DS-GA 3001 001 | Lecture 4

Reinforcement Learning

Jeremy Curuksu, PhD
NYU Center for Data Science
jeremy.cur@nyu.edu

February 26, 2025

DS-GA 3001 RL Curriculum

Reinforcement Learning:

- Introduction to Reinforcement Learning
- Multi-armed Bandit
- Dynamic Programming on Markov Decision Process
- **Model-free Reinforcement Learning**
- Value Function Approximation (Deep RL)
- Policy Function Approximation (Actor-Critic)
- Planning from a Model of the Environment (AlphaZero)
- Aligning AI systems to Human Preferences (ChatGPT)
- Examples of Industrial Applications
- Advanced Topics

Reinforcement Learning

Last week: Dynamic Programming on MDPs

- Markov Decision Process
- Value Functions and Bellman Equations
- Dynamic Programming

Today: Model-free Reinforcement Learning

- **Monte Carlo and Temporal Difference**
- **Sample-based Policy Evaluation**
- **Sample-based Policy Optimization**
- **Off-policy Learning**

Generalization to Model-free RL

Why?

- **Dynamic Programming** requires a model of state transitions and rewards to carry out a one-step look-ahead full-width backup at each iteration
- **Problem:** In most cases, a MDP model of state transitions and rewards is not available
- **Solution:** Sample the state-action space

Monte Carlo and Temporal Difference

Sampled-based RL: Monte Carlo

Use samples of experience to learn without model

- We want to learn v_π from series of experience under policy π:

$$v_\pi(s) = \mathop{\mathbb{E}}_\pi(G_t|s)$$

- Instead of updating true expected return, sample its average:

$$v_{k+1}(s_t) = v_k(s_t) + \alpha_k \left(\sum_{i=0}^{T} \gamma^i r_{t+i+1} - v_k(s_t) \right)$$

- This requires sampling an entire episode for each update
- This is called **Monte Carlo policy evaluation**

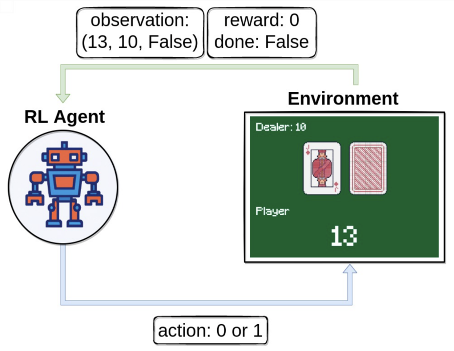
Example of MC Policy Evaluation

Case Study: Blackjack

Goal: Draw cards so that their sum $>$ dealer's sum and ≤ 21

Overview of Environment:

- Initial state: Two cards for player, two cards for dealer
- Dealer shows one of its cards
- Cards are worth their number, face cards $= 10$, ace $= 1$ or 11
- Player requests cards 1 by 1
- When player exits dealer draws



Definition of States:

- Agent's cards (sum of points)
- Dealer's showing card
- Boolean that represents whether the agent has a usable ace

Example of MC Policy Evaluation

Case Study: Blackjack

Goal: Draw cards so that their sum $>$ dealer's sum and ≤ 21

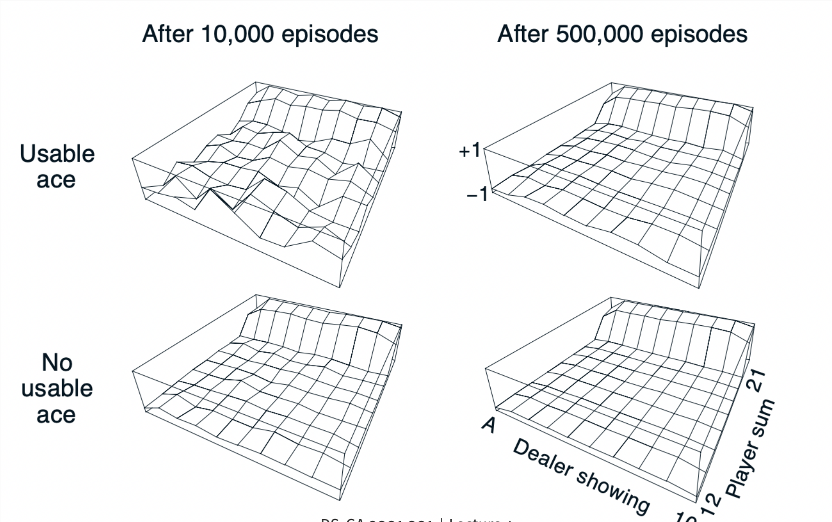
Definition of Actions:

