

Lecture 5: Model-Free Control

David Silver

Outline

- 1 Introduction
- 2 On-Policy Monte-Carlo Control
- 3 On-Policy Temporal-Difference Learning
- 4 Off-Policy Learning
- 5 Summary

Model-Free Reinforcement Learning

- Last lecture:
 - **Model-free prediction**
 - *Estimate* the value function of an *unknown* MDP
- This lecture:
 - **Model-free control**
 - *Optimise* the value function of an **unknown** MDP

Uses of Model-Free Control

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:

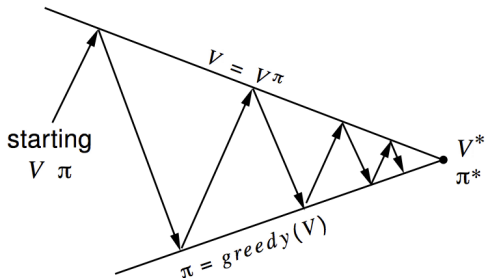
- MDP model is **unknown**, but experience can be **sampled**
- MDP model is known, but is **too big** to use, except by **samples**

Model-free control can solve these problems

On and Off-Policy Learning

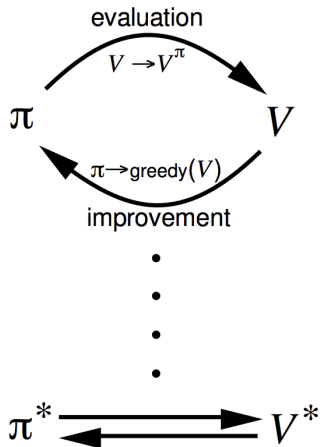
- **On-policy** learning
 - “Learn on the job”
 - Learn about policy π from experience sampled from π
- **Off-policy** learning
 - “Look over someone’s shoulder”
 - Learn about policy π from experience sampled from μ

Generalised Policy Iteration (Refresher)



Policy evaluation Estimate v_π
 e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
 e.g. Greedy policy improvement



Generalised Policy Iteration With Monte-Carlo Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Model-Free Policy Iteration Using Action-Value Function

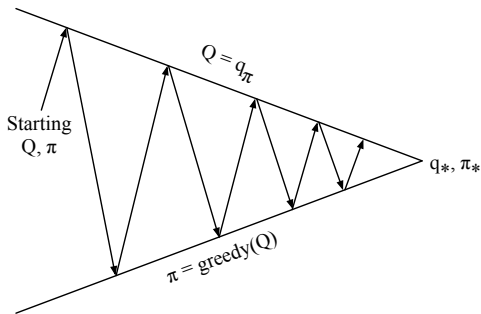
- Greedy policy improvement over $V(s)$ requires **model** of MDP

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over **$Q(s, a)$** is model-free

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

Generalised Policy Iteration with Action-Value Function



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Example of Greedy Action Selection



"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

- There are two doors in front of you.
- You open the left door and get reward 0
 $V(\text{left}) = 0$
- You open the right door and get reward +1
 $V(\text{right}) = +1$
- You open the right door and get reward +3
 $V(\text{right}) = +2$
- You open the right door and get reward +2
 $V(\text{right}) = +2$
- \vdots
- Are you sure you've chosen the best door?

ϵ -Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$



ϵ -Greedy Policy Improvement

Theorem

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



$$q_\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a)$$

$$= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a)$$

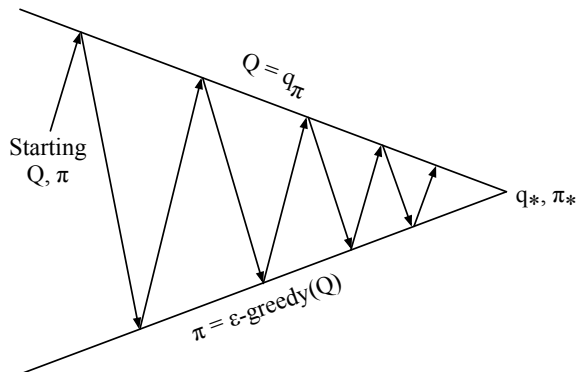
$$\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a)$$



