Lecture 5: Model-Free Control

David Silver

Outline

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

next lecture : scale-up, large scale problem

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

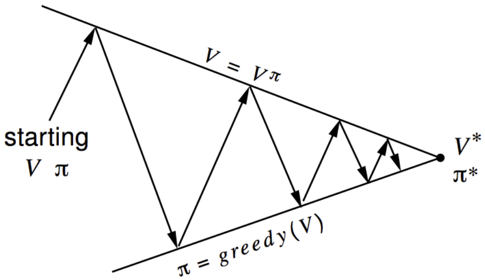
- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

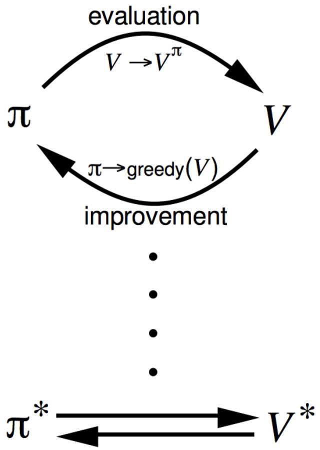
use same policy

different policies

policy for acting and policy for evaluating

Generalised Policy Iteration (Refresher)

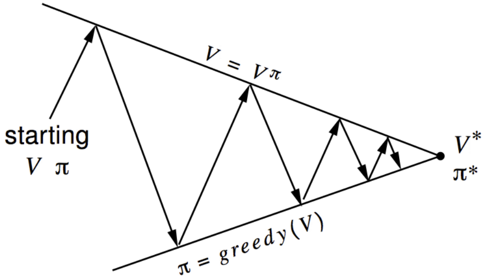




Policy evaluation Estimate v_π
e.g. Iterative policy evaluation

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?

There are two problems

1. learning speed : it may be very slow if the length upto the terminal state is so long.
2. exploration issues : if we act greedily all the time, we don't guarantee the exploration of the entire space.
   we may not explore a state which has better potential.

Model-Free Policy Iteration Using Action-Value Function

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

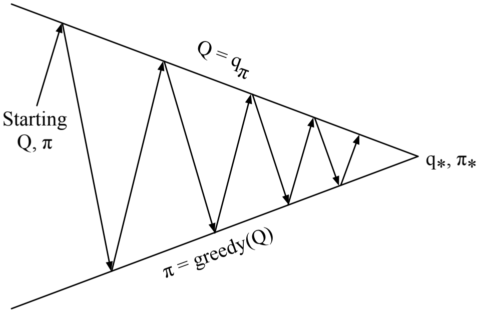
$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

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

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

We use the evaluation of action (action-value function) instead of the evaluation of state (state-value function).

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
  $V(left) = 0$
- 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
  $V(right) = +2$

$\vdots$

- 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

uniformly random

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

ϵ-Greedy Policy Improvement

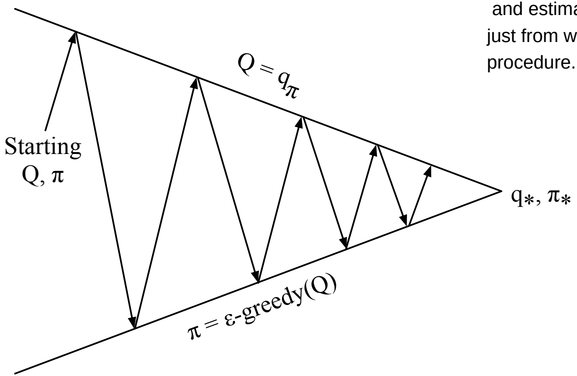
Theorem

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

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

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

Monte-Carlo Policy Iteration

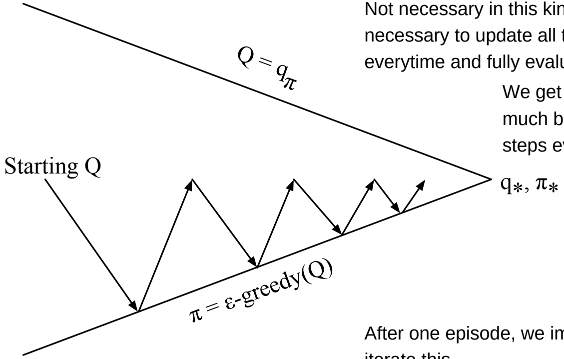


Run some episodes using our current policy and estimate value of all states and all actions just from what we've seen. That's our evaluation procedure.

