Model-Free Control

Model-Free RL

- Model-free prediction: how good is a specific policy?
  - Given no access to the decision process model parameters
  - Instead have to estimate from data/experience

- Model-free control: how can we learn a good policy?

Target Policy vs Behavior Policy

- All learning control methods face a dilemma: They learn action values for optimal behavior, but need to behave non-optimally to explore new actions.

- A more straightforward approach is to use two policies: The policy being learned about is called the target policy, and the policy used to generate behavior is called the behavior policy.
  - Target policy: It is the policy that an agent is trying to evaluate or improve. i.e agent is learning value function for this policy.
  - Behavior policy: It is the policy that is being used by an agent for action select. i.e agent follows this policy to interact with the environment.

- The target policy is typically the *deterministic* greedy policy with respect to the current estimate of the action-value function.
- The behavior policy remains *stochastic* and more exploratory, for example, an ϵ-greedy policy.

On-Policy vs Off-Policy

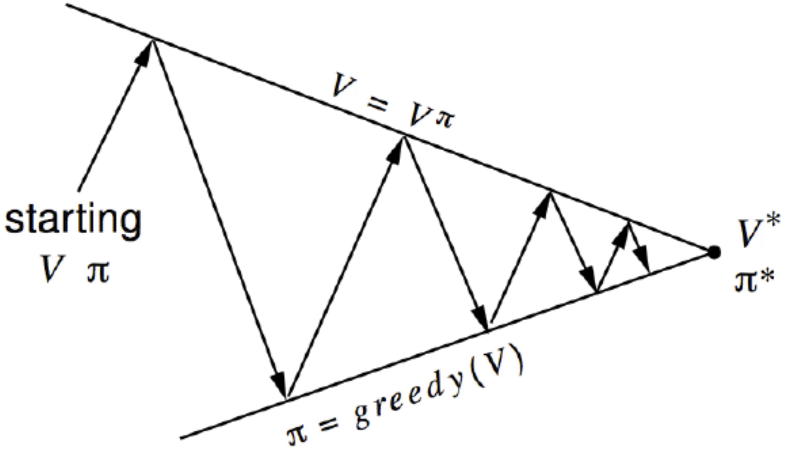
- How to learn about the optimal policy while behaving according to an exploratory policy?

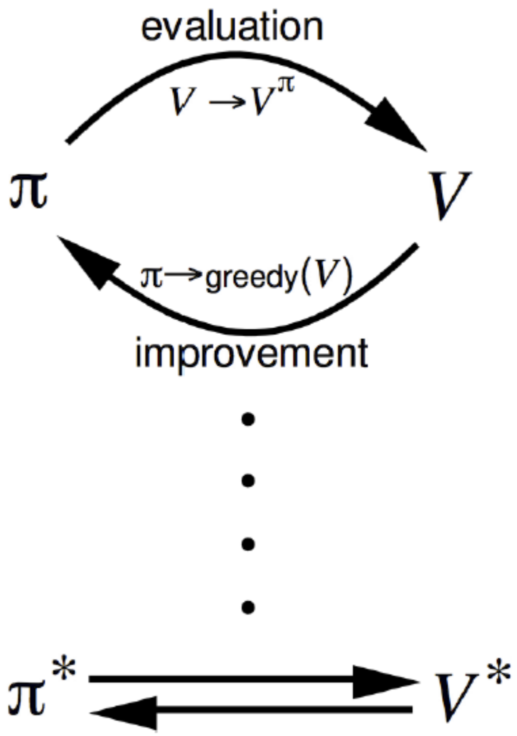
- On-policy
  - target policy = behavior policy
  - learn to estimate and evaluate a policy from experience obtained from following that policy
  - learns action values not for the optimal policy, but for a near-optimal policy that still explores.
  - generally simpler and are considered first
  - e.g. Sarsa

On-Policy vs Off-Policy

- Off-policy
  - target policy $\neq$ behavior policy
  - learn to estimate and evaluate a policy using experience gathered from following a different policy
  - more powerful and general
  - can be applied to learn from data generated by a conventional non-learning controller, or from a human expert.
  - re-use experience generated from old policies ($\pi_0, \pi_1, \ldots$).
  - are often of greater variance and are slower to converge
  - e.g. Q learning

