Reinforcement Learning & Intelligent Agents

Lecture 7: Model-Free Control

S. M. Ahsan Kazmi

Recap

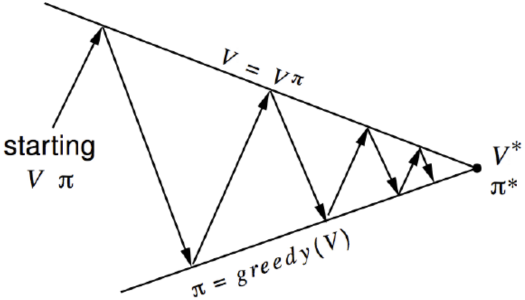
Last lecture:

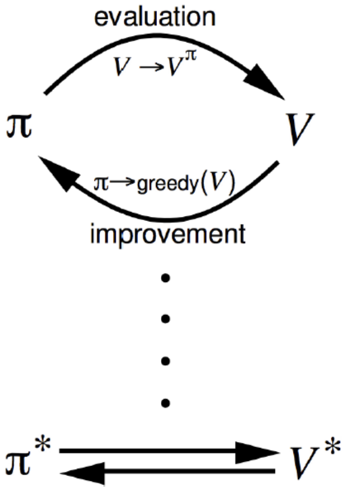
- Model-free prediction to estimate values in an unknown MDP
    - Temporal-Difference Learning

This lecture:

- This lecture: Model-free control
    - Optimise the value function of an unknown MDP
    - On-Policy Learning
    - Off-Policy Learning

Recap: Generalized Policy Iteration





Policy evaluation Estimate v_π

e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

e.g. Greedy policy improvement

Why

- For most of these problems, either:
    - MDP model is unknown, but the experience can be sampled
    - MDP model is known but is too big to use, except for samples

- Model-free control can solve these problems

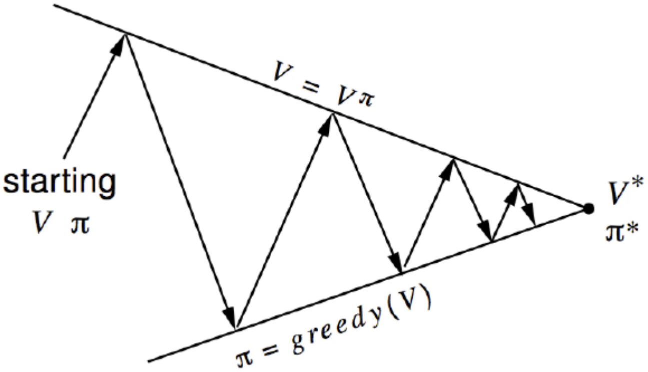
- On-policy learning
    - Learn on the job
    - Learn about policy π from experience sampled from π

- Off-policy learning
    - Look over someone's shoulder
    - Learn about policy π on experience sampled from some other policy distribution

On-Policy Learning

- Monte-Carlo Control
- TD Control

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

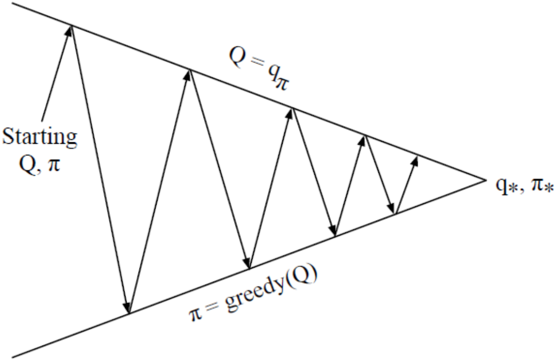
- Greedy policy improvement over V(s) requires a model of MDP

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

- Greedy policy improvement over Q(s; a) is model-free

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

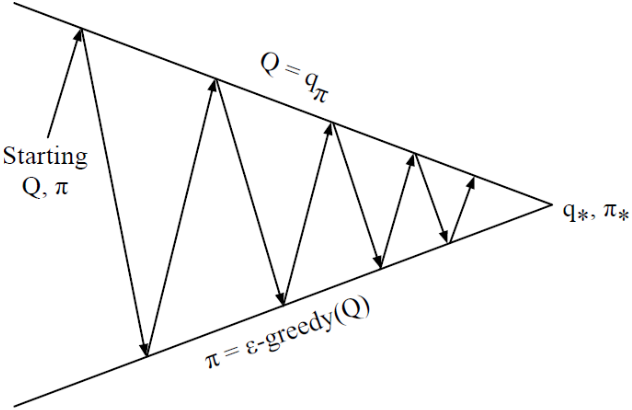
Generalised Policy Iteration with Action-Value Function



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

Policy improvement Greedy policy improvement? No exploration!