$$= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s)$$

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

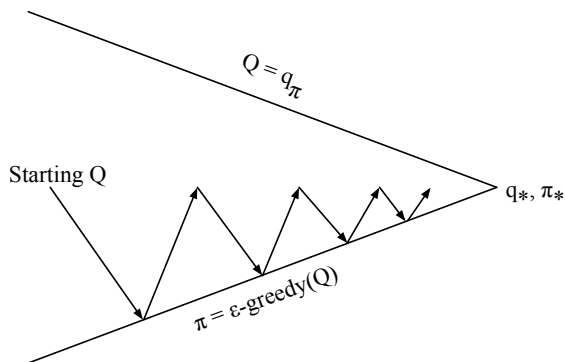
Monte-Carlo Policy Iteration



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement ϵ -greedy policy improvement

Monte-Carlo Control



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ -greedy policy improvement

GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

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

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

- The policy converges on a greedy policy,

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

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

GLIE Monte-Carlo Control

- Sample k th episode using π : $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- 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

$$\epsilon \leftarrow 1/k$$

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

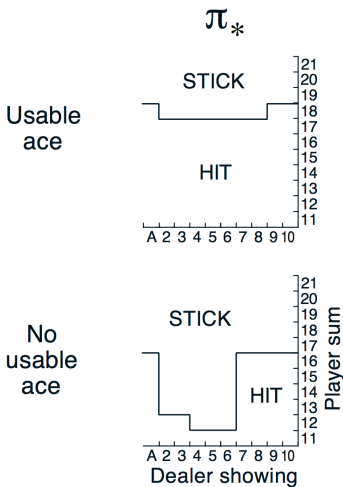
Theorem

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

Back to the Blackjack Example



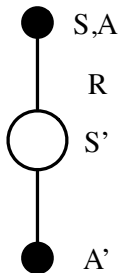
Monte-Carlo Control in Blackjack



MC vs. TD Control

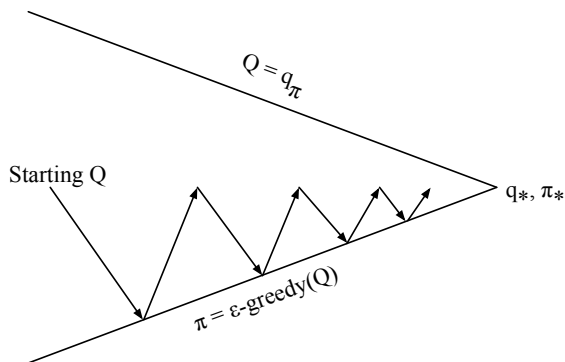
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
 - Lower variance
 - Online
 - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
 - Apply TD to $Q(S, A)$
 - Use ϵ -greedy policy improvement
 - Update every time-step

Updating Action-Value Functions with Sarsa



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

On-Policy Control With Sarsa



Every **time-step**:

Policy evaluation **Sarsa**, $Q \approx q_\pi$

Policy improvement ϵ -greedy policy improvement

Sarsa Algorithm for On-Policy Control

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

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)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

until S is terminal

Convergence of Sarsa

Theorem

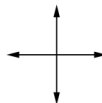
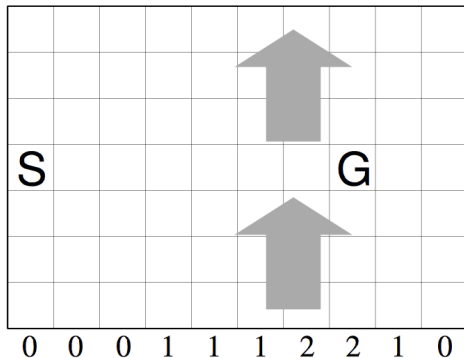
Sarsa converges to the optimal action-value function, $Q(s, a) \rightarrow q_(s, a)$, under the following conditions:*

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

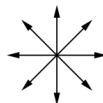
$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