- Draw another card (automatically draw if sum < 12)
- Stop and terminate

Definition of Rewards:

- 0 when agent or dealer draw cards
- -1 (lose) if agent's sum > 21
- -1 (lose) if agent stops and agent's sum $\leq$ dealer's sum
- +1 (win) if agent stops and agent's sum $>$ dealer's sum
- No discounting

Results of Policy Evaluation on Blackjack for policy that exits only when sum is 20 or 21



Temporal Difference Learning

Sample Bellman equations instead of full episodes

- DP estimates values of states based on estimates of values of successor states, without waiting for a final outcome

$$v_\pi(s) = \mathop{\mathbb{E}}_{\pi}(G_t|s) = \mathbb{E}(r_{t+1} + \gamma\, v_\pi(s_{t+1}) \,|\, s)$$

$$\forall s\,,\ v_{k+1}(s) = \sum_a \pi(a\,|\,s) \sum_{s',\,r} p(s', r\,|\,s, a)\,[r + \gamma\, v_k(s')]$$

- Instead of updating true expected DP target, sample it:

$$v_{k+1}(s_t) = v_k(s_t) + \alpha_k\,(r_{t+1} + \gamma\, v_k(s_{t+1}) - v_k(s_t))$$

- This **does not** require sampling entire episodes for each update
- This is called **Temporal Difference policy evaluation**

Bias-variance tradeoff of MC vs. TD

MC samples to learn v_π online from entire episodes:

- MC learns from complete sequences, no need to bootstrap
- But MC must wait until the end of an episode to update v_π
- MC works only in episodic (terminating) environments
- **Return $G_t = (r_{t+1} + \gamma\, r_{t+2} + \dots)$ is an unbiased estimate of $v_\pi(s_t)$**
  ...but has high variance

TD samples to learn v_π online by bootstrapping:

- TD can learn from incomplete sequences, by bootstrapping
- TD learns after every step, before knowing the final outcome
- TD works in continuing (non-terminating) environments
- **TD target $r_{t+1} + \gamma\, v(s_{t+1})$ is a biased estimate of $v_\pi(s_t)$**
  ...but has lower variance

Case Study: Drive Home*

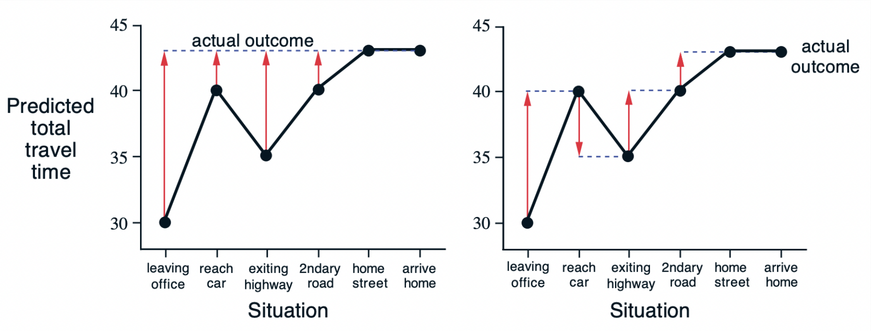
An episode driving back home from the office...

