Deep Reinforcement Learning

8 - Model-Free Control – Off-Policy

Prof. Dr. Malte Schilling

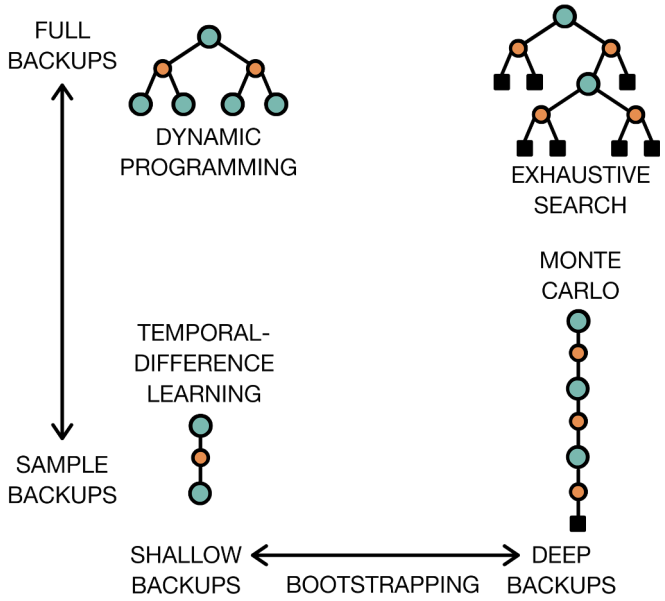
Autonomous Intelligent Systems Group

Overview Lecture

Model-free control

Optimise policy using GPI

- Monte-Carlo (full trajectories)
- Temporal Difference Learning – using TD estimates for bootstrapping



Distinguish

- On-Policy Approach: Improving directly the policy
- Off-Policy Approach: Use a behavior policy for exploration

(Arulkumaran u. a. 2017)

Recap – Model-Free Policy Evaluation Approaches

Iterative approximation of value function for given policy π:

$$v_{n+1}(S_t) = v_n(S_t) + \alpha\Big(G_t - v_n(S_t)\Big)$$

Different Methods:

Approach	Target computation
Monte-Carlo	$G_t^{\text{MC}} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$
TD(0)	$G_t^{(1)} = R_{t+1} + \gamma v_t(S_{t+1})$
n-step TD	$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n v_t(S_{t+n})$
TD(λ)	$G_t^{(\lambda)} = R_{t+1} + \lambda\Big((1-\gamma) v_t(S_{t+1}) + \lambda G_{t+1}^{\lambda}\Big)$

Recap – Policy Iteration (Control)

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Policy evaluation: $\xrightarrow{\text{E}}$

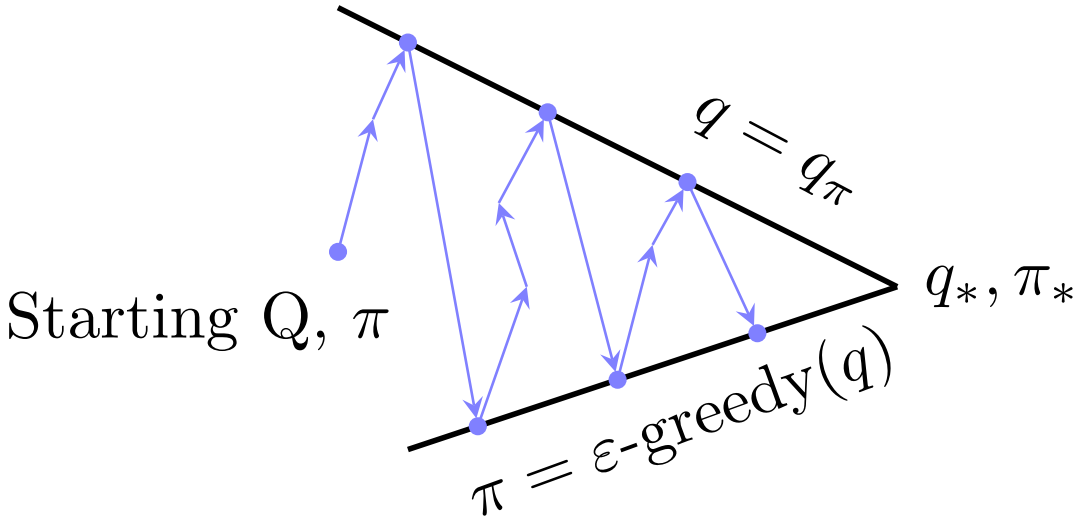
Policy improvement $\xrightarrow{\text{I}}$

For deterministic policies: each policy is guaranteed to be strictly better until we reach the optimal policy.

For finite MDP: $\exists$ only a finite number of deterministic policies; therefore this converges to an optimal policy and an optimal value function in a finite number of iterations.

Recap – Monte-Carlo Policy Improvement

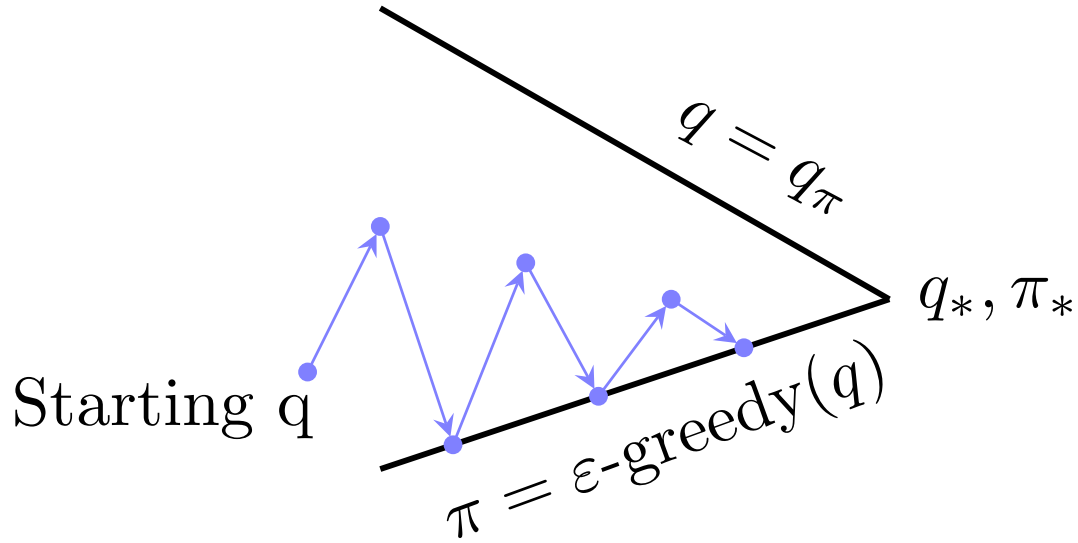
Monte-Carlo Policy Iteration



Policy evaluation: MC policy evaluation, $q = q_\pi$

Policy improvement: ε-greedy policy improvement

Monte-Carlo Control



Every episode:

MC policy evaluation, $q \approx q_\pi$

ε-greedy policy improvement

ε-Greedy Exploration

Simplest idea for ensuring continual exploration: Continue to sample randomly (for a small fraction).

- All m actions are tried with non-zero probability
- With probability $1 - \varepsilon$ choose the greedy action
- With probability ε choose an action at random

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{m+1-\varepsilon} & \text{if } a^* = \arg\max_{a \in \mathcal{A}} q(s,a) \\ \frac{\varepsilon}{m} & \text{otherwise} \end{cases}$$

(Silver 2015)

On-Policy Characteristics

The policy ...

