Model-free Control

Weizi Li

Department of Computer Science
University of Memphis

- Introduction
- On-policy Learning: MC Control
- On-policy Learning: TD Control
- Off-policy Learning: MC/TD + Importance Sampling
- Off-policy Learning: Q-learning
- Summary

Introduction

Some example problems that can be modelled as MDPs

- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Quake
- Portfolio management
- Protein Folding
- Robot walking
- Game of Go

For most of these problems, either:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

Model-free control can solve these problems

On-policy vs. Off-policy Learning

- On-policy learning: update π using sampled trajectories from π itself
- Off-policy learning: update π using sampled trajectories from another policy

On-policy Learning: MC Control

- Policy iteration = policy evaluation + policy improvement
- Policy evaluation
  - Model-based: iteratively applying the Bellman expectation equation until converge to V^π
  - Model-free: MC policy evaluation $V \approx V^\pi$
- Policy improvement: greedy policy improvement
- Can we solve the control problem using MC policy evaluation on $V(s)$ and greedy policy improvement?

- Issue: greedy policy improvement over $V(s)$ requires the model
$$\pi'(s) = \arg\max_a \left(R_s^a + \gamma \sum P_{ss'}^a V(s') \right)$$
- Solution: greedy policy improvement over $Q(s, a)$ is model-free
$$\pi'(s) = \arg\max_a Q(s, a)$$

- Policy evaluation: MC policy evaluation $Q \approx q_\pi$.
- Policy improvement: greedy policy improvement?

- Issue: acting greedily to Q may result in some states/actions never get explored
- Solution: ϵ-greedy

- All actions are tried with non-zero probability
- With probability $1 - \epsilon$, choose the greedy action
- With probability ϵ, choose an action randomly

- Policy evaluation: MC policy evaluation $Q \approx q_\pi$
- Policy improvement: ϵ-greedy policy improvement

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\
&= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\
&\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s)
\end{aligned}
$$

Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

- Update the policy after every episode so that it gets improved based on the most recent Q
- One more issue: no need to explore more if we have obtained the optimal policy
- Solving this issue gives us the first full solution for finding the optimal policy in an unknown MDP: GLIE Monte-Carlo Control

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \to \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname*{argmax}_{a' \in \mathcal{A}} Q_k(s, a'))$$

- For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

- Sample kth episode using π: $\{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

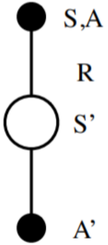
GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

On-policy Learning: TD Control

- Policy evaluation: TD policy evaluation $Q \approx q_\pi$
- Policy improvement: ϵ-greedy policy improvement

- TD has lower variance (but higher bias)
- TD can learn from incomplete trajectories

- Update Q by moving towards the TD target



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A) \right)$$

- Policy evaluation: Sarsa $Q \approx q_\pi$
- Policy improvement: ϵ-greedy policy improvement
- Update the policy after every *time step* so that it can be improved based on the most recent Q
- "The best known algorithm in RL."—David Silver

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Theorem

Sarsa converges to the optimal action-value function, $Q(s,a) \to q_*(s,a)$, under the following conditions:

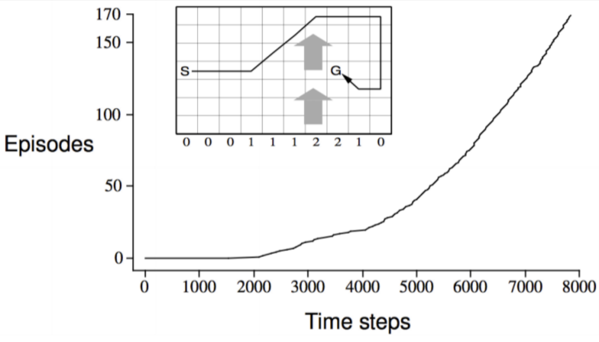
- *GLIE sequence of policies $\pi_t(a|s)$*
- *Robbins-Monro sequence of step-sizes α_t*

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

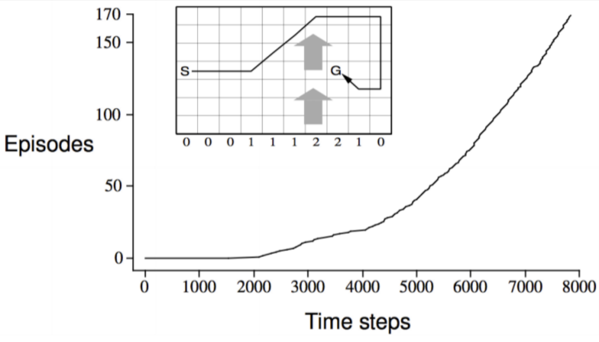
$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

- Sarsa can work even without satisfying these conditions.