State	Elapsed Time (min)	Predicted Time to Go	Predicted Total Time
Leave office	0	30	30
Reach car, raining	5	35	40
Exit highway	20	15	35
Behind truck	30	10	40
Home street	40	3	43
Arrive home	43	0	43

* Sutton and Barto, 2018

Case Study: Drive Home*

An episode with Monte Carlo vs. Temporal Difference



Monte Carlo | **Temporal Difference**

* Sutton and Barto, 2018

Convergence of MC vs. TD

MC and TD both converge to the same values as $N \to \infty$, but what about finite experience?

Case study: Two states A and B, no discounting, N = 8 episodes:

A, 0 then B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

What are $v(A)$ and $v(B)$?

Convergence of MC vs. TD

MC and TD both converge to the same values as $N \to \infty$, but what about finite experience?

Case study: Two states A and B, no discounting, N = 8 episodes:

A, 0 then B, 0

B, 1

B, 1

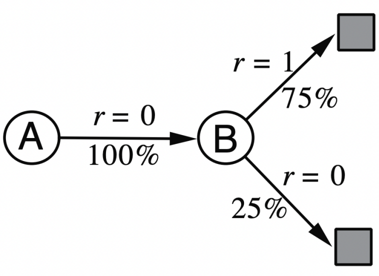
B, 1

B, 1

B, 1

B, 1

B, 0



Convergence of MC vs. TD

MC and TD both converge to the same values as $N \to \infty$, but what about finite experience?

- Repeatedly sampling a finite number of episodes is equivalent to sampling from an *empirical* model
- MC converges to best mean-squared fit for observed returns:

$$v(s) \to \arg\min \sum_k \left(G_t^k - v(s)\right)^2$$

  MC does not exploit sequential dependence of states

- TD converges to solution of max likelihood Markov model:

$$v(s) \to \sum_{s',r} \hat{p}(s', r \mid s)\,[r + v(s')]$$

  TD exploits the Markov property

Convergence of MC vs. TD

MC and TD both converge to the same values as $N \to \infty$, but what about finite experience?

- Repeatedly sampling a finite number of episodes is equivalent to sampling from an *empirical* model
- MC converges to best mean-squared fit for observed returns:

$$v(s) \to \arg\min \sum_k \left(G_t^k - v(s)\right)^2$$

AB case study: $v(A) = 0\,,\;\; v(B) = 0.75$

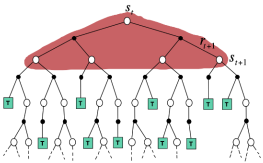
- TD converges to solution of max likelihood Markov model:

$$v(s) \to \sum_{s',r} \hat{p}(s', r \mid s)\,[r + v(s')]$$

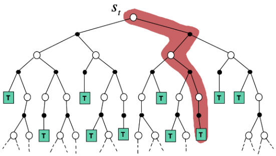
AB case study: $\hat{p}(B, 0 \mid A) = 1\,,\;\; v(B) = 0.75 \;\;\Rightarrow\;\; v(A) = 0.75$

DP vs. MC vs. TD

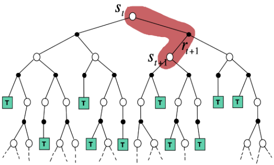
Dynamic Programming



Monte Carlo



Temporal Difference



Bootstrapping: update involves an estimate

DP bootstraps

MC does not bootstrap

TD bootstraps

Sampling: update samples an expectation

DP does not sample

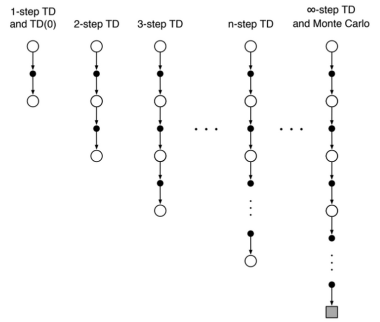
MC samples

TD samples

n-step TD and TD(λ)