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

Policy improvement ϵ-greedy policy improvement

Monte-Carlo Control



Let's try to make more efficient. Note that we are solving DP. Not necessary in this kind of policy iteration framework, not necessary to update all the way to the top of the line Q everytime and fully evaluate our policy.

We get the enough information which guide much better policy just after spending few steps evaluating.

After one episode, we immediately improve the policy and iterate this.

Every episode:

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

Policy improvement ϵ-greedy policy improvement

GLIE

Natural question come up with is how we can really guarantee that we find the best possible policy. What we really desire is pi* and we really want to know the best possible behavior in this environment.

So, to do that we have to kind of balance two different things. We need to make sure that we continue exploring without excluded things. One idea how to balance those is following. GLIE idea.

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,

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

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

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

counting how many times we've seen the state-action pair in doing the incremental update of the mean (below).

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

- Improve policy based on new action-value function

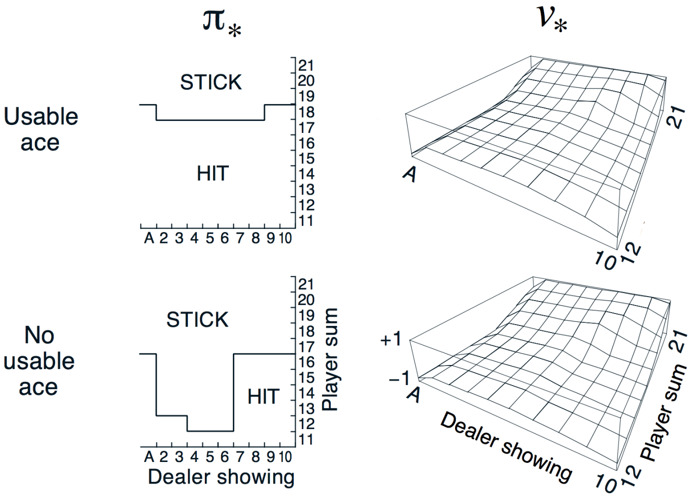
$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

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

Back to the Blackjack Example

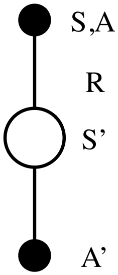
Monte-Carlo Control in Blackjack



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 k=1

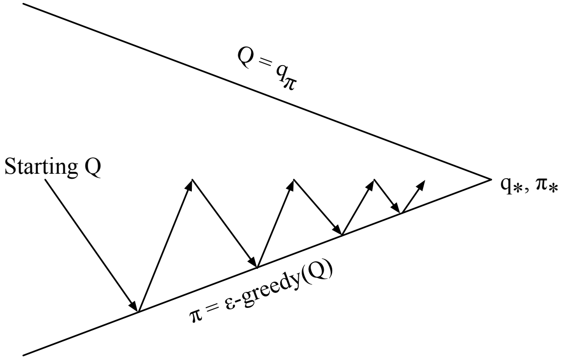
Updating Action-Value Functions with Sarsa



start at specific state and action

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

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

Convergence of Sarsa

Theorem

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

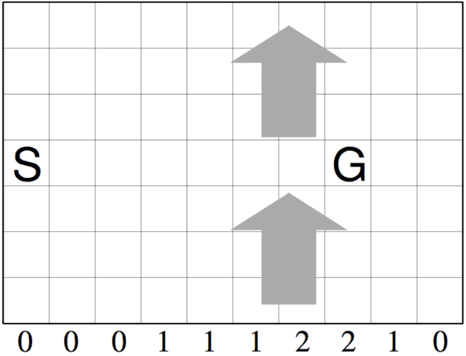
- *GLIE sequence of policies $\pi_t(a|s)$*
- *Robbins-Monro sequence of step-sizes α_t*

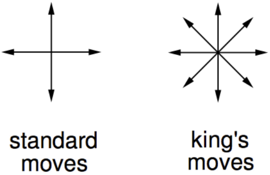
$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

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

In practice, we don't worry about this (the second condition of theorem), sometimes the first condition either. Typically works anyway. That's empirical results. But, this is the theory.

