Faculté
des **sciences économiques** et de **gestion**
Université de Strasbourg

Reinforcement Learning

UE 2 Machine learning

Nattirat Mayer

Fall-Winter Semester 2025-26

Chapter 6. Temporal-Difference Learning

Here comes a new chapter!

6.1 Intro to Temporal-Difference (TD) Learning

In Chapter 4, DP (learning from the environment)

- DP uses *bootstrapping* technique to approximate value functions
- However, DP is model-based, i.e., we need to know the MDP environment (i.e., transition probability $p(s', r|s, a)$ and reward function $R(s, a)$).

In Chapter 5, MC (learning from experiences)

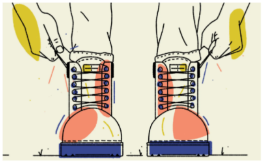
- MC is model-free, i.e., MC does not need knowledge of the environment
- However, MC is a slow learner as it learns from complete episodes

This chapter, TD combine best of both DP and MC. That is,

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes
- TD updates a guess towards a guess

Bootstrapping, Sampling, and TD

- Bootstrapping: update involves using current estimates to improve other estimates
  - MC does **not** bootstrap
  - DP **bootstraps**
  - TD **bootstraps**
- Sampling: learning from actual experience (interacting with the environment) rather than a known model
  - MC **samples**
  - DP does **not** sample
  - TD **samples**



Bootstrapping

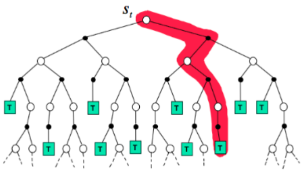
Reference: Huffpost, 2018

Sampling

Reference: cognitiveclass.ai, 2025

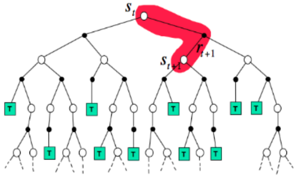
Monte-Carlo

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



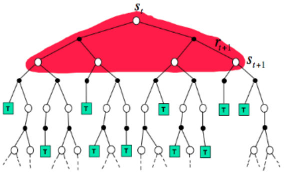
Temporal-Difference

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



Dynamic Programming

$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$



In this course, we look at the main TD Learning methods which consist of the following:

- **TD(0)**: the simplest form of TD learning for $V(S)$
- **SARSA**: an *on-policy* TD method for learning $Q(S_t, A_t)$
- **Q-Learning**: an *off-policy* TD method for learning $Q(S_t, A_t)$

6.2 TD(0)

Recall, we consider the incremental update rule of the form:

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

where α is a constant step-size parameter $\alpha \in (0, 1]$. In MC, the agent must wait until the end of an episode to determine the increment to $V(S_t)$.

TD updates the value at each step of the episode following the equation below,

$$V(S_t) \leftarrow V(S_t) + \alpha\big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big]$$

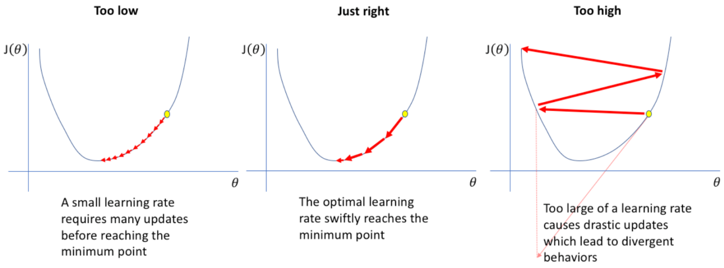
Same the equation presented last slide,

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

- $V(S_t)$
- TD target (our new/better guess)
- TD error (how wrong our guess)

The step-size parameter α

- It controls how much new information changes the estimate
- $\alpha \in (0, 1]$
- It is also regarded as 'learning rate' as in Deep Neural Network.


Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

TD(0) is to look for a one-step ahead update of $V(S_t)$. There are more TD(.) family of algorithms as follows:

- TD(0): one-step lookahead (fully bootstrapped)
- TD(1): two-step lookahead
- TD(n): n-step lookahead
- TD(∞): uses the full return $\Rightarrow$ equivalent to Monte Carlo
- TD-λ combines all n-step returns with exponentially decaying weights

6.3 SARSA

- SARSA (State-Action-Reward-State-Action) is an **on-policy** TD control algorithm.
- It is a sample-based version of policy iteration that uses the Bellman equation for action values.
- The update rule is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

6.4 Q-Learning

- **Q-Learning** is an **off-policy** Temporal-Difference (TD) algorithm.
- It is a sample-based version of **value iteration** that applies the *Bellman optimality equation*.
- The update rule is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- Unlike SARSA, Q-Learning updates using the **greedy action** (i.e., follows a different policy than the behavior policy) — hence, it is **off-policy**.