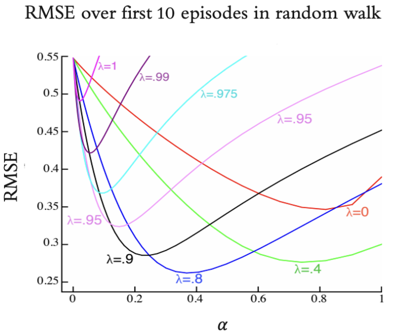
n-step TD: Temporal Difference learning from n-step updates

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n v(s_{t+n})$$

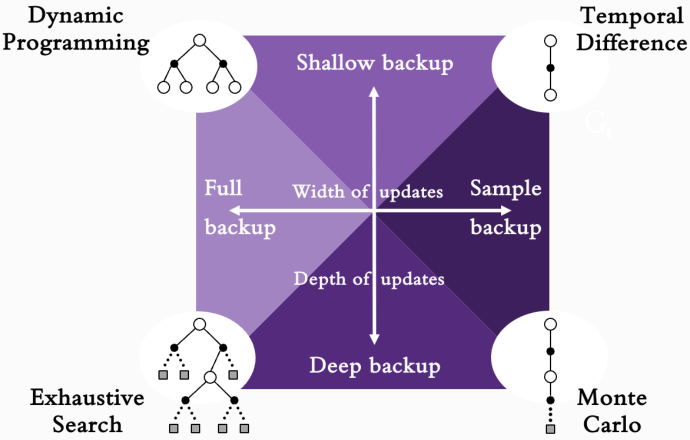


TD(λ): Weighted sum of all possible n-step updates

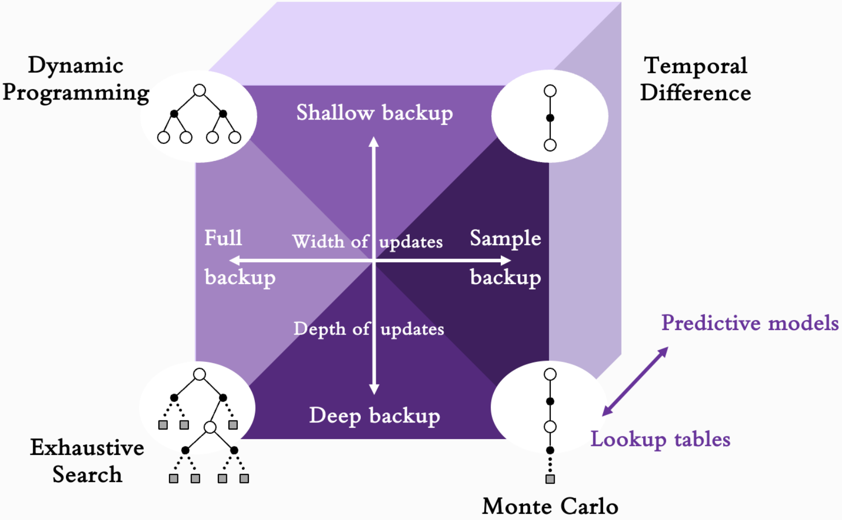
$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$