- is generally *soft*: $\pi(a|s) > 0, \forall s \in \mathcal{S}$ and $a \in \mathcal{A}$,
- gradually shifts closer and closer to a deterministic optimal policy.

We can use an ε-greedy policy.

ε-soft policy

A policy, for which

$$\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}, \forall \text{ states and actions for some } \varepsilon > 0$$

Among ε-soft policies: ε-greedy policies are closest to greedy.

Overall: Idea of on-policy Monte Carlo methods is General Policy Improvement.

Convergence: GLIE

Greedy in the Limit with Infinite Exploration (GLIE)

All state-action pairs are explored infinitely many times,

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

The policy converges on a greedy policy,

$$\lim_{t \to \infty} \pi_t(a|s) = \mathbb{1}\left(\arg\max_{a' \in \mathcal{A}} Q_t(s, a') \right)$$

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

GLIE Monte-Carlo Control

Sample $k-th$ 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:

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

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

Improve policy based on new action-value function

$$\varepsilon \rightarrow \frac{1}{k}, \pi \rightarrow \varepsilon\text{-greedy}(q)$$

(Silver 2015)

Convergence GLIE

GLIE Model-free control converges to the optimal action-value function, $q_t(s, a) \rightarrow q_*(s, a)$.

Model-free Control – Temporal Difference Learning

Advantages and Disadvantages of MC vs. TD

Monte-Carlo

Update value $v(S_t)$ towards **actual** return G_t:

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))$$

TD Learning

Update value $v(S_t)$ towards **estimated** return $R_{t+1} + \gamma v(S_{t+1})$:

$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_t + 1) - v(S_t))$$

- MC only works for episodic environments and needs full episodes
- TD is independent of the temporal span of the prediction and can learn from single transitions
- TD needs reasonable value estimates

MC vs. TD Control

TD-learning has advantages over Monte-Carlo in prediction

- Lower variance
- Online
- Can use incomplete sequences

Natural Idea

Use TD instead of MC in our control loop

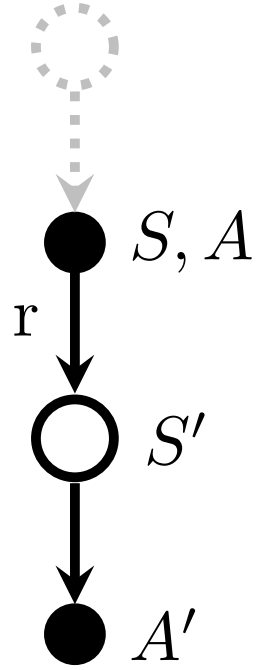
- Apply TD to $q(S, A)$
- Use ε-greedy policy improvement
- Update every time-step

(Hado van Hasselt und Borsa 2021)

SARSA - for update of the Action-Value Function

In every time-step:

- Policy evaluation: SARSA, $q(s, a) \approx q_\pi(s, a)$
- Policy improvement: ε-greedy policy improvement step



$$q'(s, a) \leftarrow q(s, a) + \alpha\Big(r + \gamma q(S', A') - q(S, A)\Big)$$

SARSA – On-Policy TD Control

Initialize $Q(s,a)$ for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\texttt{terminal},\cdot) = 0$.

for *each episode* **do**

  Initialize S

  Choose A from S using policy derived from Q (e.g. ϵ-greedy)

  for *each step in the episode, until state S is terminal* **do**

    Take action A, observe R, S'

    Choose A' from S' using policy derived from Q (e.g. ϵ-greedy)

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

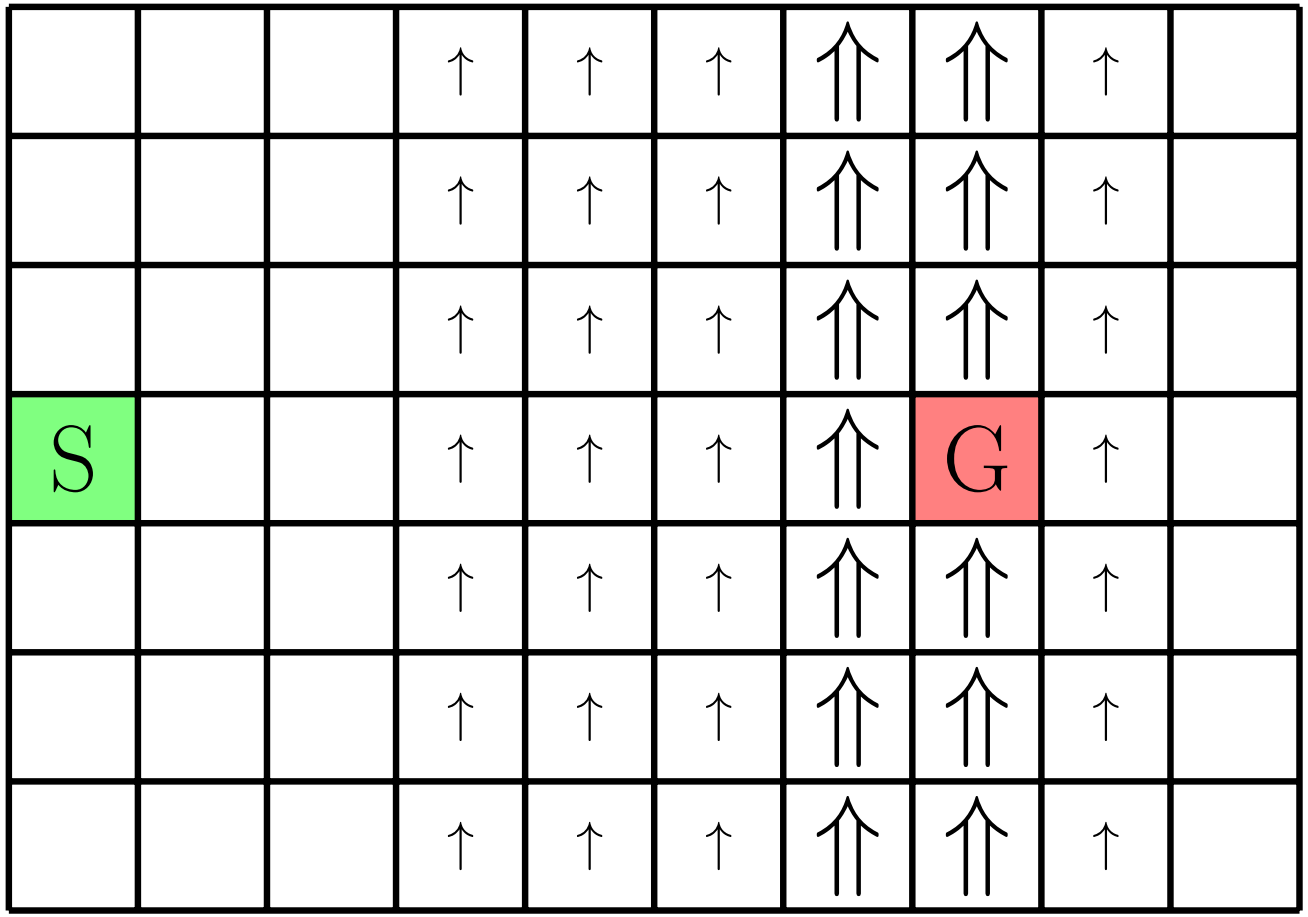
    $S \leftarrow S'$; $A \leftarrow A'$;

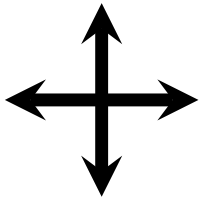
  end

end

Agent part of the algorithm; Environment interaction

Example for TD-Learning: Windy Grid World





actions

Reward:
−1 for each step

(Sutton und Barto 2018)

Example - Explanation: Windy Grid World

Goal: An agent should find a route to travel from the start point to the goal point.