Same the equation presented last slide,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- $Q(S_t, A_t)$
- TD target (our new/better guess)
- TD error (how wrong our guess)

6.5 SARSA Vs Q-Learning

Q-Learning - *Off-Policy TD*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

SARSA - *On-Policy TD*

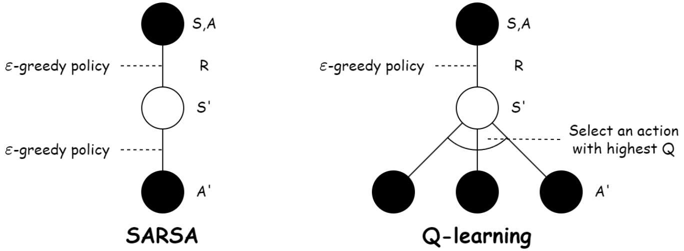
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Both **Q-Learning** and **SARSA** select actions using an ϵ**-soft policy**, often implemented as an ϵ**-greedy strategy**

$$\pi_\epsilon(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{if } a = \arg\max_a Q(s,a) \\ \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{otherwise.} \end{cases}$$

The optimal policy is, thus:

$$\pi_*(s) = \arg\max_a Q(s,a)$$

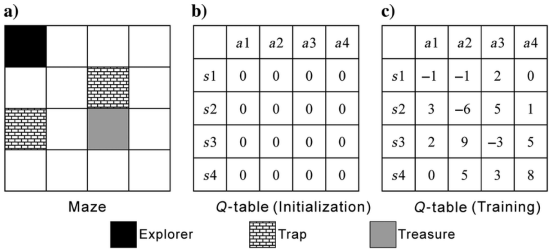

S,A
ε-greedy policy
R
S'
ε-greedy policy
A'
SARSA
S,A
ε-greedy policy
R
S'
Select an action with highest Q
A'
Q-learning

SARSA Vs Q-Learning

- Q-Learning is better/faster learning than SARSA
- Q-learning agent can learn from imitation and experience replay (off-policy)
- Q-Learning allows for randomness/stochastic exploration (e.g. with ϵ-greedy)

- SARSA is often safer to learn (on-policy)
- SARSA gives better total reward while learning

TD is a tabular learning with Q-table.



Q-table (Initialization)

	$a1$	$a2$	$a3$	$a4$
$s1$	0	0	0	0
$s2$	0	0	0	0
$s3$	0	0	0	0
$s4$	0	0	0	0

Q-table (Training)

	$a1$	$a2$	$a3$	$a4$
$s1$	−1	−1	2	0
$s2$	3	−6	5	1
$s3$	2	9	−3	5
$s4$	0	5	3	8

6.6 TD Tutorial

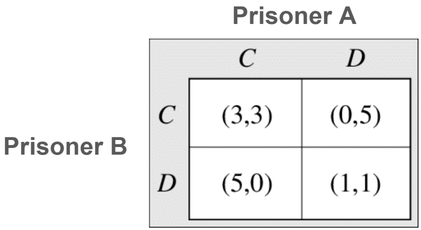
RL Application in Economics

Game Theory - Prisoner's Dilemma

Reference: Tobin (2025) - The University of Michigan

- Two prisoners are put the two in separate cells
- Each one could cooperate (C) or defect (D)

- If both cooperate (C), each gets 3 coins
- If one defects (D) and another cooperates (C), then the one who defects get 5 coins, and another 0
- If both defect (D), each gets 1 coins
- The goal is to earn as many coins as one can



Prisoner B \ Prisoner A	*C*	*D*
C	(3,3)	(0,5)
D	(5,0)	(1,1)

What would you do? No matter what your opponent does, your best option is to always defect (D).

Your best option is to always D...with a **single** prisoner's dilemma.

Another question to ask is ...

