Lecture 6: Model-Free Control

Hado van Hasselt

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

Uses of Model-Free Control

Some example problems that can be modelled as MDPs

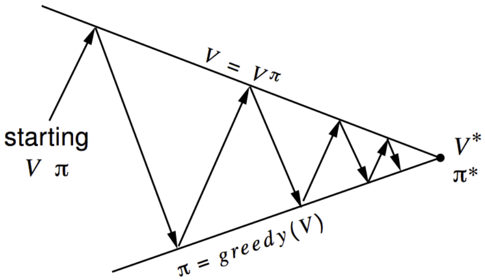
- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Portfolio management
- Protein Folding
- Robot walking
- Atari video games
- Game of Go

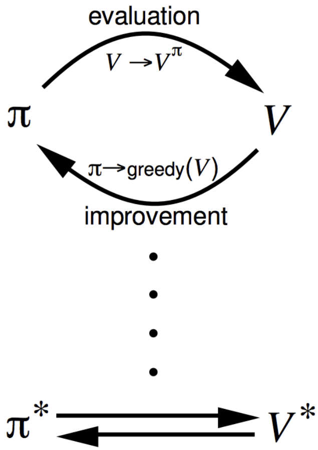
For most of these problems, either:

- 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

Generalized Policy Iteration (Refresher)





Policy evaluation Estimate V^π

e.g. Iterative policy evaluation

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

e.g. Greedy policy improvement

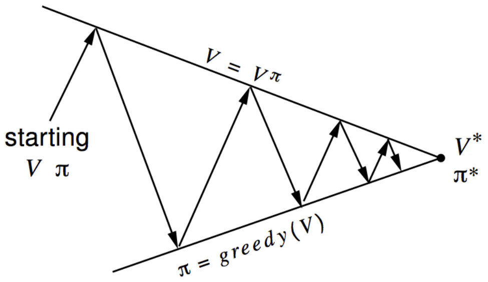
Monte Carlo

- Recall, Monte Carlo estimate from state S_t is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

- $\mathbb{E}[G_t] = V^\pi$
- So, we can multiple estimates to get V^π

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

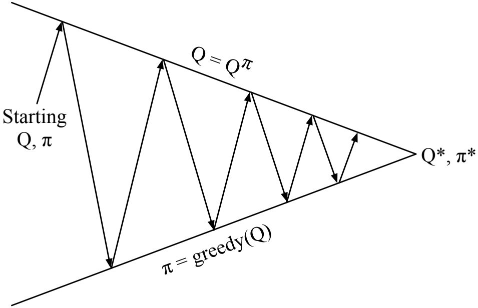
- Greedy policy improvement over $V(s)$ requires model of MDP

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

- 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?