Recap: Generalized Policy Iteration

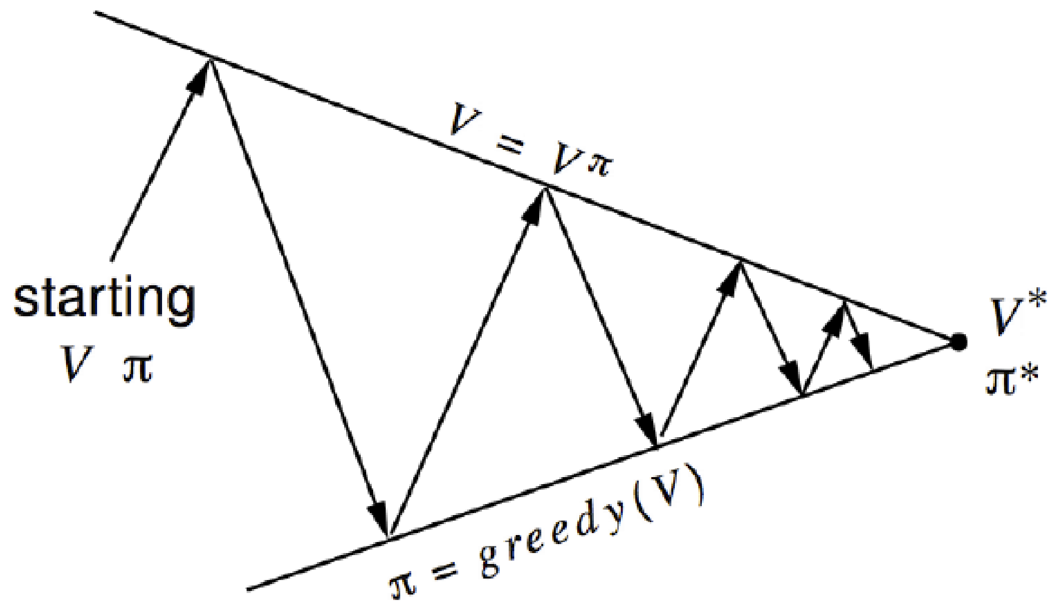




- Policy evaluation Estimate V_π
  - e.g. Iterative policy evaluation
- Policy improvement Generate $\pi' \geq \pi$
  - e.g. Greedy policy improvement

R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2018

Generalized Policy Iteration With Monte-Carlo Evaluation



- We already learned model-free policy evaluation method(e.g. MC)
- Also learned greedy policy improvement

- Therefore, how about this GPI?
  - Policy evaluation: Monte-Carlo policy evaluation, V = v_π?
  - Policy improvement: Greedy policy improvement?

Model-Free Policy Iteration Using Action-Value Function

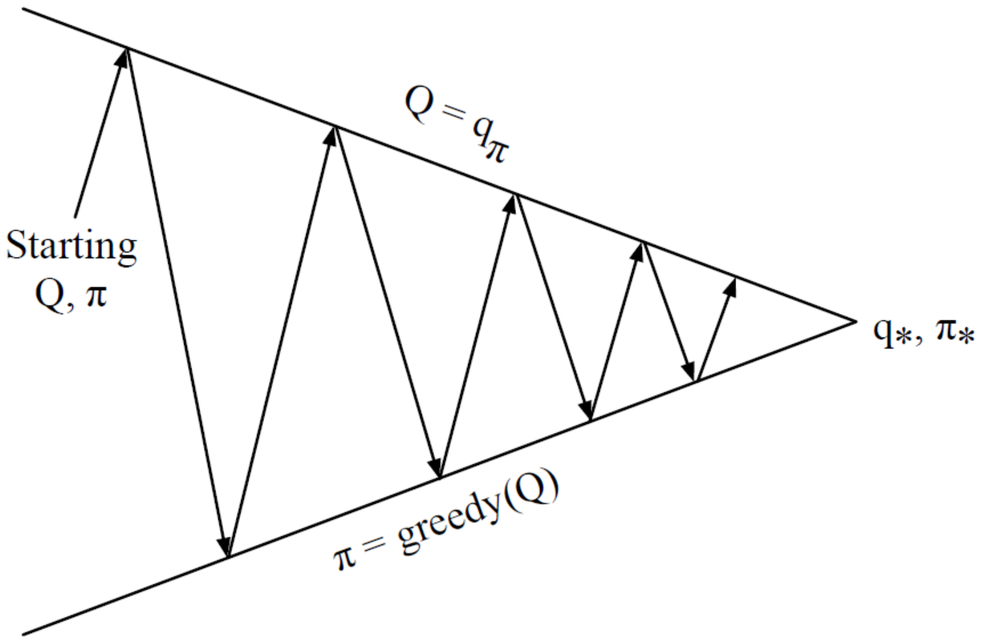
- Greedy policy improvement over **V(s)** requires model of MDP

$$\pi'(s) = argmax_{a \in A} R_s^a + P_{ss'}^a V(s')$$

- Greedy policy improvement over **Q(s,a)** is model-free
  - We **do not** know $P_{ss'}^a$ (assuming stochastic) and R_s^a
  - We need action-value function for model-free control

$$\pi'(s) = argmax_{a \in A}\ Q(s,a)$$

Generalized Policy Iteration With Monte-Carlo Evaluation



- Now use Q instead of V

- Policy evaluation: Monte-Carlo policy evaluation, $Q = q_\pi$?
- Policy improvement: Greedy policy improvement?