Windy Gridworld Example



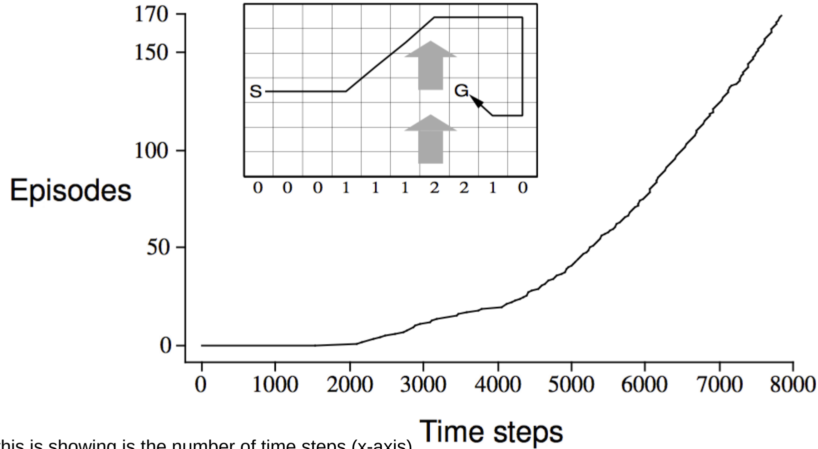


These numbers indicate the moving by wind when a grid move to the next step.

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

Sarsa on the Windy Gridworld

naive version of SARSA



What this is showing is the number of time steps (x-axis)

What we're looking at is how many episodes are completed in that time steps. (y-axis)

n-Step Sarsa

Control the bias-variance trade-off

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

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

never bootstrap from a value function

- Define the n-step Q-return

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

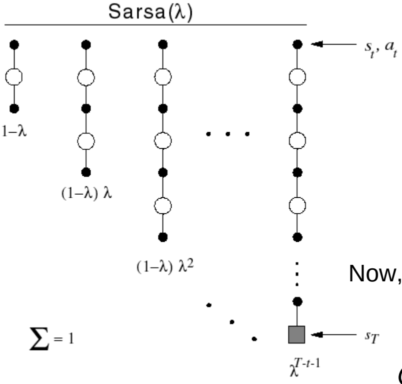
- n-step Sarsa updates $Q(s, a)$ towards the n-step Q-return

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

Just like last lecture, consider the algorithm which is robust choices of n and which is average over many different n.

Forward View Sarsa(λ)

We can control the lambda to see more or less far-sighted way or how much we prefer large n or short n.



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

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

Now, we can plug this return into the update (target).

- Forward-view Sarsa(λ)

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

Then, what's the problem with this approach so far? Every things are reflected to control our problems for previous methods like building the spectrum b/w MC algorithms, TD algorithms, and SARSA algorithm all the way out to future (if lambda = 1, MC, and if lambda = 0, SARSA) and controlling the bias-variance trade-off by averaging all over these things.

The only problem is that we are kind of looking forward in time. That's an online algorithm which update immediate in each step.

Backward View Sarsa(λ)

The idea is very similar the last class.