What if we play the game of prisoner's dilemma many times? i.e., what if your opponent use your last move against you in the future? **What is the best strategy in this repeated games?**

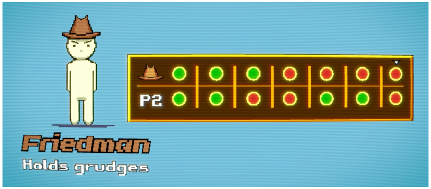
'The Prisoner's Dilemma captured the essence of the tension between doing what is good for the individual and what is good for everyone.' - Robert Axelrod

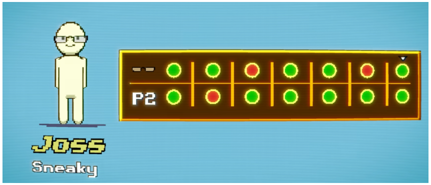
Axelrod’s Tournaments

Prof. Robert Axelrod, a political scientist, wanted to find out about the best strategy for the repeated prisoner’s dilemma. He started tournaments in which game theorists sent him their computer programs (their strategies). Note that the following content is from Veritasium (2023) YouTube video.

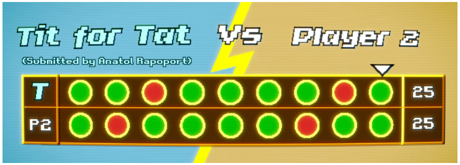
- **First Tournament in 1980s**
  - 200 repeated games
  - All strategies compete head-to-head against each other and against itself
  - Goal: Which strategy earn the highest coins?

- **First Tournament in 1980s**
  - In total, Axelrod received 14 strategies.
  - Some strategies are ....





Veritasium, 2023

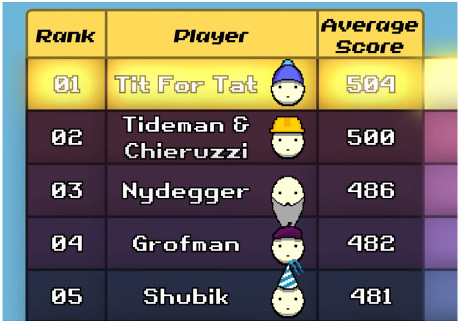


Veritasium, 2023

- And other strategies such as
  - **Graaskamp** works the same as Joss, but instead of defecting probabilistically, Graaskamp defects in the 50th round to try and probe the strategy of its opponent and see if it can take advantage of any weaknesses
  - **Name Withheld** provides the most elaborate strategy with longest lines of code

Guess who is the winner?

Guess who is the winner?



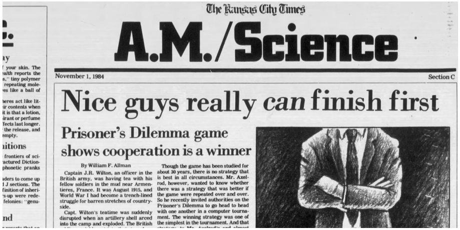
Rank	Player	Average Score
01	Tit For Tat	504
02	Tideman & Chieruzzi	500
03	Nydegger	486
04	Grofman	482
05	Shubik	481

Veritasium, 2023

After the first tournament, Axelrod found that the top 4 winners share these traits:

- Nice (not the first to D)
- Forgiving (retaliate (fight back) but not holding grudges). Return defection for defection, cooperation for cooperation.
- Don't be envious: focus on maximizing your own 'score', as opposed to ensuring your score is higher than your 'partner's'
- Don't be too clever: or, don't try to be tricky. Clarity is essential for others to cooperate with you.

Be nice. Be ready to forgive. But don't be a pushover.


The Kansas City Times

A.M./Science

November 1, 1984 Section C

Nice guys really *can* finish first

Prisoner's Dilemma game shows cooperation is a winner

By William F. Allman

Captain J.R. Wilton, an officer in the British army, was having tea with his fellow soldiers in the mud near Armentieres, France. It was August 1915, and World War I had become a trench-lined struggle for barren stretches of countryside.