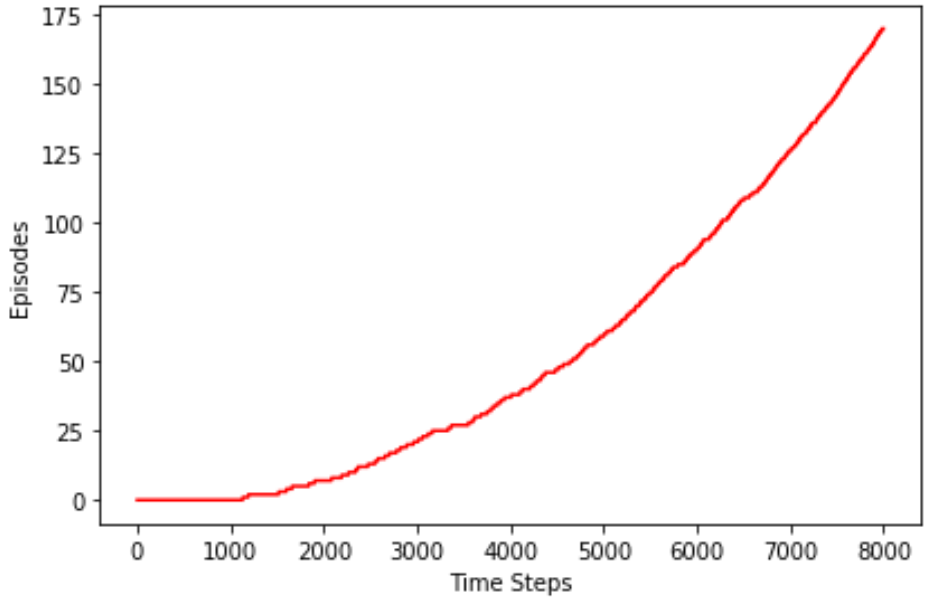
Environment:

- 10×7 grid environment
- actions are movements in the four main directions
- crosswind (upwards) $\rightarrow$ shifts agent one (or two) grid further to the top

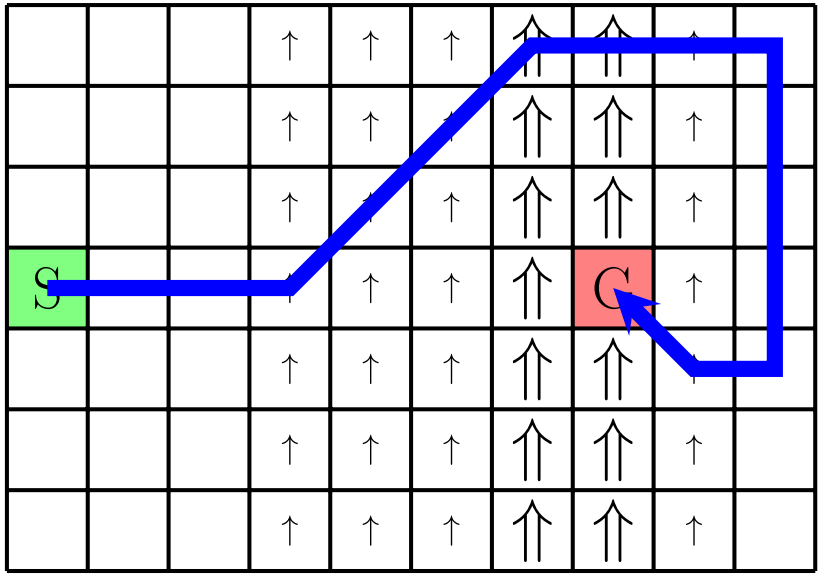
Reward: Aiming for reaching goal as fast as possible, -1 for each timestep in environment.

Example Results for Windy Grid World via SARSA

Learning optimal path



Learned Policy



$\varepsilon = 0.1, \alpha = 0.5,$
initially $q(S, A) = 0, \forall S, A.$

Policy Improvement Example

You can step **policy evaluation** and **policy improvement** in this interactive grid environment.

Here shown for a form of TD Learning.

(Karpathy 2015)

Summary SARSA

Updating Action-Value Function with SARSA

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)\Big)$$

Convergence

Tabular SARSA converges to the optimal action-value function, $q(s, a) \rightarrow q_*(s, a)$, if the policy is GLIE (Greedy in the Limit with Infinite Exploration)

Model-free Control – Off-Policy Approaches

On and Off-Policy Learning

On-policy learning

- Learn about behaviour policy π from experience sampled from π

Off-policy learning

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

(Hado van Hasselt und Borsa 2021)

Off-Policy Learning

Evaluate target policy $\pi(a|s)$ to compute $q_\pi(s, a)$ while following behaviour policy $b(a|s)$:

$$S_1, A_1, R_2, \ldots, S_T \sim b$$

Why is this important?

- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
- Learn about optimal policy while following exploratory policy
- Learn about multiple policies while following a single behavioral policy

(Silver 2015)

Q-Learning Control Approach

Q-Learning estimtaes the value of the greedy policy

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \Big(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t) \Big)$$

When always acting greedy, the policy might not explore sufficiently.

Convergence Q-Learning

Q-learning control converges to the optimal action-value function, $q \rightarrow q_*$, as long as each action in each state is infinitely often selected (by a behavioral policy).

This works for any behavioral policy that selects all actions sufficiently often (requires appropriately decaying step sizes, e.g., $\alpha = \frac{1}{t^\omega}$ with $\omega \in [0.5, 1]$, and in general it must hold: $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$)

Q-Learning for Off-Policy TD Control

Input: Step size $\alpha \in (0, 1]$, and a small $\epsilon > 0$.
Initialize $Q(s, a)$ for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\texttt{terminal}, \cdot) = 0$.
for *each episode* **do**
  Initialize S
  // SARSA:Choose A from S using policy derived from Q
  for *each step of the episode, until* S *is terminal* **do**
    Choose A from S using policy derived from Q (e.g. ϵ-greedy)
    Take action A, observe R, S'
    // Key difference to Sarsa in next step, was $Q(S', A')$
    // In Q-Learning: Like using a greedy policy from S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$ // SARSA was also doing $A \leftarrow A'$
  end
end

Agent part as in SARSA; ; Q-Learning specific part of the algorithm; Environment interaction

Difference bw. SARSA (on-policy) and Q-Learning (off-policy)

Q-learning is using a different policy for choosing next action A' and updating q.

It is evaluating π as proposed by a different greedy policy b which makes it an off-policy algorithm (for selection which action to use for estimate still uses q_π).

SARSA uses π all the time and is an on-policy algorithm.

	SARSA	Q-learning
Choosing A'	π	b
Updating Q	π	π

In SARSA: π is an ε-greedy policy ($\varepsilon > 0$ with exploration).

In Q-Learning: π is the greedy target policy ($\varepsilon = 0$, NO exploration), but b is, e.g., ε-greedy.

