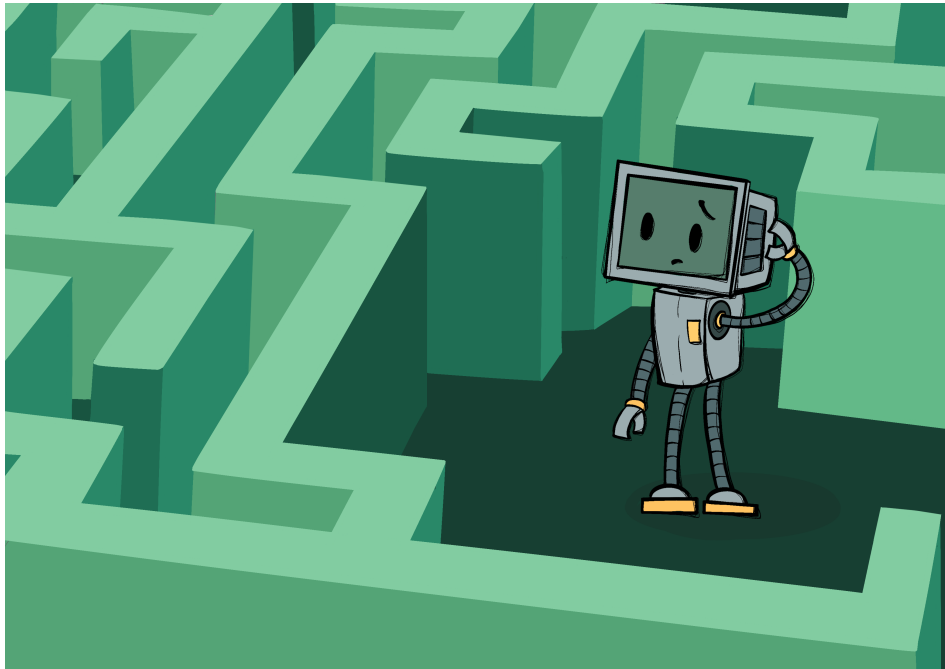


Module 5

Reinforcement Learning

*"By using a **maze** in which a **robot** is rewarded by finding a **battery**, this exercise demonstrates that RL is not as straight forward as one might think."*



About the Module

This module is about **Reinforcement Learning (RL)**. The goal is to provide the students with a basic understanding of what **RL** is, how it works and what common problems and pitfalls are. The focus lies more on practical exercises, therefore students will on the one hand take the role of self learning algorithms and experience how the training process works, and on the other hand play against and train learning **AI**s.

Objectives

Students will be able to...

- ...explain the basic idea of **RL**
- ...understand the core **RL**-loop and decisions based on Q-values
- ...estimate if an application is using **RL**
- ...name problems and limitations of **RL**

Agenda

Time	Content
30 min	Introduction
20 min	Exercise - Coin Game
20 min	Exercise - Hexapawn
20 min	Exercise - Menace
30 min	Exercise - Maze
15 min	Quiz - True or False?

Introduction

The Introduction is all about students becoming familiar with basic concepts and terms used in **Reinforcement Learning (RL)**. It is based on the slides, which introduce not only common applications but the general idea of how an **AI** can **learn by itself** by performing and evaluating actions.

Real World Examples

(Slides 2 - 6)

The slides start by introducing **RL** using a short video about a self learning **hide and seek AI** of OpenAI. If the students don't understand english well enough, either auto translated subtitles can be used or the teacher can take care of translating/explaining what happens.

The main point to bring across is that in **RL** the **AI** learns by itself, without direct guidance of the programmers. The agents in the video were never told **how** to do things, only **what** they can do and what their goal is.

After the video, the next example is **Leela Chess Zero (Lc0)**, a modern chess **AI**, trained using reinforcement learning. While traditional chess **AIs** had static functions to evaluate board positions (like getting fixed points for material differences and piece positions), **Lc0** learned what good positions are by playing millions of games against itself and other players, and therefore can make better decisions on what actions to take. It is important to state that modern chess engines are far superior to human players, even the best grandmaster players can be beaten using a smartphone.

The reason why games are quite popular in **RL** research is, that there are well defined rules which makes it easier to define concrete actions and goals, which are needed for **RL** to work. This will be explained in more detail after the example slides.

