# Model Free Control and Function Approximation

## Introduction to Reinforcement Learning

**Bogdan Ivanyuk-Skulskyi, Dmytro Kuzmenko**

Department of Mathematics,
National University of Kyiv-Mohyla Academy

March 6, 2023

# Table of Contents

# 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

# Model-free Policy Iteration

- Init policy $\pi$
- Repeat:
  - Policy evaluation: compute $Q^\pi$
  - Policy improvement: update $\pi$ given $Q^\pi$
- May need to modify policy evaluation:
  - If $\pi$ 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)

# $\epsilon$ - greedy Policies

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

## Policy Improvement with $\epsilon$-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 $\epsilon$-greedy policies

# Monotonic $\epsilon$-greedy Policy Improvement

**Theorem 1**

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

$$Q^{\pi_i}(s, \pi_{i+1}(s)) = \sum_{a \in A} \pi_{i+1}(a|s) Q^{\pi_i}(s, a)$$

$$= \frac{\epsilon}{|A|} \left[ \sum_{a \in A} Q^{\pi_i}(s, a) \right] + (1 - \epsilon) max_a Q^{\pi_i}(s, a)$$

# Table of Contents

# Recall Monte Carlo Policy Evaluation

Init $Q(s, a) = 0$, $N(s, a) = 0 \; \forall (s, a)$, $k = 1$, Input $\epsilon = 1$, $\pi$
Loop

- Sample $k$-th episode from $\pi$
- Compute $G_{k,t} = r_{k,t} + \gamma r_{k,t+1} + \gamma^2 r_{k,t+2} + \ldots + \gamma^{T_i - 1} r_{k,T_i}$, $\forall t$
- for $t = 1 \ldots T$ do
    - If First visit to (s,a) in episode k then
        - $N(s, a) += 1$
        - $Q(s_t, a_t) += \frac{1}{N(s,a)}(G_{k,t} - Q(s_t, a_t))$
    - end if
- end for
- $k = k + 1$

# Greedy in the Limit of Infinite Exploration (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

$$\lim_{i \to \infty} \pi(a|s) \to argmax_a Q(s, a)$$

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

# GLIE Monte-Carlo Control

**Theorem 2**

*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 $\pi$
- Repeat:
  - Policy evaluation: compute $Q^\pi$ using temporal difference updating with $\epsilon$-greedy policy
  - Policy improvement: Same as Monte Carlo policy improvement, set $\pi$ to $\epsilon$-greedy $(Q^\pi)$
- 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

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

# Example: SARSA for Mars Rover

- Initialize $\epsilon = \frac{1}{k}$, $k = 1$, 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: $(s_6, a_1, 0, s_7, a_2, 5, s_7)$
- $Q(s_6, a_1) = .50 + .5(0 + \gamma Q(s_7, a_2)) = 2.5$

# PROPERTIES OF SARSA WITH $\epsilon$-GREEDY POLICIES

- Convergence:

**Theorem 3**

*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:*

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

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

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

- 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

## 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, to directly estimate the value of $\pi^*$ while acting with another behavior policy $\pi_b$ use
- 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 that (behavior) policy
- Alternatively, to directly estimate the value of $\pi^*$ while acting with another behavior policy $\pi_b$ use
- Q-learning, an off-policy RL algorithm
- 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 $\epsilon$-greedy Exploration

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

## Example: Q-learning for Mars Rover

- 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 conditions, but at the beginning, yes

# Q-LEARNING WITH $\epsilon$-GREEDY EXPLORATION

- What conditions are sufficient to ensure that $Q$-learning with $\epsilon$-greedy exploration converges to optimal $Q$?
  Visit all (s,a) pairs infinitely often, and the step-sizes $\alpha_t$ 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 $\epsilon$ large)
- What conditions are sufficient to ensure that Q-learning with $\epsilon$-greedy exploration converges to optimal $\pi^*$?
  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
- Two very popular classes of differentiable function approximators
  - Linear feature representations
  - Neural networks

# 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 $V^{\pi}$ given a particular parameterized function $\hat{V}(s; w)$

# 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

# 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

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

- Objective function is

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

- weight update is

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

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

## 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: $<s_1, G_1>, <s_2, G_2>, \ldots, <s_T, G_T>$
  - Substitute $G_t$ for the true $V^\pi(s_t)$ when fit function approximator
- Concretely when using linear VFA for policy evaluation

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

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

# MC Linear Value Function Approximation for Policy Evaluation

- Init $w = 0$,$k = 1$
- Loop
  - Sample $k$-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, ..., s_{k,L_k})$ given $\pi$
  - for $t = 1, ..., L_k$ do
    - if First visit to (s) in episode k then
    - $G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$
    - Update weights: $w = +\alpha(G_t(s) - x(s)^T w)x(s)$
- $k = k + 1$

# Table of Contents

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

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

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

- Target is $r + \gamma V^\pi(s')$, a biased estimate of the true value $V^\pi(s)$
- In value function approximation, target is $r + \gamma V^\pi(s'; w)$, a biased and approximated estimate of the true value $V^\pi(s)$
- 3 forms of approximation:
  - Sampling
  - Bootstrapping
  - Value function approximation

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

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

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

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

- In value function approximation, target is $r + \gamma V^\pi(s'; w)$, 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: $<s_1, r_1 + \gamma V^\pi(s_2; w)>, <s_2, r_2 + \gamma V^\pi(s_3; w)> \ldots$
- In linear TD(0)

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

# TD(0) Linear Value Function Approximation for Policy Evaluation

- Initialize $w = 0$, $k = 1$
- Loop
  - Sample tuple $(s_k, a_k, r_k, s_{k+1})$ given $\pi$
  - Update weights:

$$w = w + (r + \gamma x(s')^T w - x(s)^T w)x(s)$$

  - $k+ = 1$

# 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 $\pi$ relative to the true value as

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

- where
  - $\mu(s)$: probability of visiting state s under policy $\pi$. Note $\sum_s \mu(s) = 1$
  - $V^{\pi}(s; w) = x(s)^T w$ , 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 $\pi$ relative to the true value as

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

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

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

# 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 relative to the true value given the distribution d as

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

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

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