Policy Evaluation Method Space



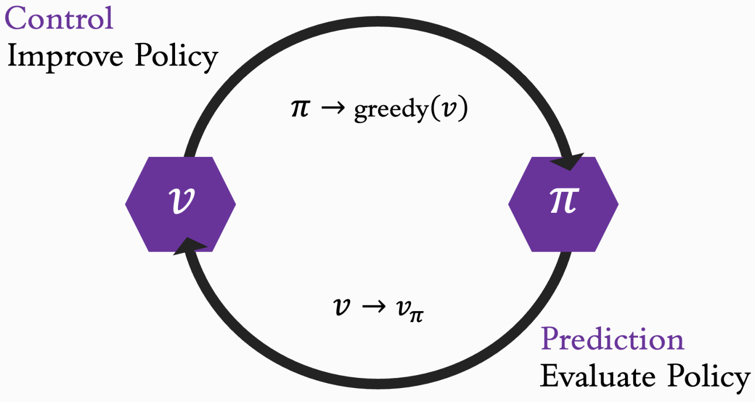
Policy Evaluation Method Space



Sample-based Policy Optimization

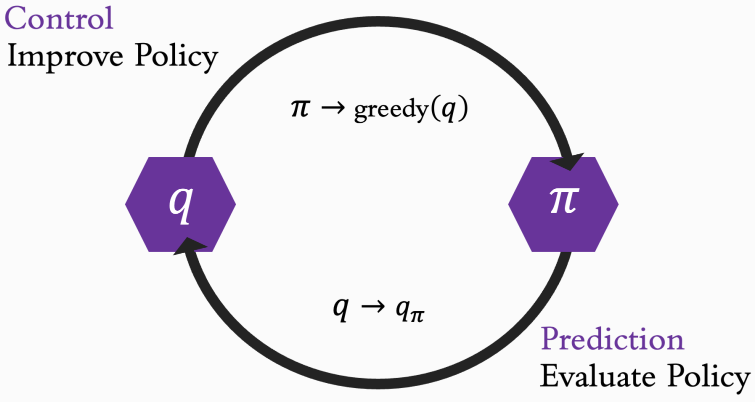
Generalized Policy Iteration

All RL methods are Generalized Policy Iteration methods



Generalized Policy Iteration

All RL methods are Generalized Policy Iteration methods



Monte Carlo Policy Iteration

Find a better policy π' given estimated value of π

- **Recall the Policy Improvement theorem:**

$$\forall \pi, \forall s: \ \pi'(s) = \arg\max_a q_\pi(s,a) \ \text{is better or same as } \pi(s)$$

- Policy improvements over $v(s)$ require a model $\Rightarrow$ **learn $q(s,a)$**
- Greedy policy improvements do not explore $\Rightarrow$ **use ϵ-greedy**
- **ϵ-greedy algorithm:**
  - Select random action (explore) with $p = \epsilon$
  - Select greedy action (exploit) with $p = 1 - \epsilon$
- **Theorem:** The action-value function $q(s,a)$ estimated from MC control with ϵ-greedy converges to the optimal $q_*(s,a)$: $q \to q_*$

Practice: Monte Carlo RL Algorithm

MC RL computes $q(s, a)$ as moving average over complete episodes, learning "episode by episode" with no model of the environment

Initialize $q(s, a)$ and $\pi(s)$ arbitrarily

Loop forever:

Initialize s_0

Experience an episode $(s_0, a_0, r_1, s_1, a_1, r_2, ..., r_T)$ following π:

Loop for each step t of episode:

$G = r_{t+1} + \gamma\, r_{t+2} + ... + \gamma^T\, r_{t+1+T}$

$q(s, a) = q(s, a) + \alpha\, (G - q(s, a))$

Update π at s given $q(s, a)$ by ϵ-greedy soft update

Practice: ϵ-Greedy Soft Policy

Select random action with probability ϵ
Select greedy action with probability $1 - \epsilon$

Loop for all a in s:

$$a^* = \arg\max_a q(s, a)$$

$$\pi(s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{for } a = a^* \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{for } a \neq a^* \end{cases}$$

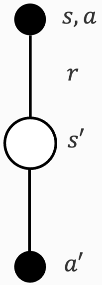
* $|\mathcal{A}(s)|$ is the number of possible actions in s

Temporal Difference Policy Iteration

Update Action-Value Functions with "SARSA":

- Apply 1-step TD to $q(s, a)$:

$$q(s,a) = q(s,a) + \alpha \left(r + \gamma\, q(s', a') - q(s,a) \right)$$



- Update $\pi(s)$ given $q(s, a)$ by ϵ-greedy soft update
- Same as MC algorithm but use TD return instead of MC return to update $q(s, a)$
- Compared to MC, TD can learn online at every step, it can learn from incomplete episodes, and it has potentially large bias but lower variance