- Does it work now?

Problem of Greedy Policy Improvement

- Initialize policy π
- Repeat:
  - Policy evaluation: compute $Q_{\pi_i}(s,a)\ \forall s,a$
  - Policy improvement: greedy update π given Q_{π_i}

$$\pi'(s) = argmax_{a\in A}\, Q(s,a)$$

- Greedy method for policy improvement -> deterministic policy
- If π is deterministic, can't compute Q(s,a) for any $a \neq \pi(s)$
  - May need to modify policy evaluation
  - Policy improvement is using an *estimated* Q

- How to interleave policy evaluation and improvement?

Example of Greedy Action Selection

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

https://www.davidsilver.uk/teaching/

- There are two doors in front of you.
- You open the left door and get reward 0
  V(left) = 0
- You open the right door and get reward +1
  V(right) = +1
- You open the right door and get reward +3
  V(right) = +2
- You open the right door and get reward +2
  V(right) = +2

...

- Are you sure you've chosen the best door?

ε-greedy Exploration

- If π is a deterministic policy, then in following π one will observe returns only for one of the actions from each state
  - many state–action pairs may never be visited.

- Need to try all (s, a) pairs but then follow π
- Want to ensure resulting estimate Q_π is good enough so that policy improvement is a monotonic operator

- ε-greedy exploration: Simple idea to balance exploration and exploitation
- All *m* actions are tried with non-zero probability
  - With probability *1-* ϵ, choose the greedy action
  - With probability ϵ, choose an action at random

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

ε-greedy Policy Improvement

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

$$q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s)\, q_\pi(s,a) = \frac{\epsilon}{m}\sum_a q_\pi(s,a) + (1-\epsilon)\max_a q_\pi(s,a)\,\frac{1-\epsilon}{1-\epsilon}$$

- Since

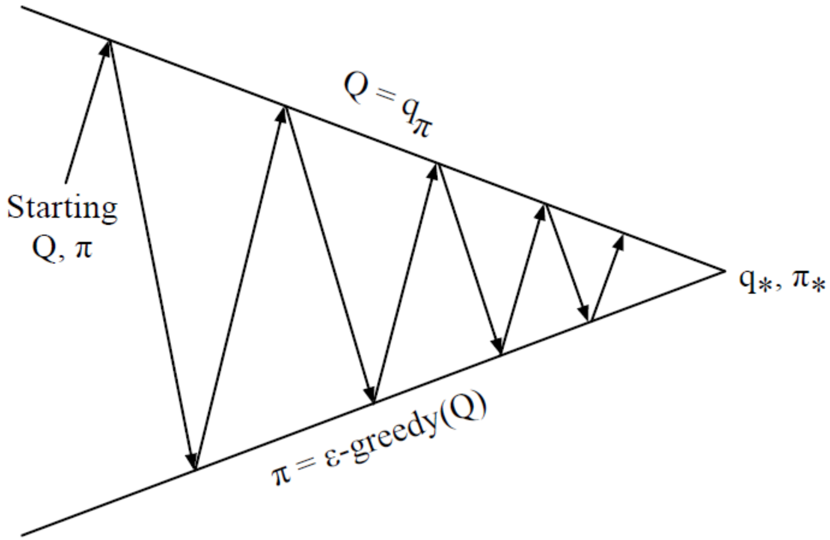
$$\sum_a (\pi(a|s) - \epsilon/m) = \sum_{a \neq a_*} (\pi(a|s) - \epsilon/m) + \sum_{a = a_*} (\pi(a|s) - \epsilon/m) = (0) + (1 - \epsilon + \frac{\epsilon}{m} - \frac{\epsilon}{m}) = 1 - \epsilon$$