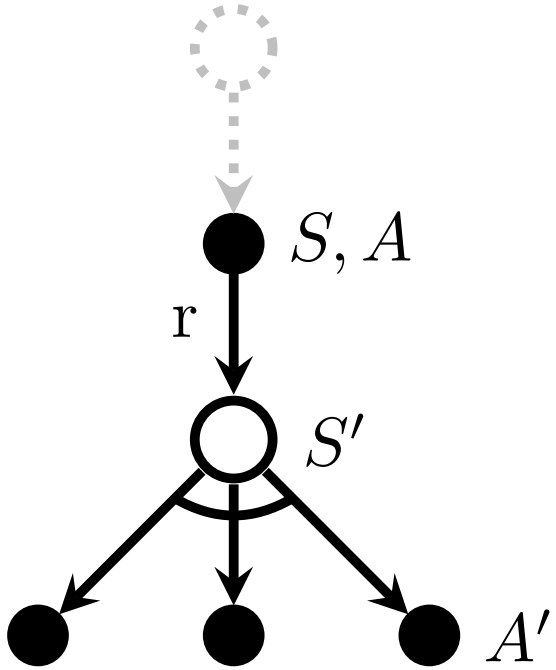
Q-Learning

In every time-step:

- Policy evaluation:

$$q(s, a) \approx q_\pi(s, a)$$

- Policy improvement: ε-greedy policy improvement step

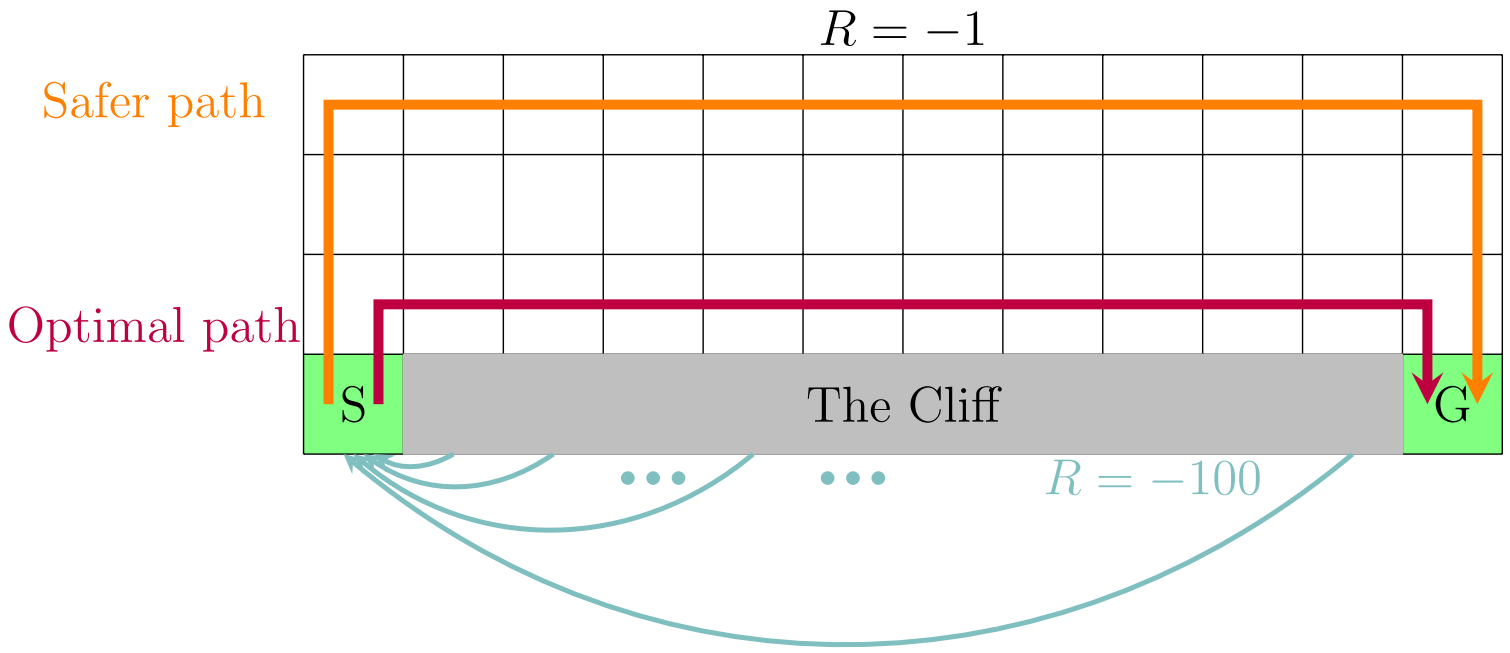


$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \Big(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t) \Big)$$

Cliff Walking Example

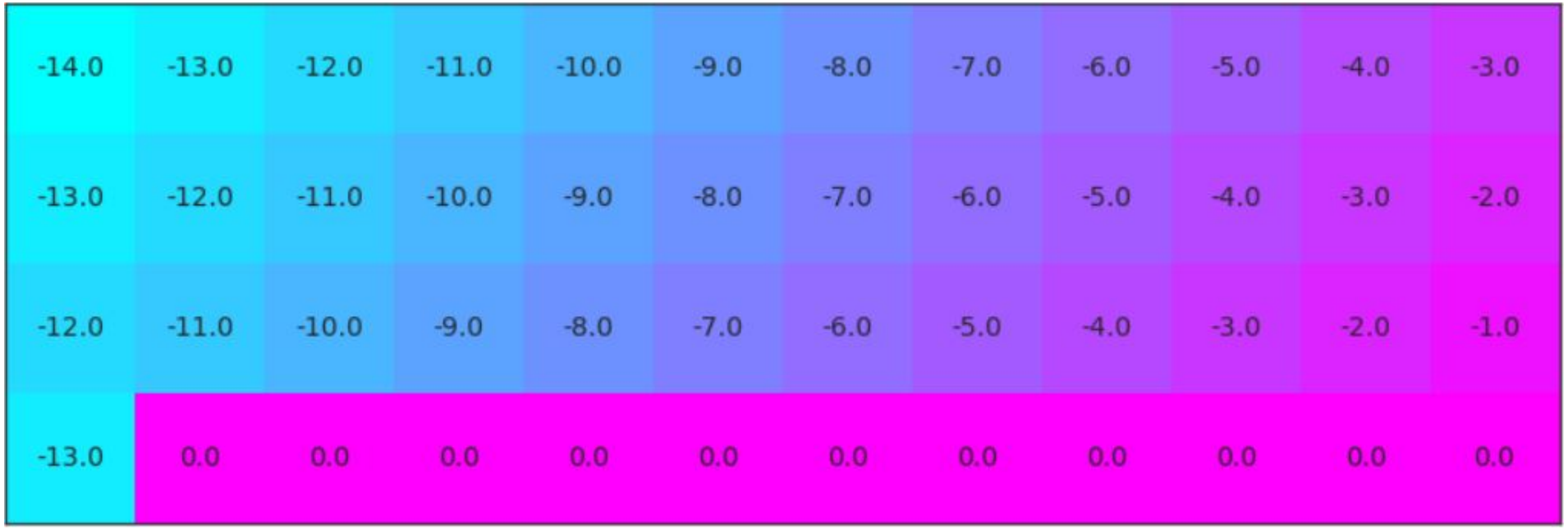
A standard undiscounted episodic task:

- **Action Space:** four actions (up, down, left, right)
- **Reward:** $-1 \; \forall$ transitions, except into region cliff, there $r = -100$ and reset



(Sutton und Barto 2018)