- X-axis: the number of time steps used
- Y-axis: the number of episodes completed (one episode: from s to G)



- It takes about 2000 steps to finish the first episode. After that, less and less steps are needed to finish an episode.



- Consider the following *n*-step returns for $n = 1, 2, \infty$:

$$
\begin{array}{lll}
n = 1 & (\textit{Sarsa}) & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\
n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T
\end{array}
$$

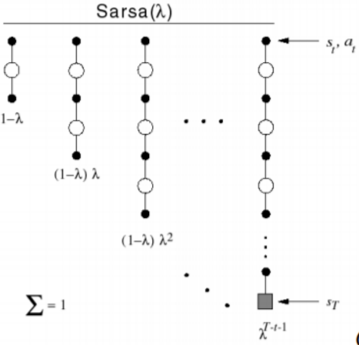
- Define the *n*-step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

- *n*-step Sarsa updates $Q(s, a)$ towards the *n*-step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)$$

Forward-view Sarsa(λ)



- The q^λ *return* combines all n-step Q-returns $q_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}q_t^{(n)}$$

- Forward-view Sarsa(λ)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(q_t^\lambda - Q(S_t, A_t)\right)$$

- Just like TD(λ), we use eligibility traces in an online algorithm
- But Sarsa(λ) has one eligibility trace for each state-action pair

$$E_0(s, a) = 0$$
$$E_t(s, a) = \gamma \lambda E_{t-1}(s, a) + \mathbf{1}(S_t = s, A_t = a)$$

- $Q(s, a)$ is updated for every state s and action a
- In proportion to TD-error δ_t and eligibility trace $E_t(s, a)$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$
$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)$$

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    $E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    Initialize S, A
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
        $E(S, A) \leftarrow E(S, A) + 1$
        For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$
            $E(s, a) \leftarrow \gamma \lambda E(s, a)$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal


Path taken
Action values increased by one-step Sarsa
Action values increased by Sarsa(λ) with λ=0.9

Off-policy Learning:
MC/TD + Importance Sampling

- Goal: evaluate π (compute V^π or Q^π) using sampled trajectories from μ
- π: target policy; the policy we want to update until it becomes the optimal policy
- μ: behavior policy; the policy actually generates actions being executed in the environment

- Use case 1 (multiple agents): learn from observing humans or other agents
- Use case 2 (single agent): use data from exploratory or other policies

- Goal: estimate the expectation of a function under a different distribution

- Expectation of a function under one distribution:

$$\mathbb{E}_{x\sim p(x)}[f(x)] = \sum p(x)f(x)$$

- Now, consider two distributions $p(x)$ and $q(x)$:

$$\mathbb{E}_{x\sim p(x)}[f(x)] = \sum q(x)\frac{p(x)}{q(x)}f(x) = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$$

- On-policy learning: use return from π itself, i.e., G_t
- Off-policy learning: use return from μ to evaluate π
  - weigh G_t using importance sampling corrections along the entire trajectory

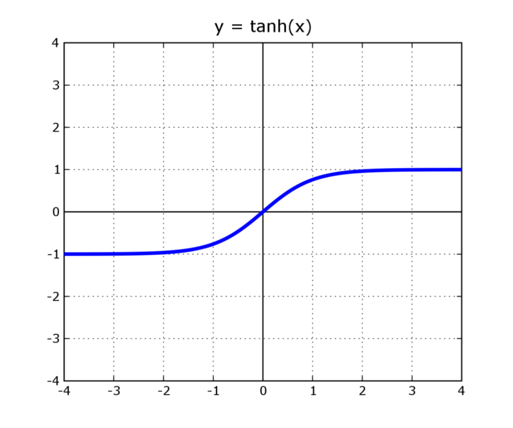
$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1})\dots\pi(A_T|S_T)}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1})\dots\mu(A_T|S_T)} G_t$$

  - update towards the *corrected* return

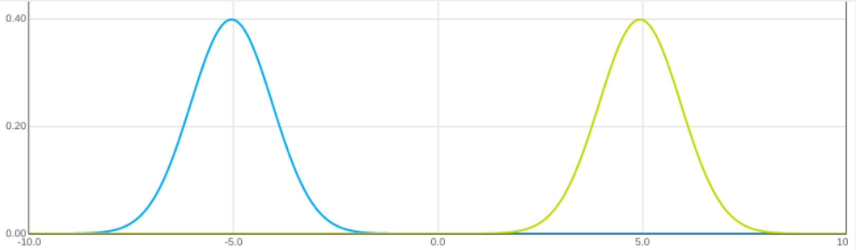
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\pi/\mu} - V(S_t)\right)$$

- Importance sampling can dramatically increase variance
- π and μ need to be similar over the entire trajectory for this to work, which is rare in practice (do not use this method!)