ϵ-Greedy Policy Improvement

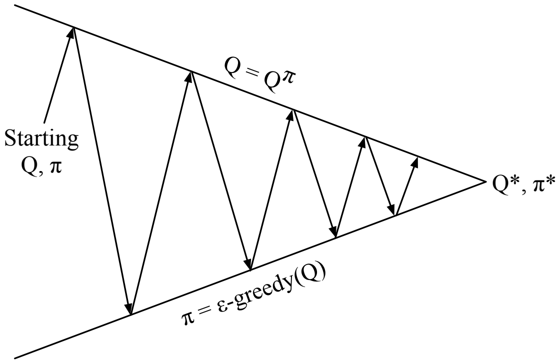
Theorem

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

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

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

Monte-Carlo Policy Iteration

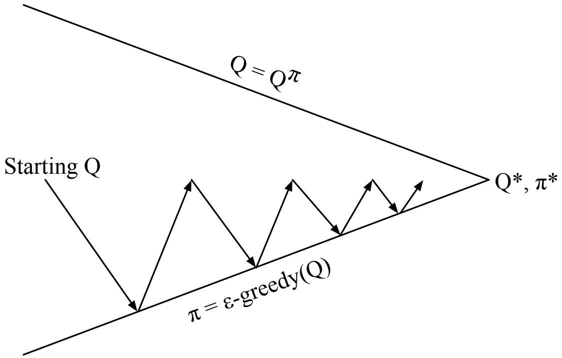


Policy evaluation Monte-Carlo policy evaluation, $Q = Q^\pi$

Policy improvement ϵ-greedy policy improvement

Here π^* is best ϵ-greedy policy

Monte-Carlo Generalized Policy Iteration



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,

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

- The policy converges on a greedy policy,

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

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

GLIE Every-Visit 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

$$\epsilon \leftarrow 1/k$$

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

Theorem

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

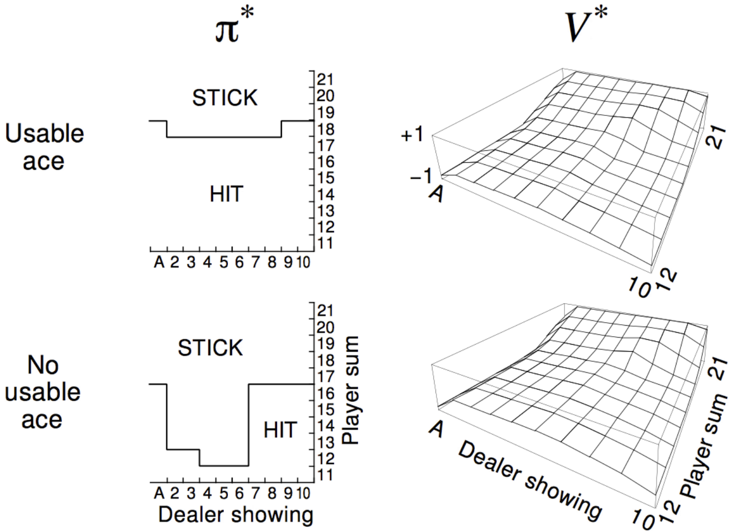
GLIE Every-Visit Monte-Carlo Control

$$
\begin{aligned}
N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left(G_t - Q(S_t, A_t)\right) \\
\epsilon &\leftarrow 1/k \\
\pi &\leftarrow \epsilon\text{-greedy}(Q)
\end{aligned}
$$

- Any practical issues with this algorithm?
- Any ways to improve, in practice?
- Is $1/N$ the best step size?
- Is $1/k$ enough exploration?

Back to the Blackjack Example

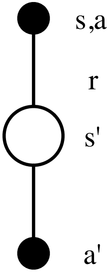
Monte-Carlo Control in Blackjack



MC vs. TD Control

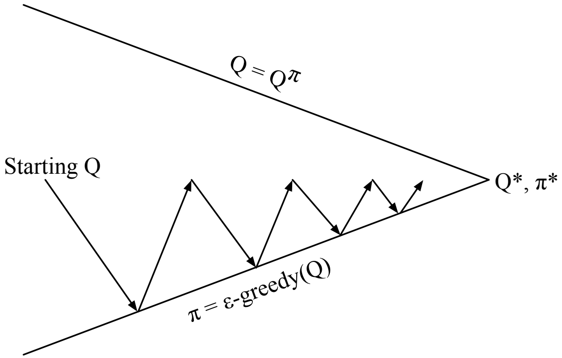
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Can learn from incomplete sequences
- Natural idea: use TD instead of MC for control
  - Apply TD to $Q(s, a)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

Updating Action-Value Functions with Sarsa



$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R + \gamma Q(s',a') - Q(s,a) \right)$$

Sarsa



Every time-step:

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

Policy improvement ϵ-greedy policy improvement

Sarsa

Initialize $Q(s, a)$ arbitrarily
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:

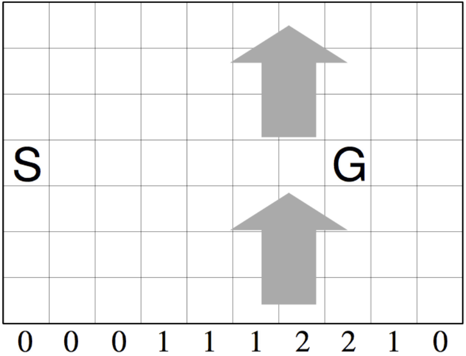
- *GLIE sequence of policies $\pi_t(s, a)$*
- *Robbins-Monro sequence of step-sizes α_t*

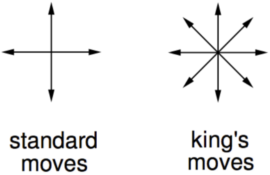
$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

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

E.g., $\alpha_t = 1/t$ or $\alpha_t = 1/t^{\omega}$ with $\omega \in (0.5, 1)$.