- Therefore,

$$q_\pi\big(s, \pi'(s)\big) = \frac{\epsilon}{m}\sum_a q_\pi(s,a) + (1-\epsilon)\max_a q_\pi(s,a)\,\frac{\sum_a(\pi(a|s) - \epsilon/m)}{1-\epsilon}$$
$$\geq \frac{\epsilon}{m}\sum_a q_\pi(s,a) + (1-\epsilon)\sum_a \frac{\pi(a|s) - \epsilon/m}{1-\epsilon}\, q_\pi(s,a)$$
$$= \sum_a \pi(a|s)\, q_\pi(s,a) = v_\pi(s)$$

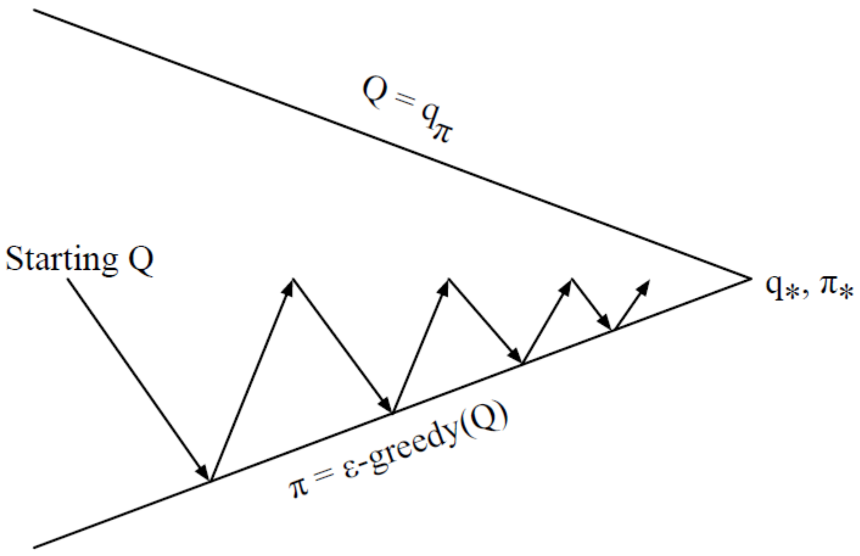
- Since $v_{\pi'}(s) \geq q_\pi\big(s, \pi'(s)\big)$, $v_{\pi'}(s) \geq v_\pi(s)$

Monte-Carlo Policy Iteration



- Now use ϵ-greedy policy improvement instead of greedy improvement
  - Policy evaluation: Monte-Carlo policy evaluation, $Q = q_\pi$
    - (Compute q values until it converge)
  - Policy improvement: ϵ-greedy policy improvement
- This works.

Monte-Carlo Control



- We can do better (more efficiently).
- For every episode:
  - Policy evaluation: Monte-Carlo policy evaluation, $Q \approx q_\pi$
    - No need to compute entire q values
  - Policy improvement: ϵ-greedy policy improvement

Greedy in the Limit with Infinite Exploration (GLIE)

- Greedy in the limit with infinite exploration (GLIE)
- A learning policy π is called GLIE if it satisfies the following two properties:
  - 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

$$\lim_{k \to \infty} \pi_k(a|s) = 1(a = \underset{a'}{\operatorname{argmax}}\, Q_k(s, a'))$$

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

GLIE Monte Carlo Control

- Sample kth episode using π: $\{S_1, A_1, R_2, \ldots S_T\} \sim \pi$
- 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, $\mathrm{Q}(\mathrm{s}, \mathrm{a}) \rightarrow Q_*(s, a)$

MC vs. TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Incomplete sequences
  - Online

- 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

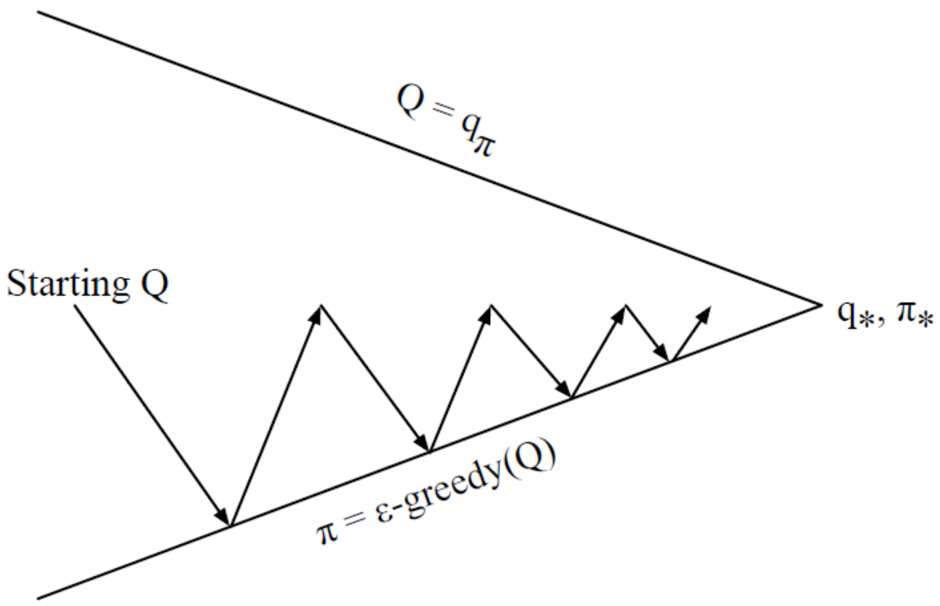
Model-Free Policy Iteration with TD Methods