As chess engines still mostly calculate moves in a similar way they did in the late 90s, when they started to beat human players consistently³, they sometimes are downplayed due to the 'simplicity' of the game. Still **RL** algorithms evolved and nowadays can go toe to toe to human players in much more complex scenarios. Therefore, the third example is about **AlphaStar**, an **AI** from Google's Deepmind research department, which has learned to play a complex modern computer game up to a level comparable to the best human players.⁴ Given that there are

roughly 10^{26} legal actions at every point in time (compared to chess with an average of less than 40), it is astounding that the **AI** can choose the best ones while adhering to the same limitations a human player has (limited vision, reaction time).

The final example is a more practical one about custom advertisement (Ad). It demonstrates, that **RL** can be used in many different ways, in this case as a revenue optimization by learning which Ads are most clicked (or hovered over) and adapting future ads displayed to maximize the chance the user will click on an Ad.

There are many more examples of use cases for **RL**, like training self driving cars using simulations, but these examples are a good enough starting point to now tackle the question, how an algorithm can actually learn by itself.

RL Basics

(Slides 7 - 20)

To be able to understand the basics of **RL**, one must know at least some definitions, which are introduced in the first few slides. Then the game TicTacToe is used as an example

Agent

An agent describes the intelligent entity that interacts with the environment and has to make sensible decisions. This is the part of a program that incorporates all the **AI**'s logic and knowledge.

In TicTacToe the agent would be the **AI** player.

Environment

For an agent to learn it needs a clearly defined environment. Usually the environment is some kind of simulation of a real environment (like a simulated street for a car to learn driving) or a game.

In TicTacToe the environment is the game board with the drawings (X,O) on it.

Action

At each point in time, the agent has to decide which action to take. The available actions must be known at any time, the **AI** then has to learn to choose the best one.

In TicTacToe every empty tile corresponds to an action of drawing ones symbol (X,O) on the tile.

State

A state is a numerical representation of meaningful parts of the environment. As computers only work with numbers, critical parts of the environment can be abstracted to be understandable to the **AI**.

In TicTacToe the state could be a number for each of the nine tiles (e.g. 0 = empty, 1 = X, 2 = O) and maybe also a number of whose turn it is (1 or 2). In another example, the coin game, which will be used in the following exercise, the state is simply the number of coins on the table.

This is usually the most abstract of the definitions, it will become clearer for most people during the next exercise.

Reward

The reward is generally a numeric value that depicts the quality or outcome of a state. Usually it is positive in case of a good result and negative in case of a bad result.

In TicTacToe the reward could be +1 for winning, -1 for losing and 0 for everything else.

Finally, the students have to think about what agents, states, actions and rewards are in the following examples. This can be approached in multiple ways ranging from a simple group discussion, to presentations in small groups, the **methods page** contains more ideas.

Leela Chess Zero

The **agent** is the chess **AI**, the **state** consists of numerical values representing the pieces on the board. **Actions** are the possible chess moves and the **reward** can be a very high/low number for winning/losing and some numbers in between for how 'good' the position is for the **AI**.

OpenAI Hide and Seek

The **agents** are the hiders and seekers, the **state** includes the position of the agents as well as of all objects and if they are locked or not. **Actions** include moving in different directions, rotating and grabbing/releasing as well as locking objects. The **reward** can be the number of seconds the hiders were not found which can also be used as a negative number for the seekers.

Custom Advertisement

The **agents** could be individual advertising spots, the **state** information about the type of advertising and if the user clicked on it or not. **Actions** could include switching to specific topics or to advertise or types of

advertisements. The **reward** could be a number indicating if the user clicked on the add or not.

In reality finding a good representation for the state and available actions is not an easy task, but a very important one. Choosing badly can result in increased training time and poorer performance, or even stop the agent from learning all together as crucial information might be missing. This is not too relevant in the upcoming exercises however, as there the **states** and **actions** are clearly defined already.

Q-Learning

(Slides 21 - 30)

The final slides introduce the concept of **Q-learning**, as well as the **RL interaction loop**. It is based around the concept of action and reaction, where the agent interacts with the environment by choosing an action to perform. This action changes the environment and as a result the agent receives the representation of the new state as well as a reward indicating how good the action was. Given this new information the agent then might update its behavior and then chooses a new action to perform. This cycle continues until a goal is reached.

