b to be ε-greedy w.r.t. Q
loop
    Take a_t ~ π_b(s_t) // Sample action from policy
    Observe (r_t, s_{t+1})
    Q(s_t, a_t) ← Q(s_t, a_t) + α(r_t + γ max_a Q(s_{t+1}, a) − Q(s_t, a_t))
    π(s_t) = arg max_a Q(s_t, a) w.prob 1 − ε, else random
    t = t + 1        ε = 1/t
end loop
```

See optional worked example and optional understanding check at the end of the slides

Q-Learning with ϵ-greedy Exploration

*tabular

- 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).

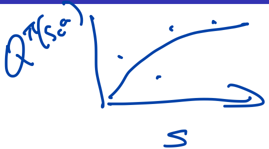
- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal π^*?
  The algorithm is GLIE, along with the above requirement to ensure the Q value estimates converge to the optimal Q.

Table of Contents

Motivation for Function Approximation

- Avoid explicitly storing or learning the following for every single state and action
  - Dynamics or reward model
  - Value
  - State-action value
  - Policy
- Want more compact representation that generalizes across state or states and actions
  - 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 $(P, R)/V/Q/\pi$

State Action Value Function Approximation for Policy Evaluation with an Oracle



1 scalar

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

neural net

Stochastic Gradient Descent

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

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

- 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:

$$\nabla J(w) = -2\, E_\pi [(Q^\pi(s,a) - \hat{Q}(s,a,w))\nabla_w \hat{Q}$$

- 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 $Q^\pi(s,a)$ and its approximation $\hat{Q}(s,a;\boldsymbol{w})$ as represented with a particular function class parameterized by $\boldsymbol{w}$.
- Generally use mean squared error and define the loss as

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

- 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:

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

- Expected SGD is the same as the full gradient update

Table of Contents

Model Free VFA Policy Evaluation

- No oracle to tell true $Q^{\pi}(s, a)$ for any state s and action a
- Use model-free state-action 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**

Table of Contents

Monte Carlo Value Function Approximation

- Return G_t is an unbiased but noisy sample of the true expected return $Q^{\pi}(s_t, a_t)$
- Therefore can reduce MC VFA to doing supervised learning on a set of (state,action,return) pairs: $\langle (s_1, a_1), G_1 \rangle, \langle (s_2, a_2), G_2 \rangle, \ldots, \langle (s_T, a_T), G_T \rangle$
  - Substitute G_t for the true $Q^{\pi}(s_t, a_t)$ when fit function approximator

MC Value Function Approximation for Policy Evaluation

Initialize **w**, $k = 1$
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, a) in episode k **then**
            $G_t(s, a) = \sum_{j=t}^{L_k} r_{k,j}$
            $\nabla_{\boldsymbol{w}} J(w) = -2[G_t(s, a) - \hat{Q}(s_t, a_t; w)] \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; w)$ (Compute Gradient)   would like $Q(s,a)$
            Update weights $\Delta \boldsymbol{w}$
        end if
    end for
    $k = k + 1$
end loop

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
- Note: Unlike MC we will focus on V instead of Q for policy evaluation here, because there are more ways to create TD targets from Q values than V values

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

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

- Use stochastic gradient descent, as in MC methods

TD(0) Value Function Approximation for Policy Evaluation

Initialize $\mathbf{w}, \mathbf{s}$
loop
  Given s sample $a \sim \pi(s)$, $r(s,a)$,$s' \sim p(s'|s,a)$
  $\nabla_{\boldsymbol{w}} J(w) = -2[r + \gamma \hat{V}(s';w) - \hat{V}(s;w)] \nabla_{\boldsymbol{w}} \hat{V}(s;w)$
  Update weights $\Delta \boldsymbol{w}$
  if s' is not a terminal state **then**