- Use temporal difference methods for policy evaluation step

- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q_π using temporal difference(TD)
  - Policy improvement: set π to ϵ-greedy (Q_π) (same as Monte Carlo policy improvement)

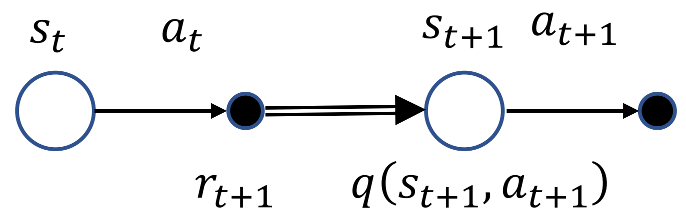
- First consider SARSA, which is an on-policy algorithm.

On-Policy Control With Sarsa



- Instead of MC, use Sarsa & update q at every step (instead of every episode)
- For every time-step:
  - Policy evaluation: Sarsa, $Q \approx q_\pi$
  - Policy improvement: ϵ-greedy policy improvement

Updating Action-Value Functions with Sarsa



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

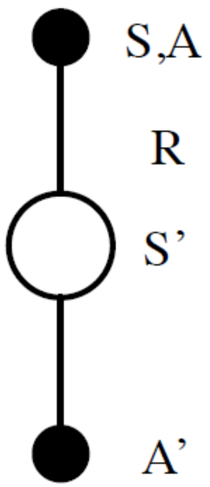
Sarsa Algorithm for On-Policy Control

- For every time-step:
  - Policy evaluation: Sarsa, $Q \approx q_\pi$
  - Policy improvement: ϵ-greedy policy improvement

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

ϵ-greedy policy improvement

police evaluation (Sarsa)



Convergence of Sarsa Algorithm

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

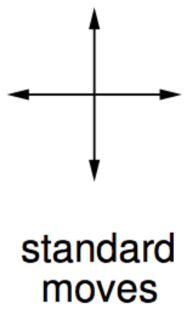
  1) Policy sequence $\pi_t(a|s)$ satisfies the condition of GLIE
  2) The step-sizes α_t satisfy Robbines-Monro sequence such that

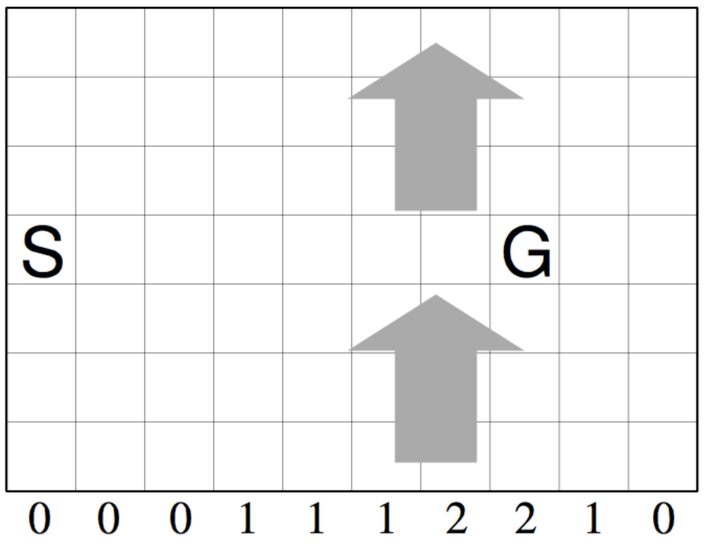
$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