Capt. Wilton's teatime was suddenly disrupted when an artillery shell arced into the camp and exploded. The British

Though the game has been studied for about 30 years, there is no strategy that is best in all circumstances. Mr. Axelrod, however, wanted to know whether there was a strategy that was better if the game were repeated over and over. So he recently invited authorities on the Prisoner's Dilemma to go head to head with one another in a computer tournament. The winning strategy was one of the simplest in the tournament. And that

ay

your skin. The
ealth reports the
s," tiny polymer
repeating mole-
es like a ball of

heres act like lit-
ir contents when
it is that a lotion,
irant or perfume
fects last longer.
the release, and
empty.

nitions

frontiers of sci-
actured Diction-
phonetic pranks

aders to come up
J sections. The
finition of inheri-
s-up were rede-
felonies: "genu-

nd

Veritasium, 2023

In this tutorial, we will reproduce *Axelrod's Iterated Prisoner's Dilemma Tournament* using RL.

- Instead of hard-coded rules (*AlwaysC*, *AlwaysD*, etc.), each agent will **learn** how to act through interactions
- We will use two TD algorithms:
  - **Q-Learning** Off-Policy
  - **SARSA** On-Policy
- The goal: discover whether RL agents can learn the same cooperative patterns that emerged naturally in Axelrod's original tournament.

Can an agent “learn to be nice”? Or will it learn to exploit? Let’s find out. Form four groups to design your agent’s strategy:

- Two groups will implement **Q-Learning** agents.
- Two groups will implement **SARSA** agents.

Innate nature of these algorithms:

Q-Learning - *Off-Policy TD* : A bold and idealistic learner. “I learn from what I should have done, not necessarily from what I actually did.”

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

SARSA - *On-Policy TD*: A cautious and honest learner. “I learn from what I did, not what I could have done.”

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

On Moodle, you will find

- `base_tournament.py` which contains the rules of the game and base opponents (*AlwaysC*, *AlwaysD*, *Tit-for-Tat*, and *Random*). No edits needed on this file. It must remain as-is.
- `q_student_agent.py` which is a template for **Q-Learning** groups. Edit this file to implement and test your Q-learning strategy.
- `sarsa_student_agent.py` which is a template for **SARSA** groups. Edit this file to implement and test your SARSA strategy.

On `q_student_agent.py` and `sarsa_student_agent.py`. You can fine tune parameters and logic of their learning agent to explore different behaviors.

Possible Modifications: Hyperparameter tuning

- Learning rate `alpha`: Controls how quickly your agent updates its knowledge.
- Discount factor `gamma`: High gamma = long-term thinking, Low gamma = short-term greedy
- Exploration rate `epsilon`: Controls randomness in decision-making

Possible Modifications for Q-Learning

1. Decaying exploration (ϵ-greedy): i.e., Encourages initial mixed strategies that gradually (possibly) converge toward Nash equilibrium.

```
self.decay_rate = 0.995
```

2. Extend state representation: Incorporate memory, like a ”tit-for-tat” strategy i.e., “forgive, but remember what the opponent did in the last two moves.”

```
self.state = 2 if opponent_last is None else (0 if opponent_last == 0 else 1)
```

3. Modify update rule (momentum): This introduces risk sensitivity and adaptive flexibility. The agent becomes more responsive to unexpected outcomes

```
self.Q[self.state, self.last_action] += self.alpha * (td_error + 0.1 * td_error**2)
```

Possible Modifications for SARSA

1. Decaying exploration (ϵ-greedy): i.e., Encourages initial mixed strategies that gradually (possibly) converge toward Nash equilibrium.

```
self.decay_rate = 0.995
self.epsilon = max(0.01, self.epsilon * self.decay_rate)
```

2. Extend state representation: Incorporate memory, like a "tit-for-tat" strategy i.e., "forgive, but remember what the opponent did in the last two moves."

```
self.state = 2 if opponent_last is None else (0 if opponent_last == 0 else 1)
```

3. Modify update rule (e.g., eligibility trace): This introduces risk sensitivity and adaptive flexibility. It makes adaptive players who adjust based on their own behavior and outcomes

```
td_target = reward + self.gamma * self.Q[next_state, next_action]
td_error = td_target - self.Q[self.state, self.last_action]
self.Q[self.state, self.last_action] += self.alpha * td_error
```

The Class RL-2025 Tournaments:

- Round 1: Standard as in Axelrod's Tournament (round=200 with noise=0.01)
- Round 2: What if we live longer (round=1000 with noise=0.01)
- Round 3: What if there are miscommunications (round=1000 with noise=0.5)

Once your modifications are done, send your file to me via email.

6.7 Downsides of Temporal Difference Learning Methods

TD agents

- Only for discrete actions
- When state and action spaces are high dimensions, we need to implement Deep Neural Network to RL