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

What we do again is to save the state an action which took most recent before we get the goal(terminal).

$$E_0(s, a) = 0$$

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

If a state-action pair is visited and actually we are in the state and action, then we increase eligibility trace by 1. (If I see that particular state-action pair before, increase my trace.) All state-action pair even once we don't visit will gonna decay a little bit. (because of lambda and gamma factors)

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

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

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_t E_t(s, a)$$ credit assignment trace or eligibility trace

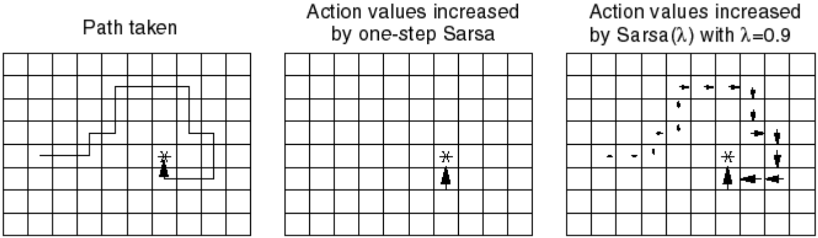
alpha, delta : TD error which is the difference b/w what actually happen

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 $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$ update all state-action pairs
            $E(s,a) \leftarrow \gamma \lambda E(s,a)$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

Sarsa(λ) Gridworld Example

It means that we only get a propagated information by one step per episode Salsa(0)



Lambda parameter determines how quickly and how far that information propagation back through our trajectory

We will build up the eligibility trace all the way along the trajectory, so each of these state-action pairs we visit will have eligibility trace increased

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

3rd point is the best known example why or where the off-policy learning is used.
We know that there is a big issue in RL which is making sure that you explore the state space actively.
At the same time, we want to learn about the optimal behavior which doesn't explore at all.

Importance Sampling

We look two mechanisms to deal with the off-policy learning

- Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X)\\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X)\\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

Radon-Nikodym Theorem

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

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

- Update value towards *corrected* return

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

- Cannot use if μ is zero when π is non-zero
- Importance sampling can dramatically increase variance

We can use this idea, BUT it has extremely high variance. So, it's just useless in practice.
MC learning is really bad idea off-policy.

Importance Sampling for Off-Policy TD

We have to use TD learning when we are working off-policy

- Use TD targets generated from μ to evaluate π
- Weight TD target $R + \gamma V(S')$ by importance sampling
- Only need a single importance sampling correction

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

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Q-Learning

The idea which works best with off-policy learning is Q-learning.
(specific TD(0) or SARSA(0))

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

St : the state we started in & At : action we took

bootstrap from the value of alternative action

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

That's the thing that tells us how much value that we actually got under a target policy.

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. $Q(s, a)$

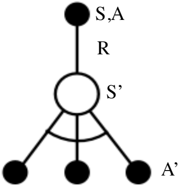
$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')$$

- 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

(SARSAMAX ?)



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

Theorem

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

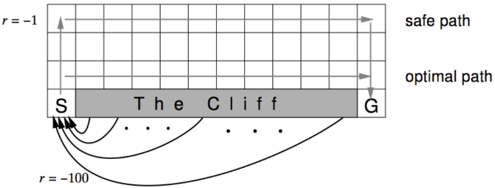
Q-Learning Algorithm for Off-Policy Control

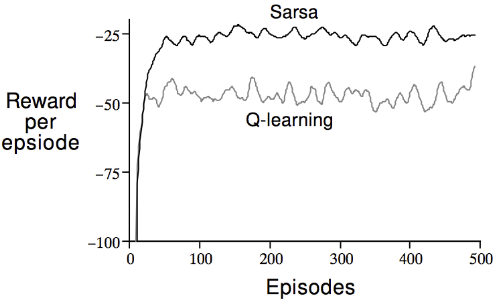
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R,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

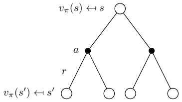
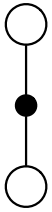
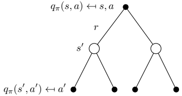
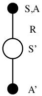
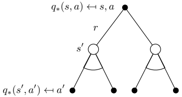
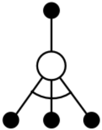
Q-Learning Demo

Cliff Walking Example





Relationship Between DP and TD

	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

Relationship Between DP and TD (2)

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

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

Questions?