Q-learning is based on the **RL** interaction loop and stores a **quality value** (Q-value) for each pair of **state** and **action**. This value indicates how 'good' the action is given the current state. Therefore, whenever a state is reached, the action with the highest **Q-value** can be chosen for optimal behavior. As the **Q-value** might not be correct at the beginning (in fact, quite often it is set to a random value for each action at the beginning), it can be adapted by increasing/decreasing it in regards to the reward received for performing the action. When this process is repeated often enough, the **Q-values** will reach a point, where always the correct action will be performed.

As long as there are not too many state-action-pairs, this values can be stored in a table where each row consists of a **state** and the **Q-values** of all available actions.

The table on slide 29 shows a small section of such a Q-table for an exemplary jump-n-run game. Reaching a diamond is rewarded with +1 and falling into the water with -1. The next table on slide 30 shows the same situation, but with a more advanced table which includes future rewards into their action. This means an

action that might result in falling into the water one action later will also get a small penalty.

This process of adapting the **Q-values** is called **training**. While it sometimes is feasible to train an **AI** during normal use, quite often the training process is independent from the final use, as it can take millions of iterations to create meaningful values.

After all this theory it is time to test the knowledge on real examples, which leads to the next three exercises, which build on each other.

Material

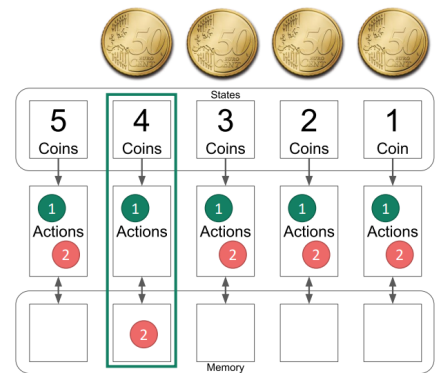
-  RL - Introduction.pdf

References

1. <https://openai.com/blog/emergent-tool-use/>
2. <https://lczero.org>
3. <https://www.chess.com/article/view/deep-blue-kasparov-chess>
4. <https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii>

Coin Game

This exercise provides an introduction into **RL** learning using the **coin game**. To help understand the basic concept, **slides** can be used as an introduction. The game requires some kind of tokens like coins, cones, treats, ... in a large quantity, so every pair of students has access to at least 5 of them. The game starts by having 5 coins on a table and the players alternately taking one or two of them away. The player taking the last token loses.



1. Introduce the game

Introduce the game using the slides 1-8 and let the students play a few matches in pairs.

2. Introduce the AI

Introduce the **AI** using the slides 9-35. It is recommended to provide the material (**board and actions**) beforehand, so the students can see how everything looks like in reality and also are able to actively play along with the examples.

3. Let them play

Then let the pairs of student play, with one taking the role of the **AI**, and the other playing normally. As the games continue more and more actions will be removed from the **AI** until nothing but the winning actions are left.

4. Key takeaways

Finally talk about the following key takeaways:

What happens, if some states are never reached (e.g. the player always chooses the same actions)?

Only states that are reached are learnt from. When the **AI** does not encounter a situation during training, it might not find the correct move.




In this example, where does the AI learn more, by winning or by losing?

By losing, in this example the **AI** is only punished for losing, never rewarded for winning, therefore winning has no effect.

Could you change the game so that a win also provides some kind of reward?

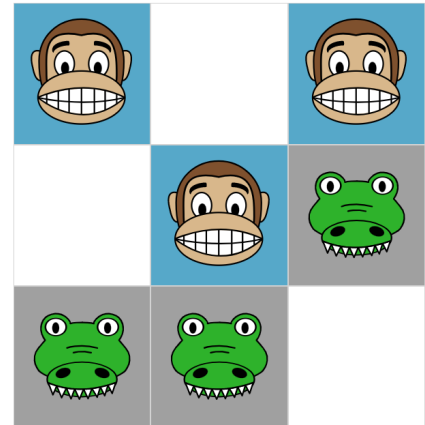
Yes, for example by adding new action-stones when the **AI** wins. Then the **AI** will more likely select one of the more successful actions. This will be demonstrated in the next exercise.