    Set $s = s'$
  else
    Restart episode, sample initial state s
  end if
end loop

Table of Contents

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

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

- Stochastic gradient descent (SGD) samples the gradient

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 for true $Q(s_t, a_t)$

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

- In Monte Carlo methods, use a return G_t as a substitute target

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

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

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

- Q-learning: Uses related TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$

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

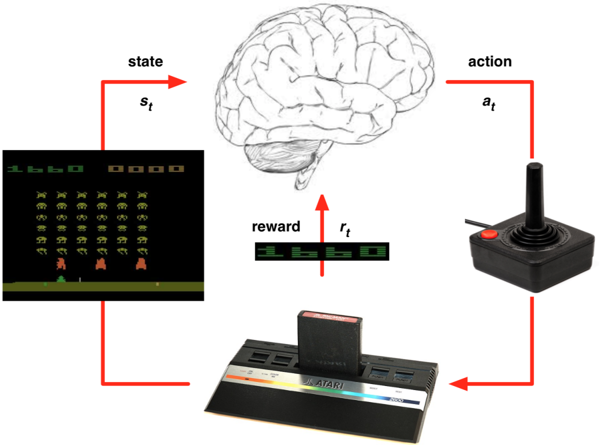
"Deadly Triad" which Can Cause Instability

- 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
  - To learn more, see Baird example in Sutton and Barto 2018
- "Deadly Triad" can lead to oscillations or lack of convergence
  - Bootstrapping
  - Function Approximation
  - Off policy learning (e.g. Q-learning)

Geoff Gordon
1995

Table of Contents

Using these ideas to do Deep RL in Atari



Q-Learning with Neural Networks

- Q-learning converges to optimal $Q^*(s, a)$ using tabular representation
- In value function approximation Q-learning minimizes MSE loss by stochastic gradient descent using a target Q estimate instead of true Q
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses these challenges by using
  - Experience replay
  - Fixed Q-targets

DQNs: Experience Replay

- To help remove correlations, store dataset (called a **replay buffer**) $\mathcal{D}$ from prior experience

s_1, a_1, r_2, s_2		
s_2, a_2, r_3, s_3	$\rightarrow$	s, a, r, s'
s_3, a_3, r_4, s_4		
$\ldots$		
$s_t, a_t, r_{t+1}, s_{t+1}$		

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

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

DQNs: Experience Replay

- To help remove correlations, store dataset $\mathcal{D}$ from prior experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow$ s, a, r, s'

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

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

- **Uses target as a scalar, but function weights will get updated on the next round, changing the target value**

DQNs: Fixed Q-Targets

Q 

- To help improve stability, fix the **target weights** used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let parameters $\boldsymbol{w}^-$ be the set of weights used in the target, and $\boldsymbol{w}$ be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$
  - Use stochastic gradient descent to update the network weights

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

weights

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

Note there are several hyperparameters and algorithm choices. One needs to choose the neural network architecture, the learning rate, and how often to update the target network. Often a fixed size replay buffer is used for experience replay, which introduces a parameter to control the size, and the need to decide how to populate it.

Check Your Understanding L4N3: Fixed Targets

- In DQN we compute the target value for the sampled (s, a, r, s') using a separate set of target weights: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^{-})$
- Select all that are true
- This doubles the computation time compared to a method that does not have a separate set of weights
- This doubles the memory requirements compared to a method that does not have a separate set of weights
- Not sure

Check Your Understanding L4N3: Fixed Targets. **Solutions**

- In DQN we compute the target value for the sampled (s, a, r, s') using a separate set of target weights: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^{-})$