Cliff Walking – Optimal Value Function



Cliff Walking Results

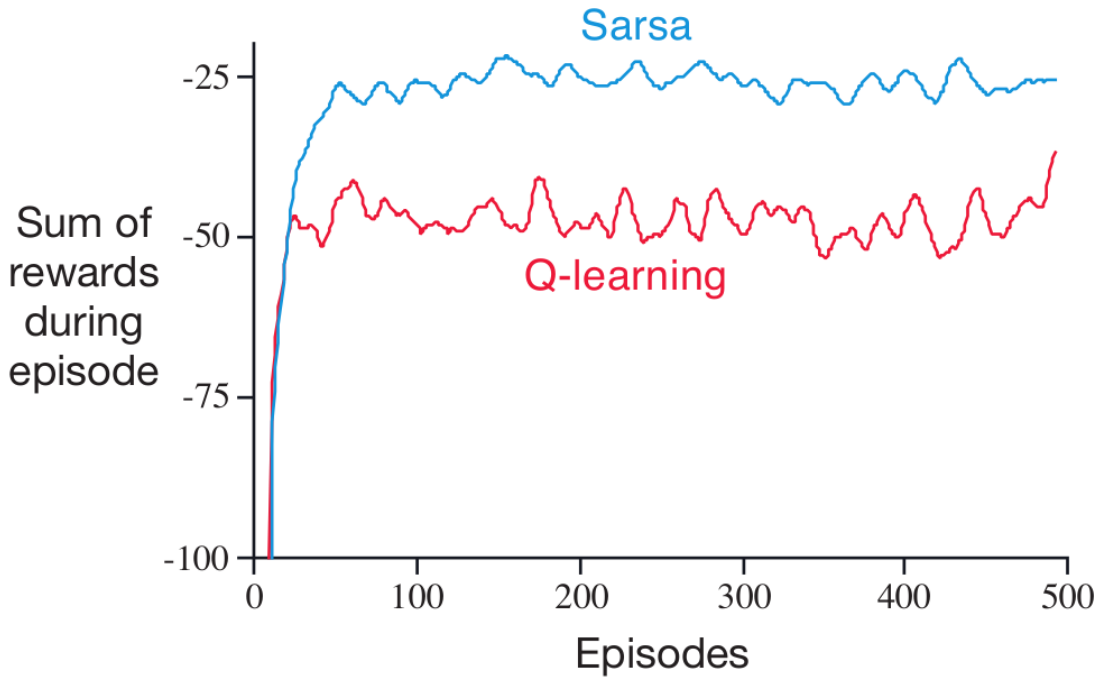
Comparison Algorithm

SARSA

- Safer path, earns higher sum of rewards

Q-Learning (ε-greedy, $\varepsilon = 0.1$)

- Optimal path, but exploration can cause falling from cliff



Summary Q-Learning

Q-Learning is an off-policy approach for learning of action-values $q(s, a)$:

- Next action is chosen using behaviour policy $A_{t+1} \sim b(\cdot|S_t)$
- But we consider alternative successor action $a' \sim \pi(\cdot|S_t)$
- And update $q(S_t, A_t)$ towards value of alternative action

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \Big(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t) \Big)$$

(Silver 2015)

Bias in Q-Learning – Example: Roulette

Roulette - Gambling Game

- Actions: 171 actions: bet $1 on one of 170 options and continue playing, or ‘stop’ which ends the episode
- Reward: is high variance, with negative expected value; for ‘stop’ $r = 0$

- Estimate the action-value function from experience.
- What would you expect after testing each action once?
- Do you see a problem with variance or bias?

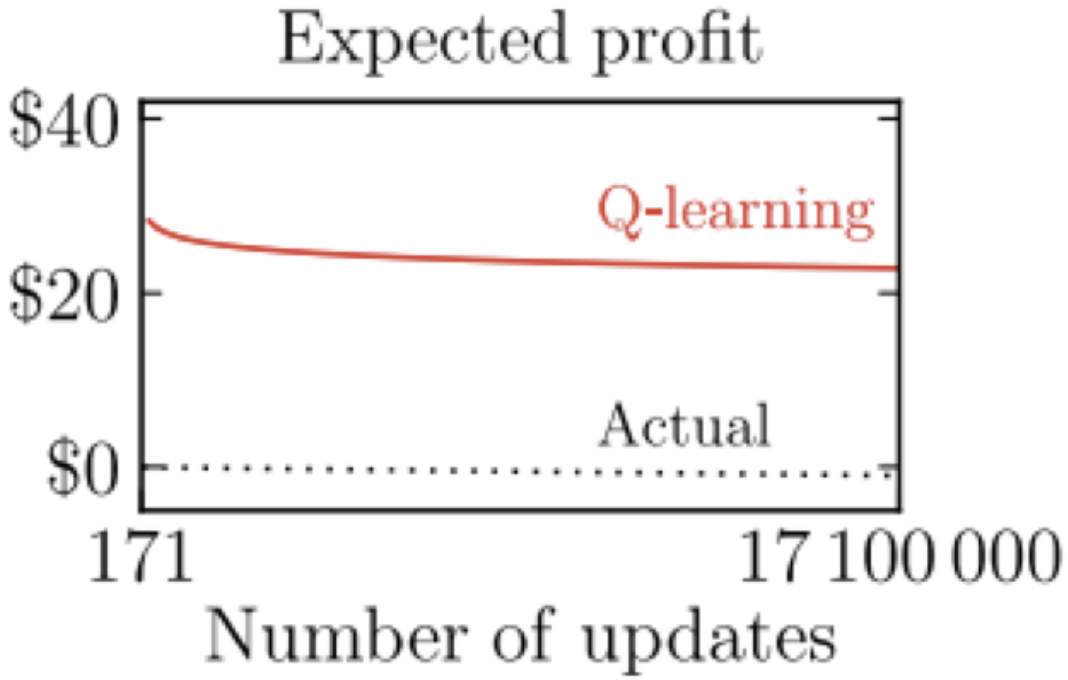
(Hado Hasselt 2010)

Example: Roulette – Overestimation in Q-Learning

Roulette - Gambling Game

All actions have high variance reward, with negative expected value.

Introduces initially large value estimate when one action randomly hits.



(Hado Hasselt 2010)

Q-learning overestimation

Q-learning overestimates because it uses the same values to select and to evaluate

$$\max_a q_t(S_{t+1}, a) = q_t(S_{t+1}, \arg\max_a q_t(S_{t+1}, a))$$

These values are only approximate!

- more likely to select overestimated values
- less likely to select underestimated values
- This causes upward bias

Solution

Decouple **selection** from **evaluation**.

(Hado van Hasselt und Borsa 2021)

Double Q-Learning

Store two action-value functions: q and q'.