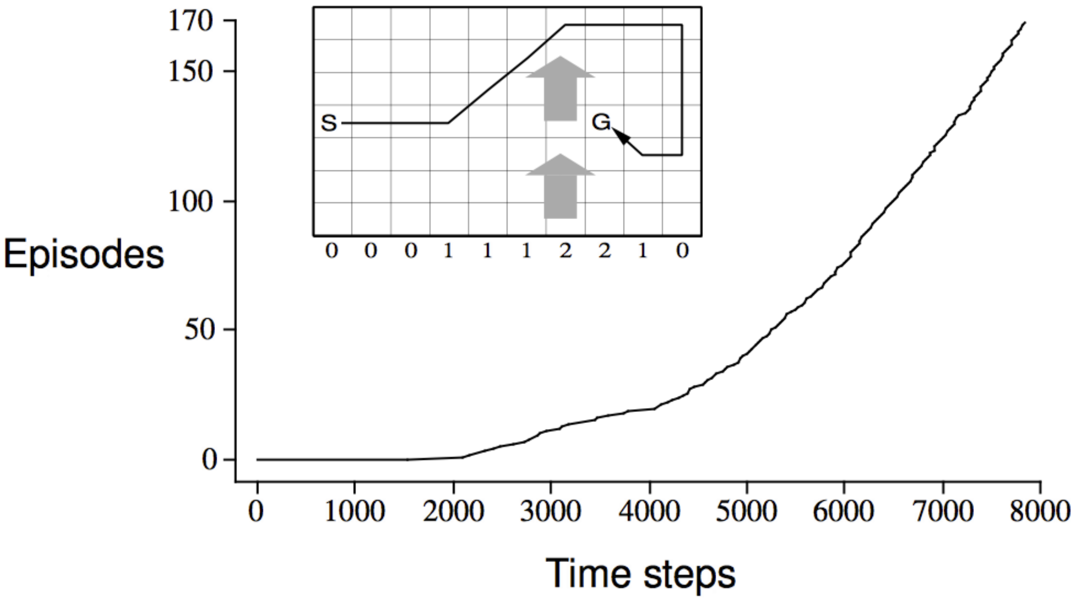
$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

  - For example, $\alpha_t = 1/t$ satisfies the above condition

Windy Gridworld







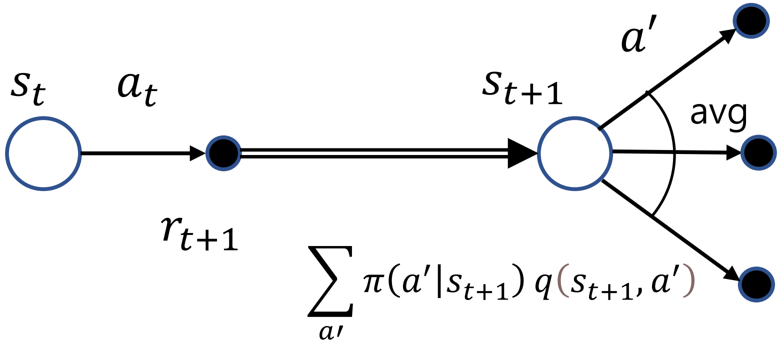
R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2018

- Reward = -1 per time-step until reaching goal
- Undiscounted
- Until t=2000, it couldn't finish episodes, but after that number of finished episodes grows exponentially

Expected Sarsa

- Expected Sarsa is very similar to Sarsa.
- However, instead of state-action values sampled using our current policy, it computes the expected value over all future state-action pairs
- The update-step is now defined as:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \underbrace{\sum_{a'} \pi(a'|s_{t+1})\, q(s_{t+1}, a')}_{\text{average q value over action } a'} - q(s_t, a_t) \right)$$



- Expected Sarsa is an off-policy algorithm

Off-policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$ while following behavior policy $b(a|s)$

$$(S_1, A_1, R_1, \dots S_T) \sim b$$

- 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

Importance Sampling

- Some off-policy methods utilize importance sampling, a general technique for estimating expected values under one distribution given samples from another

- Estimate the expectation of a different distribution

$$E_{X\sim P}[f(X)] = \sum P(X)f(X) = \sum Q(X)\frac{P(X)}{Q(X)}f(X) = E_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]$$

Importance Sampling

- What is the probability of observing a certain trajectory on random variables $a_k, s_{k+1}, a_{k+1}, \dots, s_T$ starting in s_k while following π?

$$P[a_k, s_{k+1}, a_{k+1}, \dots, s_T | s_k, \pi] = \prod_{k}^{T-1} \pi(a_k|s_k) p(s_{k+1}|s_k, a_k)$$

  - Above *p* is the state-transition probability.