- Select all that are true
- This doubles the computation time compared to a method that does not have a separate set of weights
- This doubles the memory requirements compared to a method that does not have a separate set of weights
- Not sure

Answer: It doubles the memory requirements.

DQNs Summary

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{w}^-$
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

DQNs in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step
- Used a deep neural network with CNN
- Network architecture and hyperparameters fixed across all games

DQN

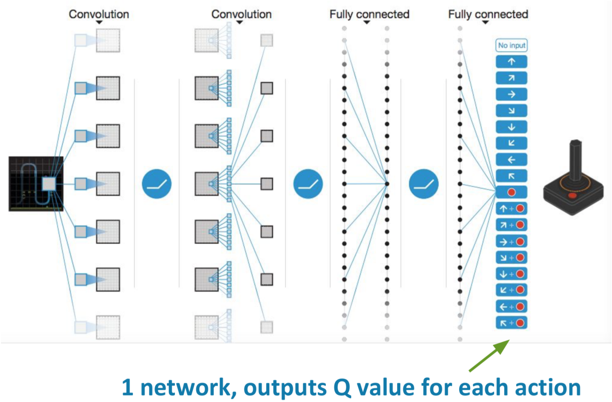


Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

DQN Results in Atari

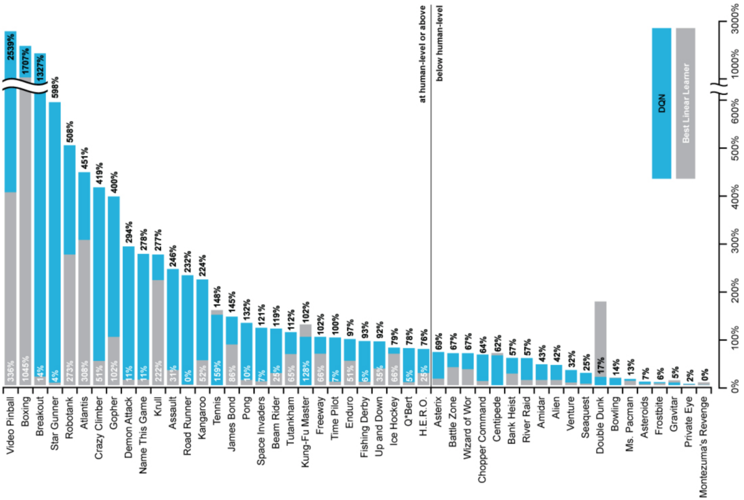


Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network
Breakout	3	3
Enduro	62	29
River Raid	2345	1453
Seaquest	656	275
Space Invaders	301	302

Note: just using a deep NN actually hurt performance sometimes!

Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network	DQN w/ fixed Q
Breakout	3	3	10
Enduro	62	29	141
River Raid	2345	1453	2868
Seaquest	656	275	1003
Space Invaders	301	302	373

Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network	DQN w/ fixed Q	DQN w/ replay	DQN w/replay and fixed Q
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Seaquest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

- Replay is **hugely** important
- Why? Beyond helping with correlation between samples, what does replaying do?

Deep RL

- Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - **Double DQN** (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
  - Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
  - Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)

What You Should Understand

- Be able to implement TD(0) and MC on policy evaluation
- 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
- Know some of the key features in DQN that were critical (experience replay, fixed targets)

Class Structure

- Last time and start of this time: Model-free reinforcement learning with function approximation
- Next time: Policy gradients

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 $V^{\pi_{i+1}} \geq V^{\pi_i}$

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

SARSA Initialization Conceptual Question

- 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

Optional Worked Example: MC for On Policy Control Solution

- 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:

- $Q^{\epsilon-\pi}(-, a_2) = [0\ 1\ 0\ 0\ 0\ 0\ 0]$
- 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 $5/6$.

Optional Worked Example SARSA for Mars Rover

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

- 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

```
Set initial ϵ-greedy policy π, t = 0, initial state s_t = s_0
Take a_t ∼ π(s_t) // Sample action from policy
Observe (r_t, s_{t+1})
loop
    Take action a_{t+1} ∼ π(s_{t+1})
    Observe (r_{t+1}, s_{t+2})
    Q(s_t, a_t) ← Q(s_t, a_t) + α(r_t + γQ(s_{t+1}, a_{t+1}) − Q(s_t, a_t))
    π(s_t) = arg max_a Q(s_t, a) w.prob 1 − ϵ, else random
    t = t + 1
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

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

- 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: $(s_6, a_1, 0, s_7, a_2, 5, s_7)$.
- $Q(s_6, a_1) = .5 * 0 + .5 * (0 + \gamma Q(s_7, a_2)) = 2.5$

Worked Example: ϵ-greedy Q-Learning Mars

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

- 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

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

- 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

Optional Check Your Understanding L4: 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

Optional Check Your Understanding SARSA and Q-Learning Solutions

- 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.