$$R_{t+1} + \gamma q'_t(S_{t+1}, \arg\max_a q_t(S_{t+1}, a)) \quad (1)$$

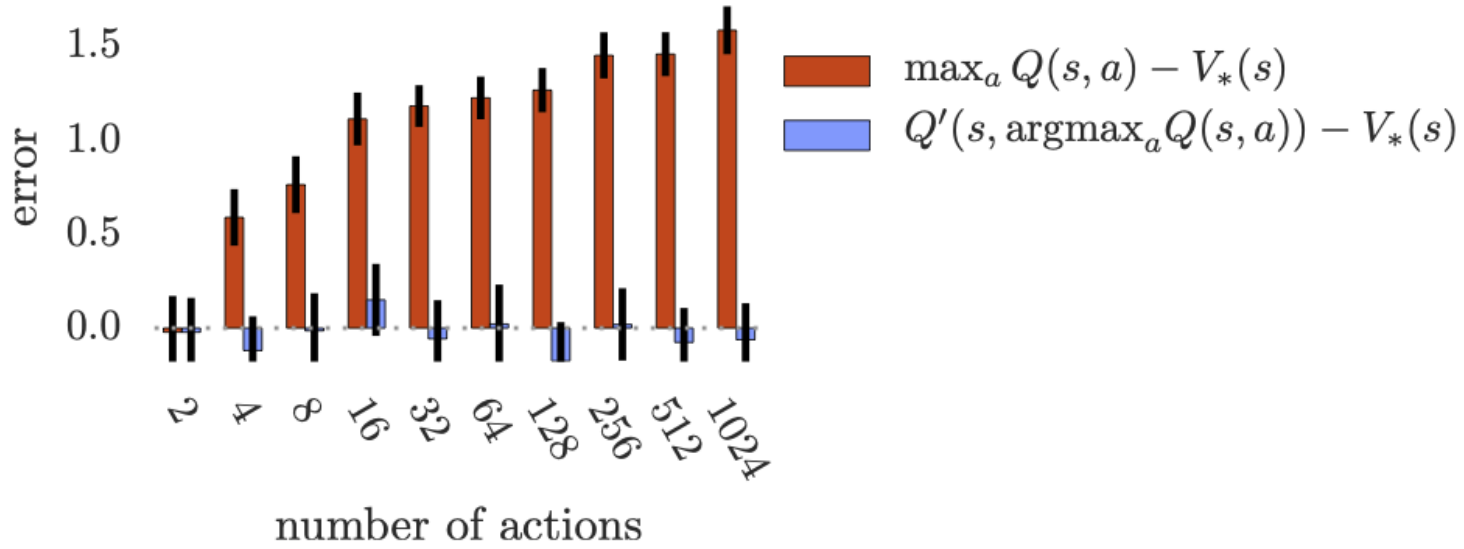
$$R_{t+1} + \gamma q_t(S_{t+1}, \arg\max_a q'_t(S_{t+1}, a)) \quad (2)$$

- Each timestep, pick q or q' (e.g., randomly) and update using (1) for q or (2) for q'.
- Both can be used to act (e.g., use policy based on $\frac{(q+q')}{2}$).

Double Q-learning also converges to the optimal policy under the same conditions as Q-learning.

(Hado van Hasselt und Borsa 2021)

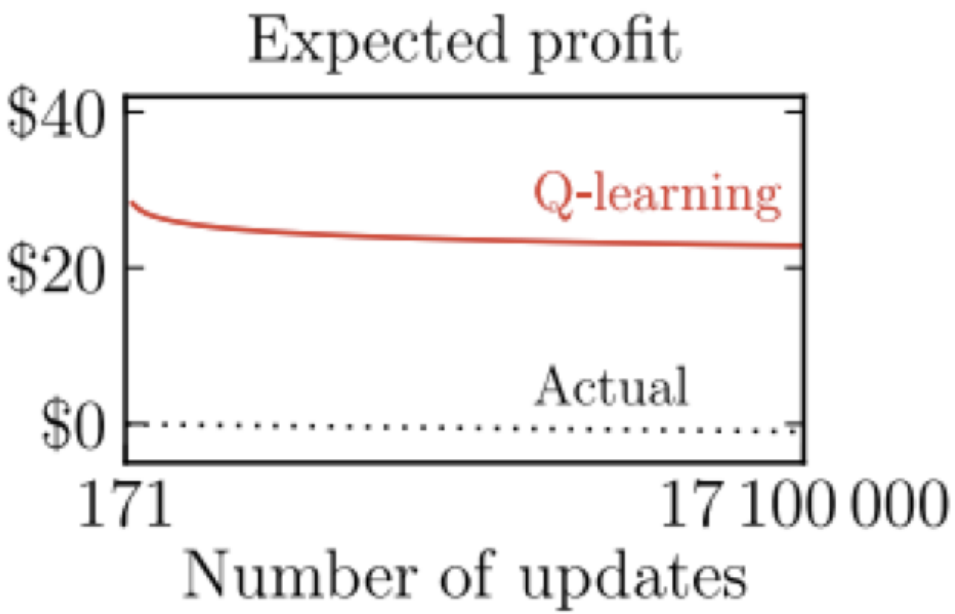
Double Q-Learning – Value Estimates

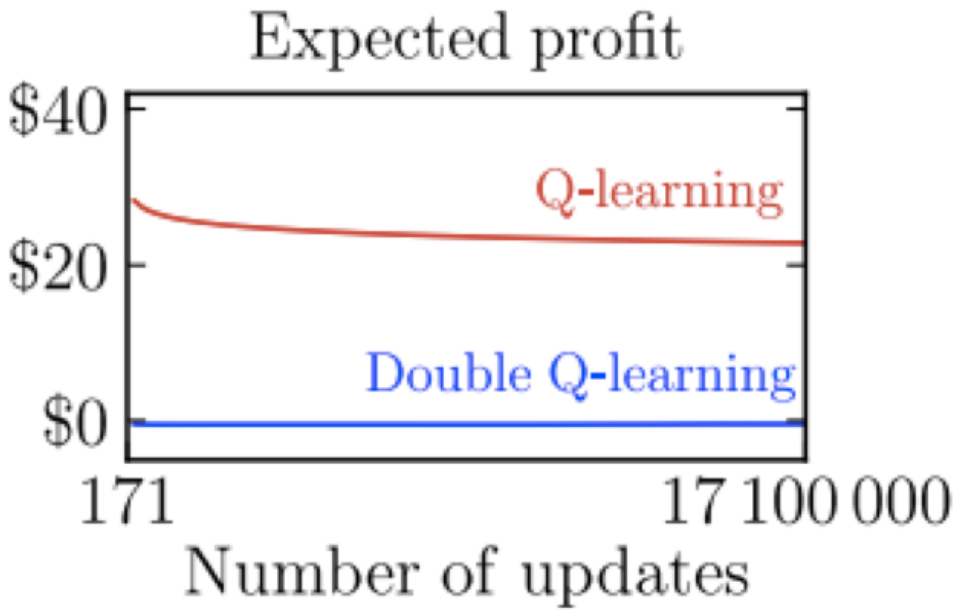


- Orange: Bias for a single Q-learning update: $Q(s,a) = V_*(s) + \varepsilon$
- Blue: used Q' identically, independently for estimate of value
- errors are independent standard normal random variables

(H. van Hasselt, Guez, und Silver 2016)