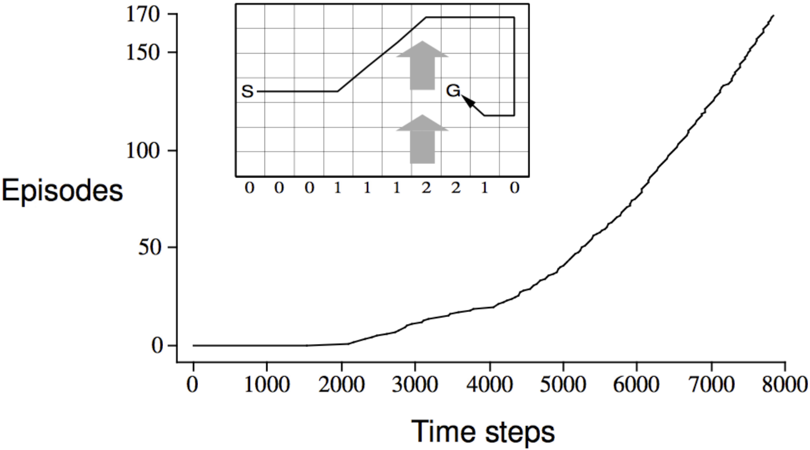
Windy Gridworld Example





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

Sarsa on the Windy Gridworld



n-Step Sarsa

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

$$\begin{array}{lll} n = 1 & \textit{Sarsa}(0) & G_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\ n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\ \vdots & & \vdots \\ n = \infty & MC & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T \end{array}$$

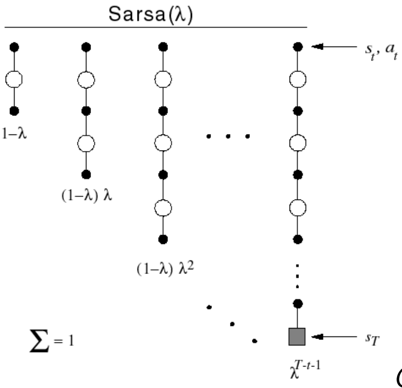
- Define the n-step Q-return

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

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

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

Forward View Sarsa(λ)



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

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

- Forward-view Sarsa(λ)

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

Forward View Sarsa(λ)

- Written recursively:

$$G_t^\lambda = R_{t+1} + \gamma(1-\lambda)Q_t(S_{t+1}, A_{t+1}) + \gamma\lambda G_{t+1}^\lambda$$

- Weight 1 on R_{t+1}
- Weight $\gamma(1-\lambda)$ on $Q_t(S_{t+1}, A_{t+1})$
- Weight $\gamma\lambda$ on R_{t+2}
- Weight $\gamma^2\lambda(1-\lambda)$ on $Q_t(S_{t+2}, A_{t+2})$
- Weight $\gamma^2\lambda^2$ on R_{t+3}
- Weight $\gamma^3\lambda^2(1-\lambda)$ on $Q_t(S_{t+3}, A_{t+3})$
- ...

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

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

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

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

- NB: I rolled step size into the traces on this slide

Sarsa(λ) Algorithm

Select A_0 according to $\pi_0(S_0)$

Initialize $E_0(s, a) = 0$, for all s, a

For $t = 0, 1, 2, \ldots$

Take action A_t, observe R_{t+1}, S_{t+1}

Select A_{t+1} according to $\pi_{t+1}(S_{t+1})$

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

$E_t(S_t, A_t) = E_{t-1}(S_t, A_t) + 1$ (accumulating trace)

or $E_t(S_t, A_t) = 1$ (replacing trace)

or $E_t(S_t, A_t) = E_{t-1}(S_t, A_t) + \alpha_t(1 - E_{t-1}(s, a))$ (dutch trace)

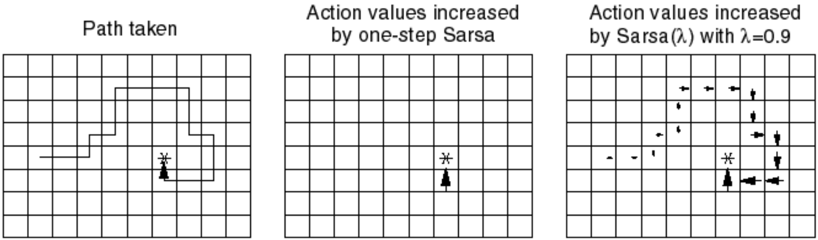
For all s, a:

$Q_{t+1}(s, a) = Q_t(s, a) + \alpha_t \delta_t E_t(s, a)$ (for accum./replac. trace)

or $Q_{t+1}(s, a) = Q_t(s, a) + \delta_t E_t(s, a)$ (for dutch trace)

$E_t(s, a) = \gamma_t \lambda_t E_{t-1}(s, a)$

Sarsa(λ) Gridworld Example



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

Off-Policy Learning

- Evaluate target policy $\pi(s, a)$ to compute $V^{\pi}(s)$ or $Q^{\pi}(s, a)$
- While following behaviour policy $\mu(s, a)$

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

- 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

Importance Sampling

- Estimate the expectation of a different distribution

$$
\begin{aligned}
\mathbb{E}_{x \sim d}[f(x)] &= \sum d(x) f(x) \\
&= \sum d'(x) \frac{d(x)}{d'(x)} f(x) \\
&= \mathbb{E}_{x \sim d'}\left[\frac{d(x)}{d'(x)} f(x)\right]
\end{aligned}
$$

Importance Sampling for Off-Policy Monte-Carlo

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

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

- Update value towards *corrected* return

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

- Importance sampling can dramatically increase variance

Importance Sampling for Off-Policy TD

- 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

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

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

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required
- Next action may be chosen using behaviour policy $A_{t+1} \sim \mu(S_t, \cdot)$
- But we consider probabilities under $\pi(S_t, \cdot)$
- Update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

- Called Expected Sarsa (when $\mu = \pi$) or Generalized Q-learning

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:

$$\begin{aligned} & R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) \\ =& R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\ Q(S_{t+1}, a)) \\ =& R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \end{aligned}$$

Q-Learning Control Algorithm

Theorem

Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow Q^*(s, a)$*, as long as we take each action in each state infinitely often.*

Note: no need for greedy behaviour!

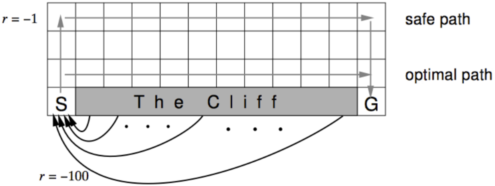
Q-Learning Algorithm for Off-Policy Control

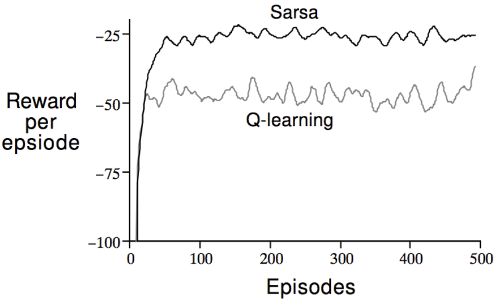
For $t = 0, 1, 2, \ldots$

Take action A_t according to $\pi_t(S_t)$, observe R_{t+1}, S_{t+1}

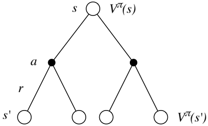
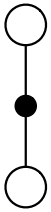
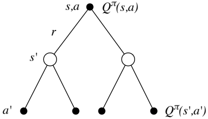
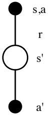
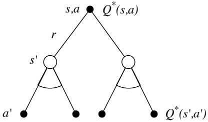
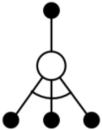
$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma_t \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t) \right)$$

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
$V(s) \leftarrow \mathbb{E}\left[r + \gamma V(s') \mid s\right]$	$V(s) \xleftarrow{\alpha} r + \gamma V(s')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[r + \gamma Q(s',a') \mid s,a\right]$	$Q(s,a) \xleftarrow{\alpha} r + \gamma Q(s',a')$
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 $x \xleftarrow{\alpha} y \quad \equiv \quad x \leftarrow x + \alpha(y - x)$