- Consider: $f(x) = tanh(x)$



- Consider: $p(x)$=left Gaussian, $q(x)$=right Gaussian



- Consider: $f(x) = tanh(x)$, $p(x)$=left Gaussian, $q(x)$=right Gaussian
- In theory
  - $\mathbb{E}_{x\sim p(x)}[f(x)] = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$
- In practice
  - $\mathbb{E}_{x\sim p(x)}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ is likely to be negative
  - $\mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \approx \frac{1}{N}\sum_{i=1}^{N} \frac{p(x_i)}{q(x_i)}f(x_i)$ is likely to be positive

- On-policy learning: use TD target from π itself, $R_{t+1} + \gamma V(S_{t+1})$
- Off-policy learning: use TD target from μ to evaluate π
  - weigh $R_{t+1} + \gamma V(S_{t+1})$ using importance sampling correction for one step
  - update towards the *corrected* TD target

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \left(R_{t+1} + \gamma V(S_{t+1})\right) - V(S_t) \right)$$

- π and μ only need to be similar over one step
- Much lower variance than "MC + importance sampling"

Off-policy Learning: Q-learning

- Goal: off-policy learning w.r.t $Q(s, a)$
- The action being executed in the environment is from μ, $A_t \sim \mu(\cdot|S_t)$. After A_t, environment transits agent to S_{t+1}
- Now, we consider an alternative successor action from π, $A' \sim \pi(\cdot|S_{t+1})$, even we don't execute it
- Next we update $Q(S_t, A_t)$ towards the value of the alternative action:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q\left(S_{t+1}, A'\right) - Q\left(S_t, A_t\right)\right)$$

- Q-learning doesn't require importance sampling and works much better than "MC/TD + importance sampling"

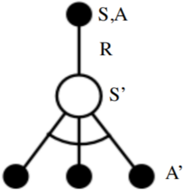
- Goal: learn a greedy policy π by following an exploratory policy μ
  - π is greedy w.r.t $Q(s, a)$

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

  - μ is ϵ-greedy w.r.t $Q(s, a)$

- Allow both π and μ to be improved via updating Q
- Now, the Q-learning target becomes:

$$
\begin{aligned}
R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a')) \\
&= R_{t+1} + \gamma max_{a'} Q(S_{t+1}, a')
\end{aligned}
$$



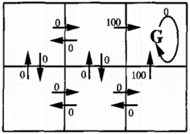
$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Theorem

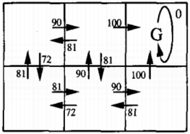
Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$;
  until S is terminal

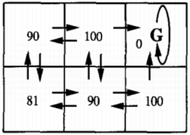
- Q-learning is off-policy because A is from μ, while a in $max_a Q(S', a)$ is from π



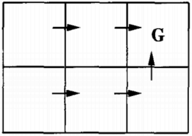
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



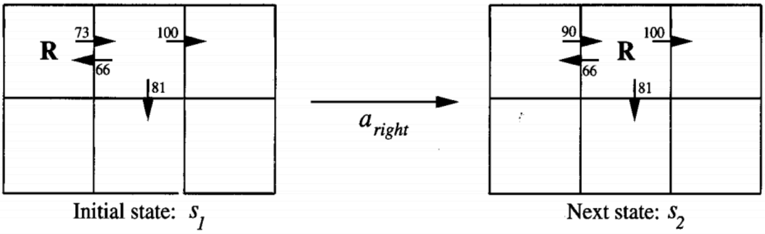
$V^*(s)$ values



One optimal policy

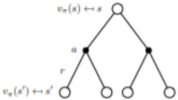
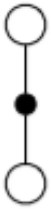
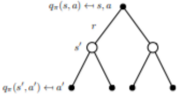
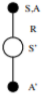
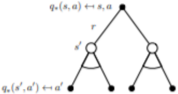
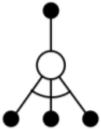
Q-learning Example

- By setting $\alpha = 1$, we have $Q(S, A) \leftarrow R + \gamma max_{a'} Q(S', a')$



$$
\begin{aligned}
\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
&\leftarrow 0 + 0.9 \ \max\{66, 81, 100\} \\
&\leftarrow 90
\end{aligned}
$$

Summary

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s,a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s,a)$	 Q-Value Iteration	 Q-Learning

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	$V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}[R + \gamma Q(S',A') \mid s,a]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$

- We have introduced three fundamental classes of methods for solving finite MDPs: DP, MC, and TD.
- These methods can often find exact solutions (with theoretical guarantees), i.e., the optimal value function and the optimal policy, when states or state-action values can be fit into a table.

- Each method class has strengths and weaknesses; they differ in efficiency and convergence rate.
- DP is well developed mathematically, but requires a complete and accurate model of the environment.
- MC doesn't require a model and is conceptually simple, but not well suited for incremental computation.
- TD requires no model and is fully incremental, but more complex to analyze.