Example Roulette – Results Double Q-Learning



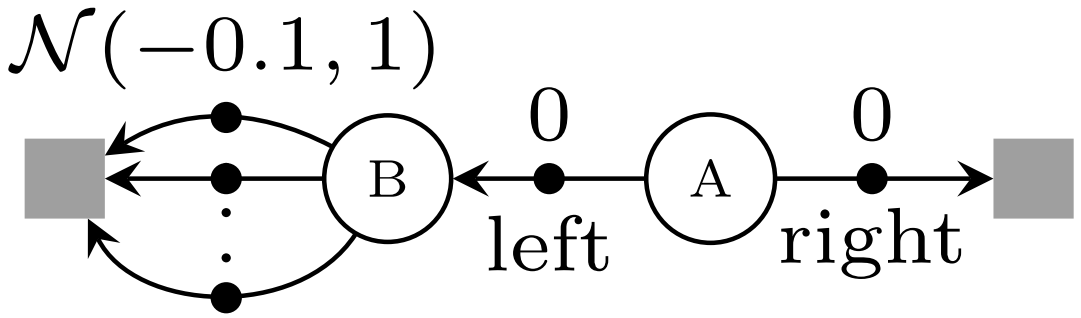


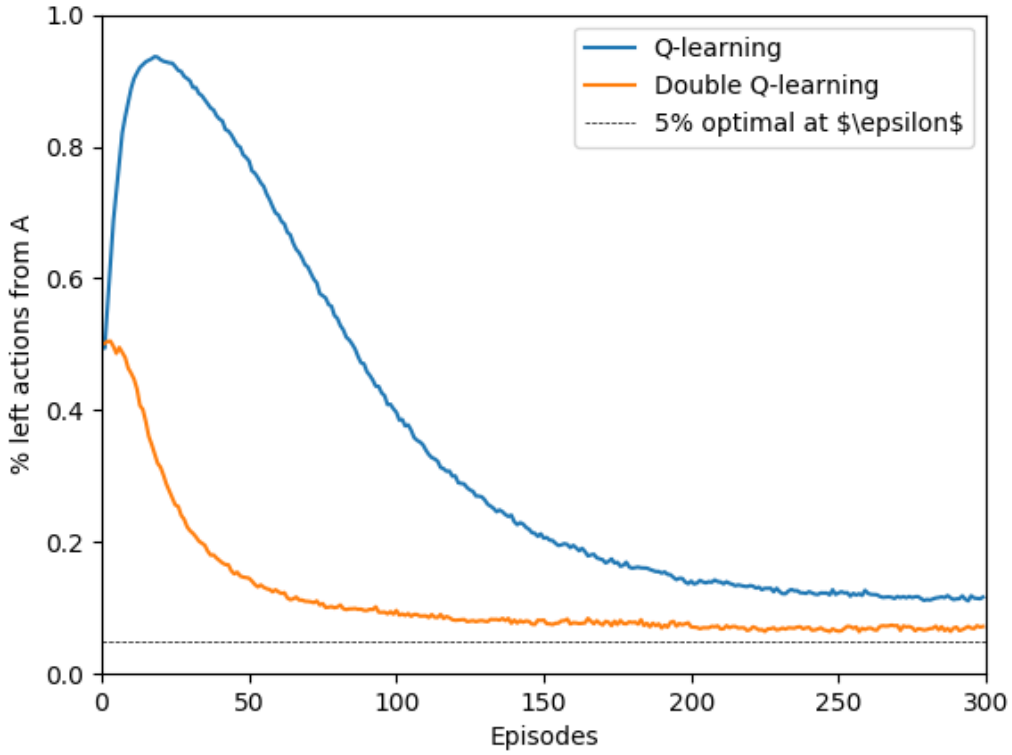
(Hado van Hasselt und Borsa 2021)

Example MDP: Double Q-Learning

Small Markov Decision Process

The reward given from ‘B’ is stochastic: for many different actions a reward is drawn from a normal distribution (this is similar to the roulette example).

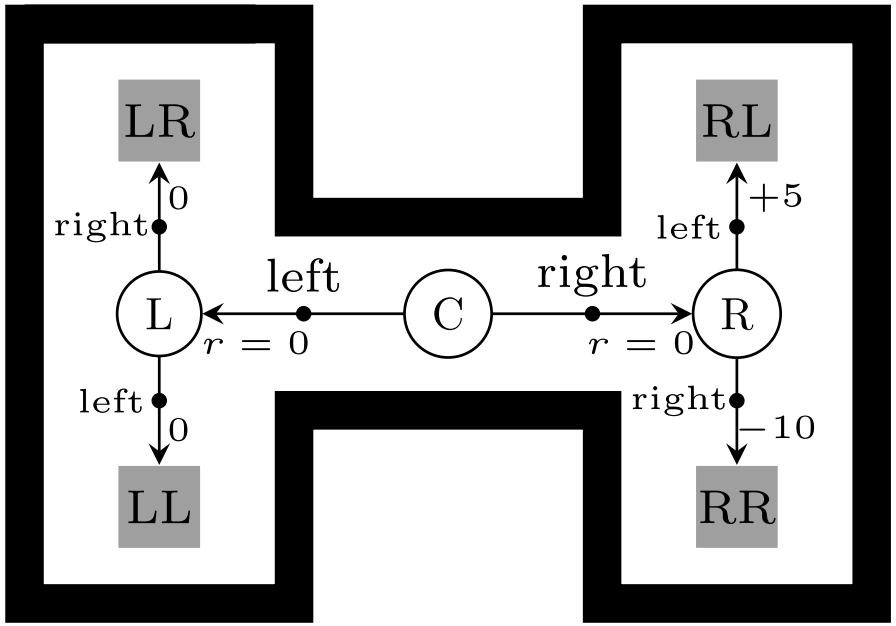




(Sutton und Barto 2018)

Possible Problem for Off-Policy Learning

Recap - Simple Maze



Off-Policy MC

We generat trajectories using a random policy.

Afterwards we update q and want to converge.

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha(G_t - q(S_t, A_t))$$

Observation: After a couple iterations, action-value-function for moving right – $q(C, right)$ – might become negative and the improved policy will pick 'left' from 'C'.

Reason: Even though $q('R', \text{'right'}) \approx 0.01$ might be very small (for stochastic π), the behavioral policy is still selecting it 50% of the time.

Action-values are wrt. a given policy. We have to **adjust for different probabilities**!

Importance Sampling

General Problem: Estimating an unknown function

Goal: Calculate an expectation of a function $f(x)$, where $x \sim p(x)$ is subject to some distribution.

The estimate is given as

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i), \text{ for MC}$$

Monte-Carlo approach

We simply sample x from the distribution $p(x)$ and take the average of all samples to get an estimation of the expectation.

Problem: It might be hard to directly use $p(x)$ for sampling.

(Zhang 2019)

Approach: Sample from a different distribution

Instead, we can use a different distribution for sampling.

But, we have to correct for this which can be directly derived for the estimate:

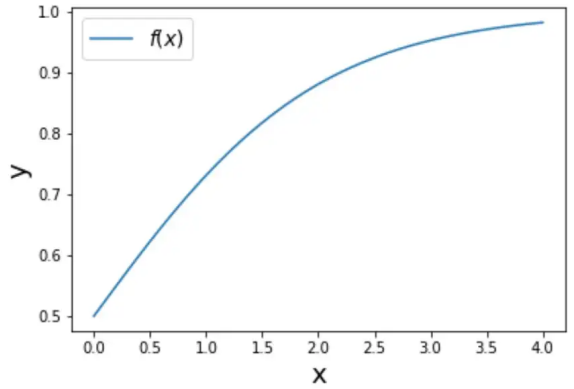
$$\mathbb{E}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{q(x)}{q(x)}p(x)dx \approx \frac{1}{n}\sum_i f(x_i)\frac{p(x_i)}{q(x_i)}$$

We estimate the expectation as we sample from another distribution $q(x)$.

$\frac{p(x)}{q(x)}$ is the sampling ratio that acts as a correction weight to offset the probability sampling from a different distribution.

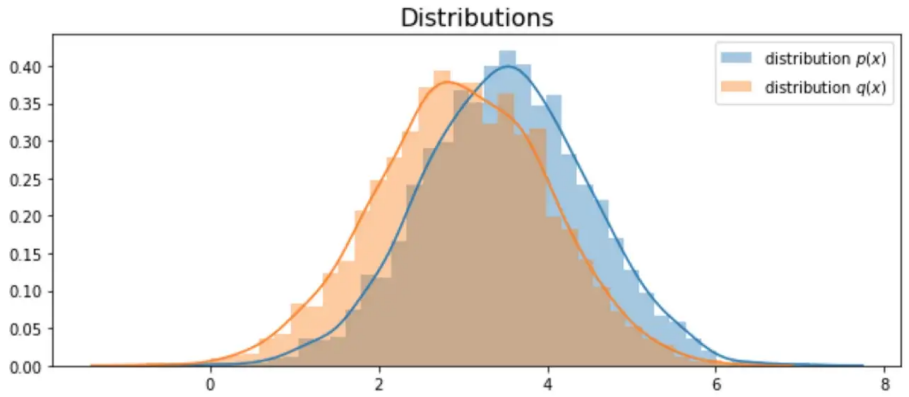
Example: Finding an Estimate for Function

Function $f(x)$



Given is a simple exponential function.