Practice: SARSA TD Algorithm

SARSA TD computes $q(s, a)$ at every step based on previous q-value estimates (bootstrapping), with no model of the environment

Initialize $q(s, a)$ and $\pi(s)$ arbitrarily

Initialize s

Select a from s following $\pi(s)$

Loop forever: (or for each step t of an episode)

Take a, observe r_{t+1} and s'

Select a' from s' following $\pi(s')$

$q(s, a) = q(s, a) + \alpha\ (r_{t+1} + \gamma\, q(s', a') - q(s, a))$

Update π at s given $q(s, a)$ by ϵ-greedy soft update

$s = s',\ \ a = a'$

Off policy learning

On-policy vs. Off-policy RL

On-policy learning

- ***"Learn on the job":*** Learn about policy π from experience sampled from π (that is, by following π)

Off-policy learning

- ***"Look over the agent's shoulder":*** Learn about a *target policy* π from experience sampled from a *behaviour policy* b
- Evaluate π by computing $v_\pi(s)$ or $q_\pi(s, a)$ while following b

$$(s_0, a_0, r_1, s_1, a_1, r_2, ..., r_T) \sim b$$

- **Motivations:**
  - Learn about optimal policy while following exploratory policy
  - Re-use experience from old policies (*Experience Replay*)
  - Learn from observing humans or other agents
  - Learn about multiple policies while following one policy

Off-policy Q-learning

Q-learning estimates the value of the greedy policy:

$$q_{k+1}(s,a) = q_k(s_t,a_t) + \alpha_k \left(r_{t+1} + \gamma \max_{a'} q_k(s_{t+1},a') - q_k(s_t,a_t) \right)$$

- **Q-learning systematically updates the greedy policy whatever the behavior policy followed is**, thus keeping learning focused on greedy actions even when the agent explores other actions

- **Theorem: Q-learning control converges to the optimal action value function, $q \to q_*$, as long as we take each action in each state infinitely often**. No need for greedy behavior because we update q-values for the greedy behavior anyway!

Practice: Q-learning TD Algorithm

Q-learning computes $q(s, a)$ at every step using Bellman optimality equation (bootstrapping), with no model of the environment

Initialize $q(s, a)$ and $b(s)$ arbitrarily

Initialize s

Loop forever: (or for each step t of an episode)

Select and take a from s following $b(s)$, observe r_{t+1} and s'

$$q(s, a) = q(s, a) + \alpha \left(r_{t+1} + \gamma \max_{a'} q(s', a') - q(s, a) \right)$$

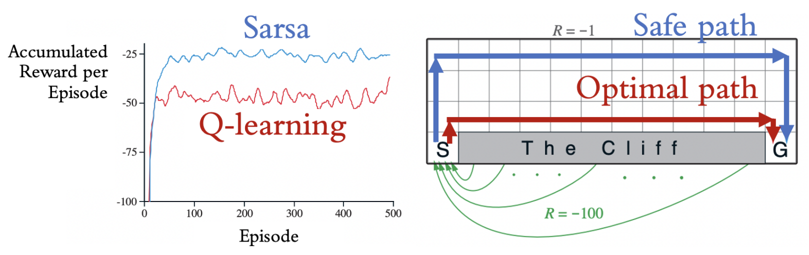
Update b at s given $q(s, a)$ by ϵ-greedy soft update

$s = s'$

SARSA vs. Q-learning Example

Cliff Walking Gridworld

- Comparison of performance of on-policy (Sarsa) and off-policy (Q-learning) methods with ϵ-greedy action selection ($\epsilon = 0.1$)



Upward Bias of Q-learning

Q-learning uses the same Q-function to select and evaluate actions, leading to a self-fulfilling prophecy