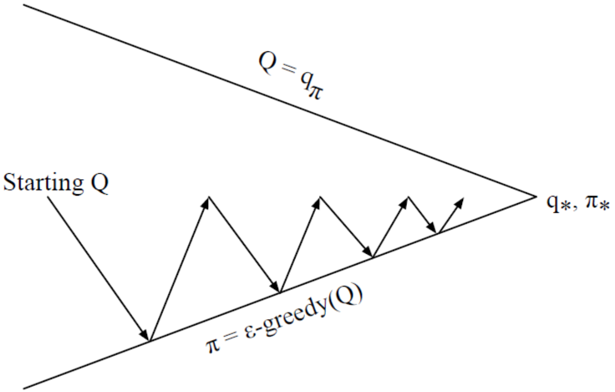
Monte-Carlo Policy Iteration



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

Greedy in the Limit with Infinite Exploration (GLIE)

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

$$\forall s, a \quad \lim_{t \to \infty} N_t(s, a) = \infty$$

- The policy converges into a greedy policy,

$$\lim_{t \to \infty} \pi_t(a|s) = \mathcal{I}(a = \operatorname*{argmax}_{a'} q_t(s, a'))$$

- For example,

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

GLIE Monte-Carlo Control

- 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

$$\epsilon \leftarrow 1/k$$

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

GLIE Model-free control converges to the optimal action-value function

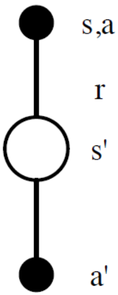
Temporal-Difference Learning For Control

MC vs. TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)

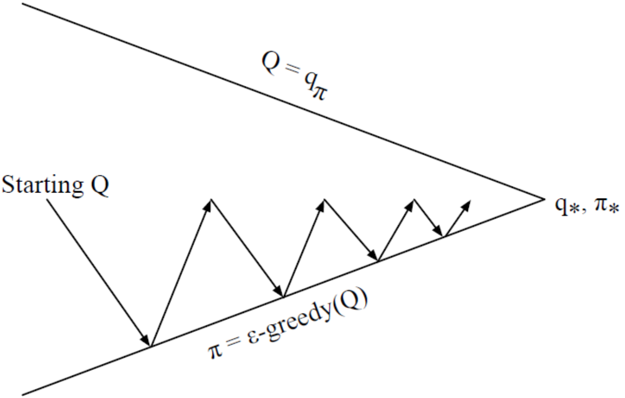
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to Q(S;A)
  - Use -greedy policy improvement
  - Update every time-step

Updating Action-Value Functions with SARSA



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

On-Policy Control With SARSA



Every time-step:

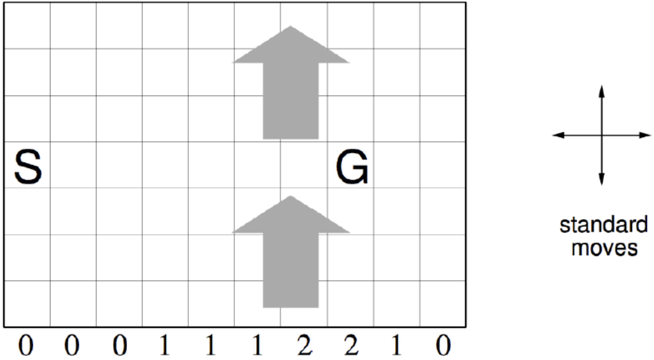
Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

SARSA Algorithm for On-Policy Control

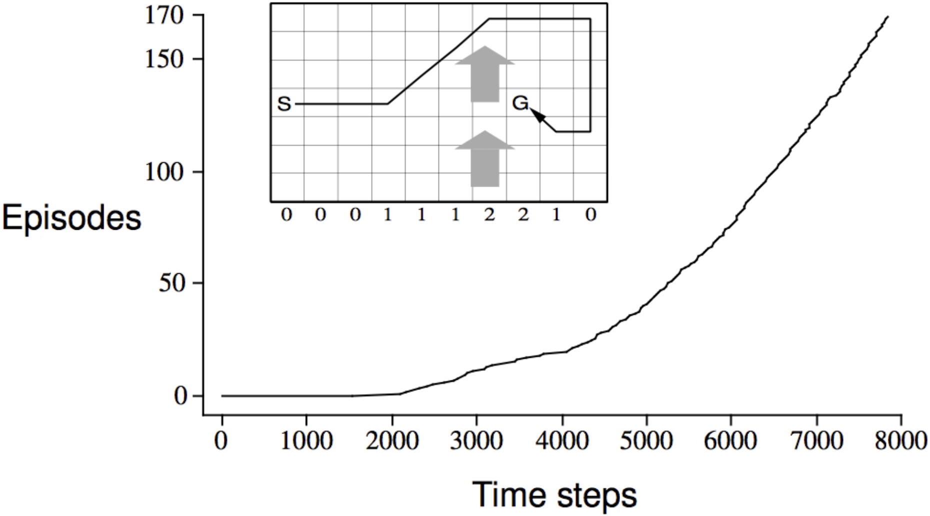
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'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