5. Material

-  RL - Coin Game.pdf
-  RL - Coin Game Board.pdf
-  RL - Coin Game Actions.pdf

Hexapawn

This exercise build on the knowledge of the previous one (coin game) and is based on the website <https://www.stefanseegerer.de/schlag-das-krokodil/> and the game **Hexapawn**. In **Hexapawn** two players play against each other on a 3 by 3 board using only pawns from chess, which can either move one field straight forward or capture an enemy pawn diagonally. A player wins by either reaching the opposite side with one pawn or by preventing the enemy to make a move (blocking all pawns).



1. Demonstrate the website

First, demonstrate the capabilities of the website and explain the rules of the game. Especially the settings **Response Time** and **Only possible moves** should be explained, as they help in the following tasks. Furthermore it is vital that students understand the meaning of the coloured points (correspond to same coloured action) and see that these points can be removed or added.

2. Let them play

Now it is time to let the students play on their own. The goal is to win as often as possible, before the **AI** cannot be beaten. This seems like a simple task, but soon students will realize they need a good understanding of the inner workings to achieve over 10 or even over 20 wins. **Be careful, when the site is reloaded, the wins will also reset!**

3. Key takeaways

The key takeaways are similar to the coin game, but more pronounced, as students have to explicitly take actions they have not been used before, to force the **AI** into unknown territory. It can also be seen, that the number of possible states increases quite fast with the number of available actions. It can easily be envisioned, that on a bigger board (like a chessboard), there are so many

possible states (roughly 10^{44})¹ that it is not feasible to train an **AI** by hand, or even include all possible states.

When the number of possible states becomes too big, other methods have to be used like reducing the number of states by using evaluation functions² to exclude actions that lead to undesirable states or approximating Q-values using neural networks³.

4. Material

-  <https://www.stefanseegerer.de/schlag-das-krokodil/>

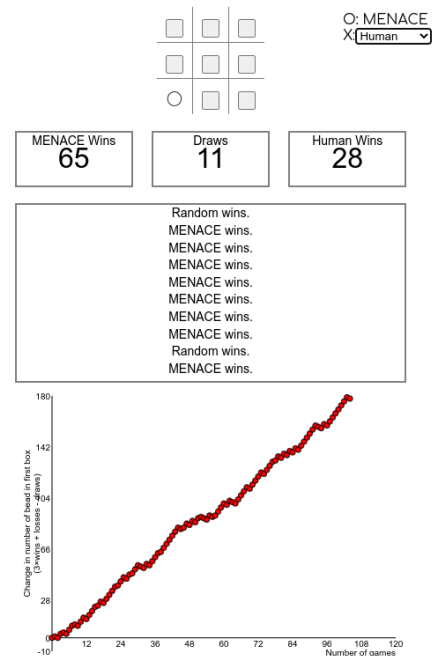
5. References

1. <https://tromp.github.io/chess/chess.html>
2. <https://www.chessprogramming.org/Evaluation>
3. <https://www.turing.com/kb/how-are-neural-networks-used-in-deep-q-learning>

MENACE

This exercise again builds on the previous one (Hexapawn) and introduces a game with even more states, TicTacToe. As a good introduction the first part of this video from Matt Parker about MENACE can be used. It explains the basic idea of MENACE, which is similar to the Hexapawn example from before, but instead of working digitally, they store coloured beads inside physical match boxes.

While it might be an interesting idea to create such a system with the class, it will probably take quite some time, not only to create, but also to train to get meaningful results. Therefore, in this exercise an online simulator of the system will be used.



1. Introduce MENACE

First, introduce the MENACE system, a **RL** TicTacToe machine, either by using the first ~2:20min (or more) this video from Matt Parker or by discussing (or group work) how the previous example could be modified to fit the game of TicTacToe.

2. Demonstrate the website

Then demonstrate how to use the simulator on the website <https://www.msccroggs.co.uk/menace/>, explain especially the meaning of the numbers and systems to the right (states and q-value of actions, higher = more likely to be chosen, mirrored states are excluded) and how you can play games yourself or let the **AI** play against other **AIs** (like the random playing **AI** or a perfectly playing **AI**).

3. Let them train their own AI

Given the knowledge of how to use the website, the students now have to train their own **AI**, which should be as good as possible.

4. Talk about the Problems

Soon the students will realize, that even if this sounds like an easy exercise, probably none of the approaches will lead to a perfectly playing **AI**. The main reasons why this occurs are:

- The **AI** simply doesn't encounter some states during training, so that it doesn't know what a good move is when it encounters them. This happens for instance if it is only trained against a perfectly playing **AI**, where it never learns that it can actually win, so it will not know how to exploit mistakes when they are made.
- The **AI** learns wrong moves, because the enemy didn't respond properly. For example it might have learned that a move usually leads to victory, because the enemy did only bad (or random) moves, but when the enemy plays correctly, the **AI** will lose.