- TD target $r_{t+1} + \gamma \max_{a'} q(s', a')$ is a biased sample of $q_*(s, a)$
- Same Q-value *estimate* used to select a' and evaluate $q(s', a')$

$$\max_{a'} q(s', a') = q(s', \arg\max_{a'} q(s', a'))$$

- This tends to overselect overestimated values and underselect underestimated values, perpetuating an upward bias
- **Solution: Decouple functions used for selection vs. evaluation**
- **This is called Double Q-learning**

Double Q-learning

Double Q-learning uses independent Q-functions to select vs. evaluate actions

- **Store two functions q_1 and q_2:**

$$q_1(s,a) = q_1(s,a) + \alpha \left(r_{t+1} + \gamma\, q_1(s', \arg\max_{a'} q_2(s',a')) - q_1(s,a) \right)$$

$$q_2(s,a) = q_2(s,a) + \alpha \left(r_{t+1} + \gamma\, q_2(s', \arg\max_{a'} q_1(s',a')) - q_2(s,a) \right)$$

- **At each step:**
  - Update either q_1 or q_2 (e.g., select each with $p = 0.5$)
  - Act by ϵ-greedy soft update using q_1 or q_2 (or $q_1 + q_2$)

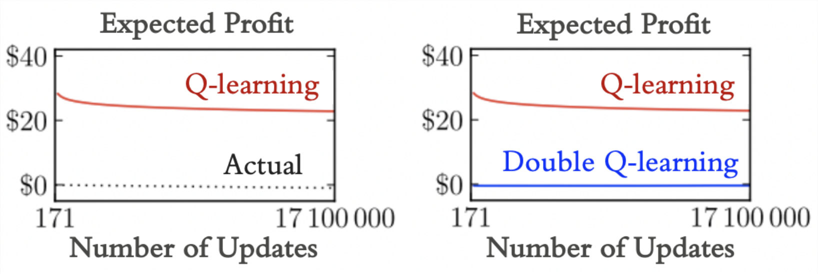
Double Q-learning Example

The Roulette

Double Q-learning Example

Double Q-learning outperforms Q-learning by reducing the overoptimism due to value estimation errors

- Comparison of Double Q-learning vs. Q-learning on the roulette case study

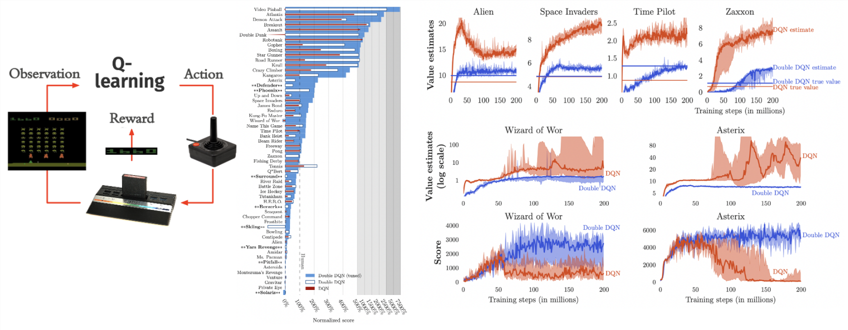


van Hasselt, 2010

Double Q-learning Example

Double Q-learning outperforms Q-learning by reducing the overoptimism due to value estimation errors

- Comparison of DDQN performance vs. DQN on 57 Atari video games



van Hasselt, Silver, 2015

Today's Takeaways

MC and TD can learn optimal policies without model, by sampling expected returns across the state-action space

- **MC samples entire episodes and updates values one episode at a time**. It is unbiased but can have large variance because episodes can be very different from one another
- **TD bootstraps to update values at every step**, shifting each estimate toward the estimate that immediately follows it
- **When sampling is finite, TD is more stable than MC** but can be more biased toward wrong results
- **Q-learning is an off-policy TD algorithm** which focuses on learning the optimal policy while sampling other policies

Thank you!