- Definition: *Importance sampling ratio*

The relative probability of a trajectory under the target and behavior policy, the importance sampling ratio, from sample step *k* to *T* is:

$$\rho_{k:T} = \frac{\prod_k^{T-1} \pi(a_k|s_k) p(s_{k+1}|s_k, a_k)}{\prod_k^{T-1} b(a_k|s_k) p(s_{k+1}|s_k, a_k)} = \frac{\prod_k^{T-1} \pi(a_k|s_k)}{\prod_k^{T-1} b(a_k|s_k)}$$

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from b (behavior policy) to evaluate π (optimal policy)
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/b} = \frac{\pi(a_t|s_t)\pi(a_{t+1}|s_{t+1})}{b(a_t|s_t)b(a_{t+1}|s_{t+1})} \cdots \frac{\pi(a_T|s_T)}{b(a_T|s_T)} G_t$$

return value if have followed π

$$E_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]$$

- Update value towards corrected return

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^{\pi/b} - V(s_t))$$

- Cannot use if b is zero when π is non-zero
- Importance sampling can dramatically increase variance ($X = G_t^{\pi/b}$)

$$Var[X] = E[X^2] - E[X]^2$$

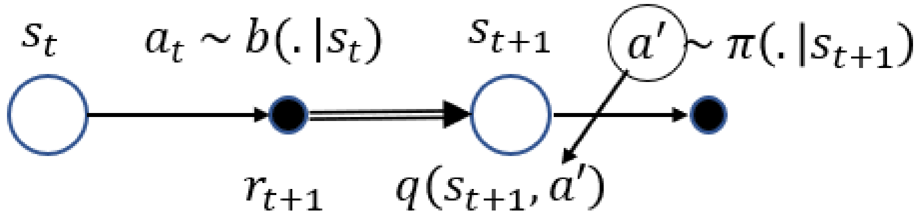
Q Learning

- We now consider off-policy learning of action-values *Q(s,a)*
- No importance sampling is required

- Next action is chosen using behavior policy $a_t \sim b(.|s_t)$
- But we consider alternative successor action $a' \sim \pi(.|s_t)$
- And update $Q(s_t, a_t)$ towards value of alternative successor action

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

- If $a' \sim b(.|s_t)$, becomes Sarsa



Off-Policy Control with Q-Learning

- We now allow both behavior and target policies to improve

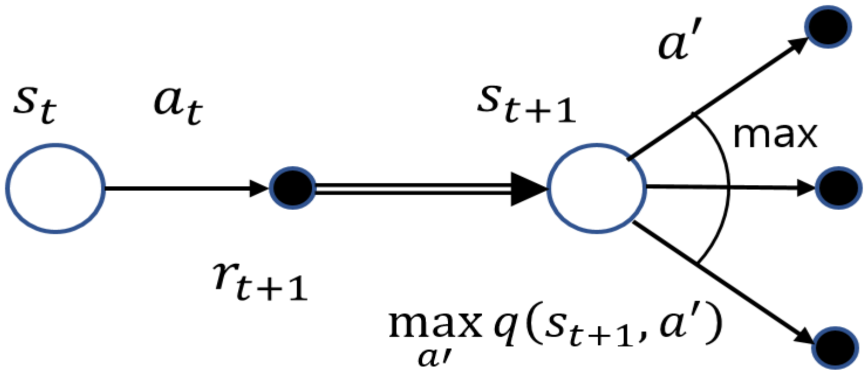
- The behavior policy b is e.g. ϵ-greedy w.r.t. Q(s,a)
- The target policy π is greedy w.r.t. Q(s,a)

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

- The Q-learning target then simplifies

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

Q-Learning Control Algorithm



- Q-learning:

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

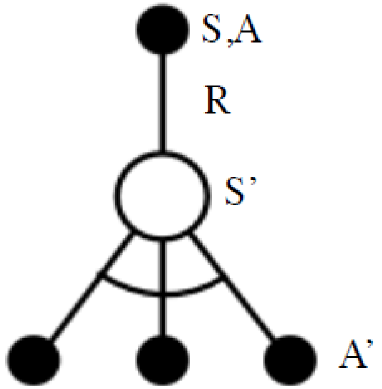
- Recall SARSA

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

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

Q-Learning for Off-Policy Control

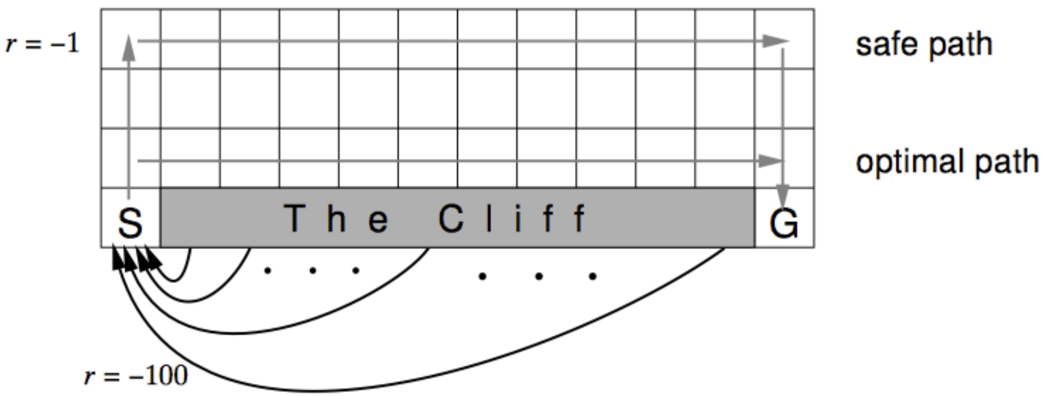
1: Initialize $Q(s,a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
2: Set π_b to be ϵ-greedy w.r.t. Q
3: **loop**
4: Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
5: Observe (r_t, s_{t+1})
6: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
7: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$
9: **end loop**



ϵ-greedy policy improvement

greedy policy

Q-Learning vs Sarsa



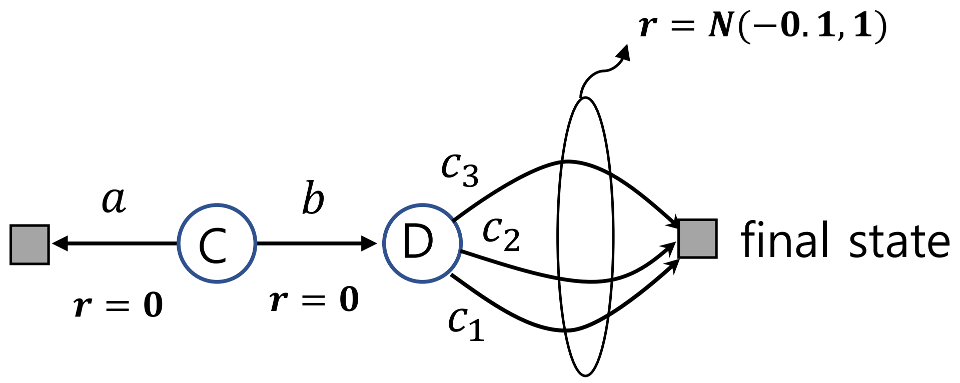
R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2018

- But the reason that SARSA took safest path is because the policy of SARSA is the epsilon greedy where *epsilon percent* of the time the agent took random walk.
- This means it is not safe at all to walk close to the cliff
- To avoid the big punishment of agent falling off the cliff, accept the small punishment of long traveling instead.

Double Q-Learning

- Q-Learning performs poorly in some environments because of overestimation of Q values
- Overestimation caused by $\max_{a'} Q(s_{t+1}, a')$ in the following

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R + \gamma \max_{a'} Q(s_t, a') - Q(s_t, a_t))$$



No	c_1	c_2	c_3	Avg(c_i)	Max(c_i)
1	-0.4	$\mathbf{0.2}$	-0.2	$\mathbf{-0.13}$	$\mathbf{0.2}$
2	$\mathbf{0.1}$	-0.3	0	$\mathbf{-0.06}$	$\mathbf{0.1}$

$$q(C, b) \leftarrow q(C, b) + \alpha(r + \gamma \max_{c_i} q(D, c_i) - q(C, b))$$

$$q(C, b) \leftarrow q(C, b) + \big(0 + \max\{-0.4, 0.2, -0.2\} - q(C, b)\big) = q(C, b) + (0.2 - q(C, b))$$

Double Q-Learning

- The proposed solution is to maintain two Q-value functions q_{A} and q_B, each one gets update from the other for the next state.
- Use q_{A} to find best action a_* (maximum q)

$$a_* = \underset{a}{\operatorname{argmax}}\, q_A(s_{t+1}, a)$$

- Use the action value of a_* from q_B

- **Double Q Learning**

$$q_{\mathrm{A}}(s_t, a_t) \leftarrow q_{\mathrm{A}}(s_t, a_t) + \alpha(r_{t+1} + \gamma q_B(s_{t+1}, a_*) - q_{\mathrm{A}}(s_t, a_t)) \qquad \text{where } a_* = \underset{a}{\operatorname{argmax}}\, q_A(s_{t+1}, a)$$

Double Q-Learning

Algorithm 1 Double Q-learning

Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end