The technical problem behind this is exploration vs exploitation¹, which will be explored more in the next exercise.

5. Material

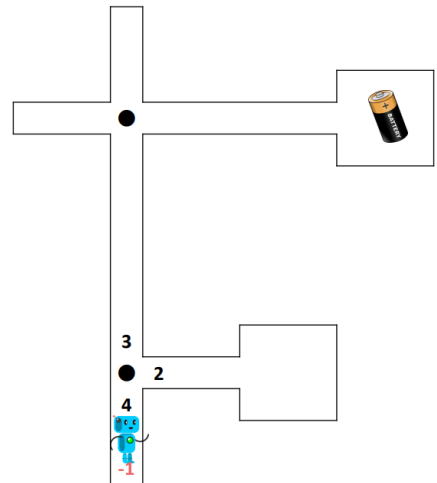
-  https://youtu.be/R9c-_neaxeU
-  <https://www.msccroggs.co.uk/menace/>

6. References

1. <https://ai-ml-analytics.com/reinforcement-learning-exploration-vs-exploitation-tradeoff/>

Maze

By using a **maze** in which a **robot** is rewarded by finding a **battery**, this exercise demonstrates that **RL** is not as straight forward as one might think. The exercise starts with **this slides**, explaining the scenario of a robot inside a maze which has to find a battery. In the beginning the robot will take random turns and over time it will learn the correct path to its goal. In the second exercise, the maze has multiple batteries with different values, which will lead to sub-optimal results, as explained in the last few **slides**.



1. Introduce the game

First, introduce the students to the game using the **slides 1-18**. Then provide each student (or pair of students) with a copy of the **RL Maze Basic**, a dice and a pen.

2. Let them play

Then the students have to play multiple rounds, until the path to the battery is clearly defined. To update values, one can simply cross it out and write the new value close to it. Some students will be 'lucky' and the path will form very fast, while with other the robot will go into every dead end before it reaches its goal. If some students are way too fast, they can try again with a new sheet of the same maze and see how it behaves differently the second time.

3. Introduce the advanced maze

Provide each student (or pair of students) with a copy of the **RL Maze Advanced** and again let them train their robot. Even if there are multiple batteries this time, the robot still goes back to the beginning whenever it reaches a battery. The game is over again, if the robot has a clear path to one of the batteries, even if it is not the one with the highest value.

4. Find out the problem and propose solutions

When everyone is finished, different robots will have reached different scores. Now create groups of around 4 students and let them discuss why some robots scored better than others and work out a possible solution. It might help to group students with different robot scores together, so they have a better visualization of the problem. Then each group has to present their findings and solutions.




5. Explain the problem

Finally, use slides 19–26 to discuss the exploration vs exploitation trade-off¹ and possible solutions for this exercise.

6. Optional: Test various solutions

Provide new copies of the **RL Maze Advanced** so that the groups can test various new approaches (like different exploration rates).

7. Material

-  RL - Maze.pdf
-  RL - Maze Basic.pdf
-  RL - Maze Advanced.pdf

8. References


1. <https://ai-ml-analytics.com/reinforcement-learning-exploration-vs-exploitation-tradeoff/>

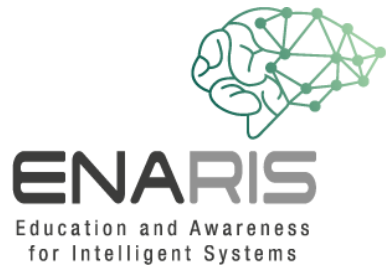
True or False?

This last chapter acts as a **recapitulation** of the whole module. It is based on a **quiz with ten true or false questions** and encourages students to think about all the aspects they have learned.

The quiz itself consists of **slides**, showing one question after another before showing the correct answers. Therefore, students can write down their responses and then calculate their score in the end. It is highly encouraged to pause after revealing the answer for every questions to discuss briefly **why** it is correct or wrong. Alternatively, this exercise can also be done in groups to further encourage discussion between students.

Material

-  RL - Quiz.pdf



EUROPEAN UNION