Windy Grid world Example

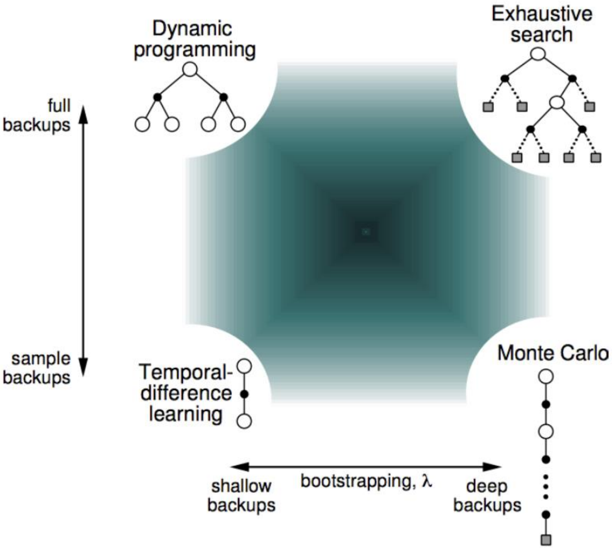


- Reward = -1 per time-step until reaching goal
- Undiscounted

Sarsa on the Windy Grid world



Unified View of Reinforcement Learning



n-Step Sarsa

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

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

- N-step Q return

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

- 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 & Backward View Sarsa

- 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 and eligibility trace

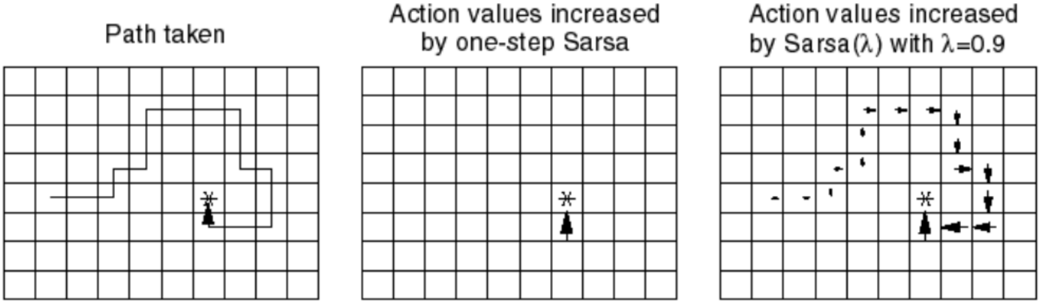
$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

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

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)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S,A) \leftarrow E(S,A) + 1$
    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

Sarsa (λ) Grid world Example



Off-Policy Learning

On and Off-Policy Learning

On-policy learning

- Learn about behaviour policy lambda from experience sampled lambda

Off-policy learning

- Learn about target policy lambda from experience sampled from other policy
- Learn ‘counterfactually’ about other things you could do: “what if...?”
- E.g., “What if I would turn left?” =) new observations, rewards?
- E.g., “What if I would play more defensively?” =) different win probability?
- E.g., “What if I would continue to go forward?” =) how long until I bump into a wall?

Off-Policy Learning

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

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

- Why is this important?

  Learn from observing humans or other agents

  Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$

  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

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

Off-Policy Control with Q-Learning

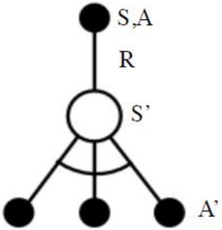
- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

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

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

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

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, $q \rightarrow q^*$, as long as we take each action in each state infinitely often.

Q-Learning Algorithm for Off-Policy Control

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

Summary

- SARSA uses a stochastic sample from the behavior as a target policy
- Q-learning uses a greedy target policy

Thanks