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

Windy Gridworld Example



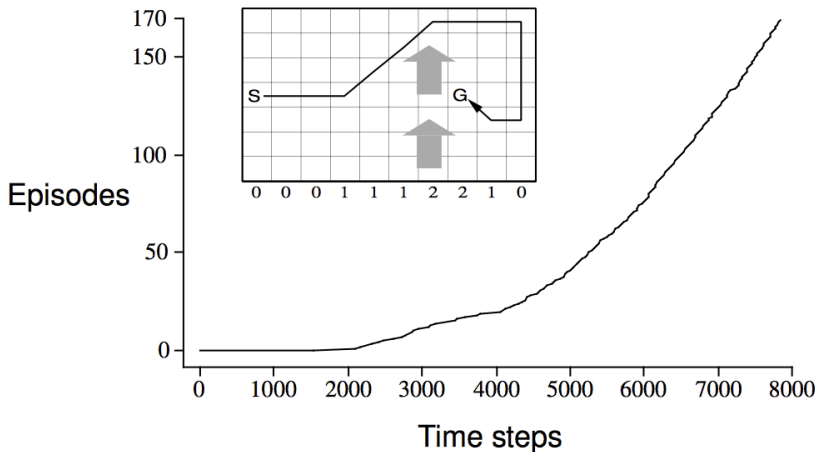
standard
moves



king's
moves

- Reward = -1 per time-step until reaching goal
- Undiscounted

Sarsa on the Windy Gridworld



n -Step Sarsa



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

$$n = 1 \quad (\text{Sarsa}) \quad q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1})$$

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

$$\vdots$$
$$\vdots$$

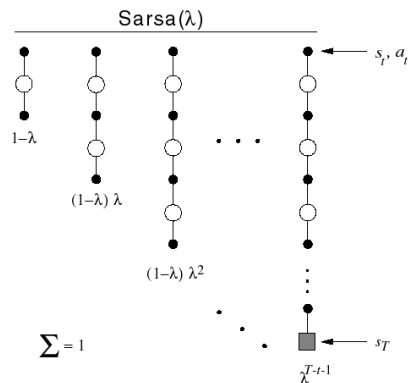
$$n = \infty \quad (\text{MC}) \quad q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Define the n -step Q-return

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

- 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)

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

Backward View Sarsa(λ)

- 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 $E_t(s, a)$

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

$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)$$

Sarsa(λ) Algorithm

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)

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$E(S, A) \leftarrow E(S, A) + \delta$

 For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

$E(s, a) \leftarrow \gamma \lambda E(s, a)$

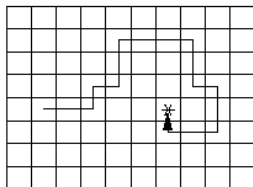
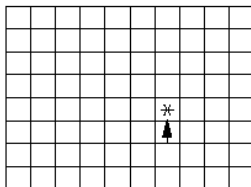
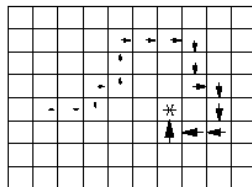
$S \leftarrow S'; A \leftarrow A'$

 until S is terminal



Sarsa(λ) Gridworld Example

Path taken

Action values increased
by one-step SarsaAction values increased
by Sarsa(λ) with $\lambda=0.9$ 

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following **behaviour** policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{t-1}$
- Learn about *optimal* policy while following **exploratory policy**
- Learn about **multiple policies** while following *one* policy

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

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

Off-Policy Control with Q-Learning

- We now allow **both** behaviour and target policies to **improve**
- The target policy π is **greedy** w.r.t. $Q(s, a)$

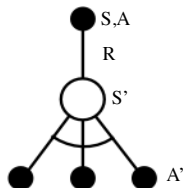
$$\pi(S_{t+1}) = \operatorname{argmax}_{a'} Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. **ϵ -greedy** w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

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



Q-Learning Control Algorithm



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

Theorem

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

Q-Learning Algorithm for Off-Policy Control

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

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)

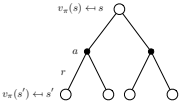
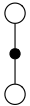
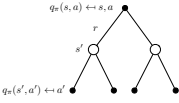

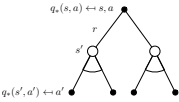
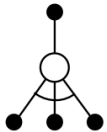
 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$;

 until S is terminal

Relationship Between DP and TD

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

Relationship Between DP and TD (2)

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

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