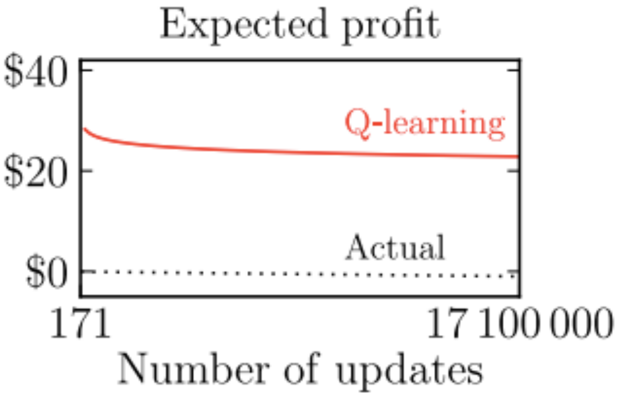
Q-learning overestimates

- Q-learning has a problem
- Recall

$$\max_a Q_t(S_{t+1}, a) = Q_t(S_{t+1}, \operatorname{argmax}_a Q_t(S_{t+1}, a))$$

- Q-learning uses same values to select and to evaluate
- ... but values are approximate
- Therefore:
  - more likely to select overestimated values
  - less likely to select underestimated values
- This causes upward bias

Q-learning overestimates



Double Q-learning

- Q-learning uses same values to select and to evaluate

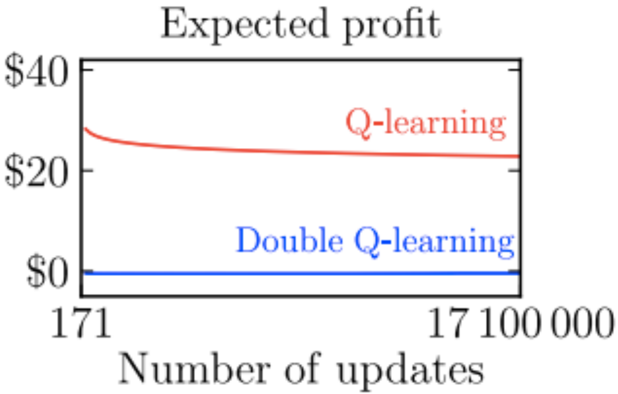
$$R_{t+1} + \gamma Q_t(S_{t+1}, \underset{a}{\text{argmax}}\ Q_t(S_{t+1}, a))$$

- Solution: decouple selection from evaluation
- Double Q-learning:

$$R_{t+1} + \gamma Q'_t(S_{t+1}, \underset{a}{\text{argmax}}\ Q_t(S_{t+1}, a))$$
$$R_{t+1} + \gamma Q_t(S_{t+1}, \underset{a}{\text{argmax}}\ Q'_t(S_{t+1}, a))$$

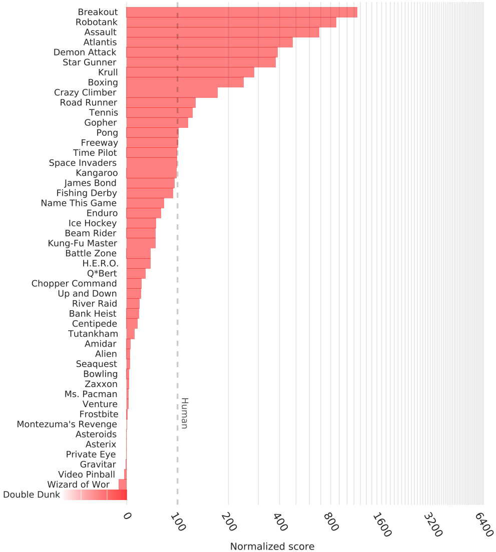
- Then update one at random for each experience

Double Q-learning



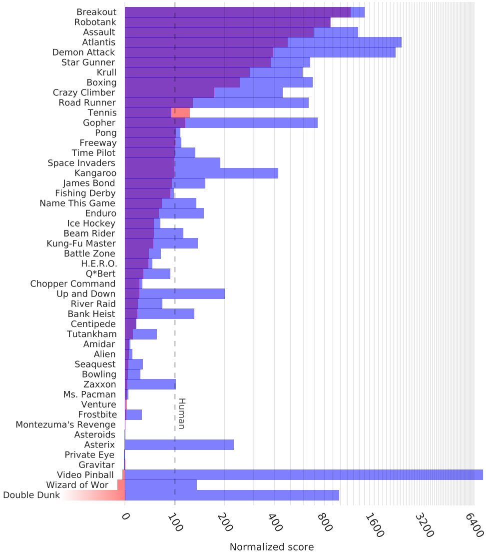
Double DQN on Atari

DQN



Double DQN on Atari

DQN
Double DQN



Questions?