Two distributions for x



We are using two normal distributions (mean of 3 and 3.5) as simple examples.

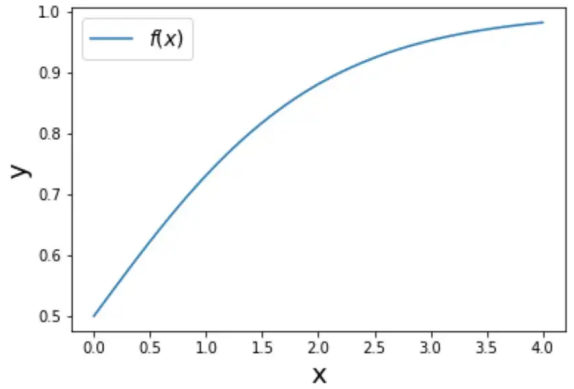
Estimates

The real estimate (ground truth) $\mathbb{E}_\pi \approx 0.954$, using 1000 samples. The estimate using $q(x)$ and the sampling ratio as correction produes $\mathbb{E} \approx 0.949$, , variance of 0.30

(Zhang 2019)

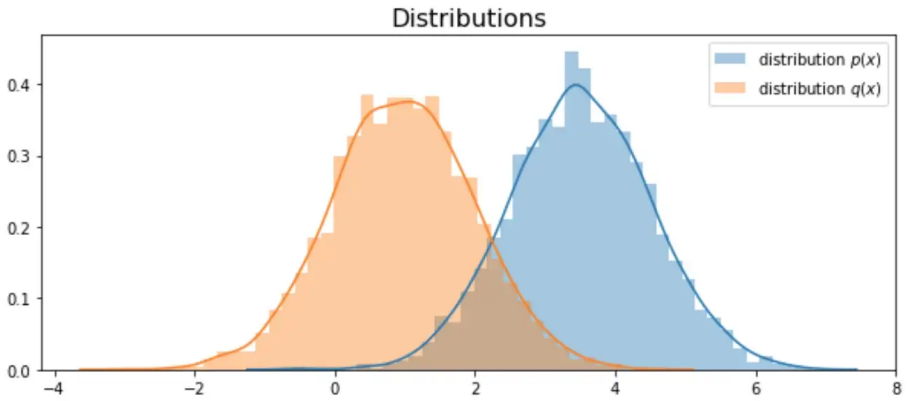
Example 2: Finding an Estimate for Function

Function $f(x)$



Given is a simple exponential function.

Two dissimilar distributions for x



We are using two normal distributions (mean of 3 and 3.5) as simple examples.

Estimates

The real estimate (ground truth) $\mathbb{E}_\pi \approx 0.954$ (using 5000 samples). The estimate using $q(x)$ and the sampling ratio as correction produes $\mathbb{E} \approx 0.995$, variance of 83.36.

(Zhang 2019)

Variance when using Importance Sampling

$$\mathbb{E}[f(x)] \approx \frac{1}{n} \sum_i f(x_i) \frac{p(x_i)}{q(x_i)}$$

When the importance sampling ratio is high, this will introduce large variance:

$$Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2, \text{ with } X = f(x) \frac{p(x)}{q(x)}$$

Therefore, we should aim for selecting $q(x)$ appropriately, i.e. in a way where $f(x)p(x)$ is already large.

Importance Sampling in RL

From a starting state S_t, the probability of the subsequent state–action trajectory occurring under a policy π is

$$
\begin{aligned}
& p(A_t,\, S_{t+1}, A_{t+1}, \ldots, S_T | S_t, A_{t:T-1} \sim \pi) \\
&= \pi(A_t, S_t) p(S_{t+1} | S_t, A_t) \pi(A_{t+1}, S_{t+1}) \cdots p(S_T | S_{T-1}, A_{T-1}) \\
&= \prod_{k=t}^{T-1} \pi(A_k | S_k) p(S_{k+1} | S_k, A_k)
\end{aligned}
$$

Importance Sampling in RL

In off-policy RL: we are optimizing policy π, but follow behavioral policy b.

Importance Sampling Ratio

The relative probability of the trajectory under the target and behavior policies is given as

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k,A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

Note: The ratio only depends on the probabilities of selecting an action wrt. to the two differing policies! Does not require knowledge on transition probabilities.

Importance Sampling as a Correction

Intuition:

- scale down rewards that are rare under π, but common under b
- scale up rewards that are common under π, but rare under b

Importance sampling can dramatically *increase variance*.

(Hado van Hasselt und Borsa 2021)

(Ordinary) Importance Sampling

Goal: estimate the expected returns for the target policy π

$$\mathbb{E}[G_t|S_t = s]$$

Available: only returns G_t due to the behavior policy which can give us

$$\mathbb{E}[G_t|S_t = s] = v_b(s)$$

The ratio $\rho_{t:T-1}$ transforms the collected returns to have the right expected value towards the target policy:

$$\mathbb{E}[\rho_{t:T-1}G_t|S_t = s] = v_\pi(s)$$

Importance Sampling for Off-Policy Monte-Carlo

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

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

- Update value towards corrected return

$$v(S_t) \leftarrow v(S_t) + \alpha\left(G_t^{\pi/b} - v(S_t)\right)$$

(Silver 2015)

Importance Sampling for Off-Policy TD

- Use TD targets generated from b to evaluate π
- Weight TD target $R + \gamma v(S_{t+1})$ according to similarity between policies
- Only need a single importance sampling correction

$$v(S_t) \leftarrow v(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{b(A_t|S_t)}(R_{t+1} + \gamma v(S_{t+1})) - v(S_t)\right)$$

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

(Silver 2015)

Q-Learning - No Importance Sampling needed

Q-Learning is off-policy. But we update the value functions:

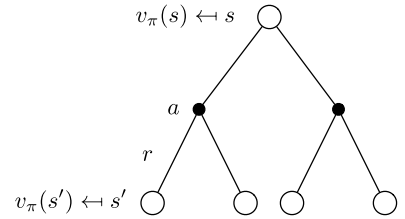
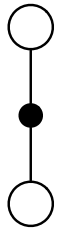
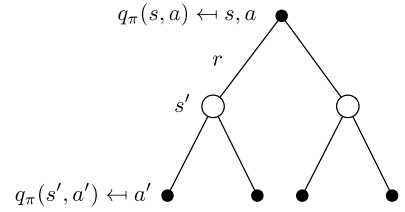
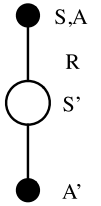
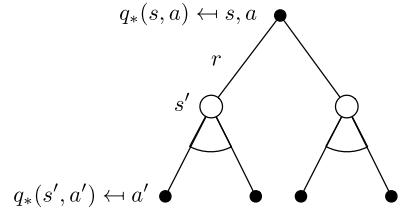
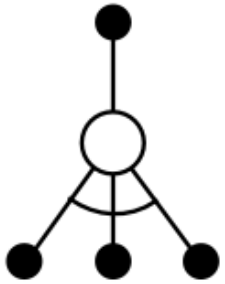
$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t) \right)$$

- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim b(\cdot|S_t)$
- But we consider alternative successor action $A' \sim (\cdot|S_t) and update$q(S_t,A_t) towards value of this alternative action

(Silver 2015)

Relationship Between DP and TD

(Silver 2015)

	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
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

(Silver 2015)