Machine Learning Project 2022-2023

Multi-Agent Learning in Canonical Games and Dots-and-Boxes (OpenSpiel)

Giuseppe Marra, Wannes Meert

February 2022

Read this text completely. Deviations from the instructions may result in lower scores.

1 Introduction and Related Literature

We live in a multi-agent world and to be successful in that world, agents, and in particular, artificially intelligent agents, need to learn to take into account the agency of others. They will need to compete in marketplaces, cooperate in teams, communicate with others, coordinate their plans, and negotiate outcomes. Examples include self-driving cars interacting in traffic, personal assistants acting on behalf of humans and negotiating with other agents, swarms of unmanned aerial vehicles, and robotic teams.

This assignment covers topics in multi-agent reinforcement learning (RL). We assume elementary knowledge of single-agent reinforcement learning (a good reference when less familiar with RL is Sutton and Barto [11]). When moving from single-agent RL to multi-agent RL (MAL), Game Theory plays an important role as it is a theory of interactive decision making. Throughout the assignment you will use some elementary game theoretic concepts in combination with multi-agent learning, which is non-stationary and reflects a moving target problem (see [9] for basic concepts about game theory when less familiar).

In this assignment we start by tackling some canonical games from the 'pre-Deep Learning' period. To learn how (multi-agent) reinforcement learning and game theory relate to each other, you will work with tabular RL methods using ε-greedy and Boltzmann exploration [13, 5] and express them using replicator dynamics [4]. Afterwards we move to the Dots-and-Boxes game, first using the minimax algorithm from game-theory [9] and second using RL and machine learning (for some Deep RL references see [8, 6]).

We will be working in the OpenSpiel open-source software framework.[1] OpenSpiel is a collection of environments, algorithms and game-theoretic analysis tools for research in general reinforcement learning and search/planning in games. OpenSpiel supports n-player zero-sum, cooperative and general-sum games [7]. Learning how to use advanced, state-of-the-art software toolboxes for AI is part of the project and we expect you to explore the documentation (including docstrings, and code example files).

2 Approach

You will work on this project in a team of two or three students. Note that since a lot of code is available within the OpenSpiel framework we do expect students to show a good understanding of the techniques deployed, and be able to conduct a knowledgeable conversation about the techniques deployed during the defence of the project. We expect that your code runs on the computers available at the Department of Computer Science, and thus can participate in the tournament.

[1] https://github.com/deepmind/open_spiel

Please direct any questions that you may have about the project to the Toledo forum or the classroom discussion moments such that all students can benefit from the discussion.

3 Deadlines

3.1 Form Groups — **Before March 5th, 23:59**

Mail to **both** wannes.meert@cs.kuleuven.be and giuseppe.marra@kuleuven.be whether you work on this project in a team of two or three and include the names of all team members (one email per team).

3.2 Submit Draft of Report (optional) — **Before March 24th, 23:59**

If you submit a draft report of tasks 1 and 2, (high-level) feedback will be provided individually.

3.3 Submit Report and Prototype — **Before May 17th, 23:59**

Submit your report (PDF, ≤10 pages) to Toledo. Your report should fulfill the following criteria:

- Mention the directory on the departmental computers where your code and agent are stored.
- Clearly state the **problem statement**.
- Formulate your design choices as **research questions** and answer them.
- Write out the **conclusions** you draw from your experiments together with a scientifically supported motivation for these conclusions.
- Be concrete and precise about methods, formulas and numbers throughout the text. A scientific text should allow for **reproducibility**.
- Clearly **cite** all your sources.
- Report the total **time each of you spent** on the project, and how it was divided over the different tasks mentioned.

3.4 Peer assessment — **Before May 17th, 23:59, individually**

Send by email a peer assessment of your partner's efforts. This should be done on a scale from 0-4 where 0 means "My partner did not contribute", 2 means "I and my partner did about the same effort", and 4 means "My partner did all the work". Add a short motivation to clarify your score. If you are in a team of more than two people, send a score for each partner. This information is used only by the professor and his assistants and is not communicated further.

3.5 Oral discussion — **Week of May 22nd**

Pick a slot on Toledo to discuss your report and answer questions.

4 Tasks

Your report should discuss the following tasks (mention the task numbers).

[*The final mark per task is determined by the combination of the report and the oral discussion*]

		Player 2		
		R	P	S
Player 1	R	0	-0.25	0.5
	P	0.25	0	-0.05
	S	-0.5	0.05	0

		Player 2	
		O	M
Player 1	O	$3, 2$	$0, 0$
	M	$0, 0$	$2, 3$

		Player 2	
		A	B
Player 1	A	$-1, -1$	$1, 1$
	B	$1, 1$	$-1, -1$

		Player 2	
		C	D
Player 1	C	$-1, -1$	$-4, 0$
	D	$0, -4$	$-3, -3$

Figure 1: Biased Rock-Paper-Scissors Game, Dispersion Game, Battle of the Sexes and Prisoners Dilemma

Task 1: Literature Study

Describe briefly what **literature** you have read and what you have learned from it. You are expected to at least read the sections about the concepts mentioned in the assignment. They are all explained in the references. This is not just a list of papers, but a critical analysis of the existing work and how your project relates to it, especially those works which influenced your design decisions.[2]

[*With this task you can earn 1 out of 20 points of your overall mark.*]

Task 2: Learning & Dynamics

Matrix Games

Here we learn how to play four benchmark matrix games: Biased Rock-Paper-Scissors Game, Dispersion Game, Battle of the Sexes and the Prisoners Dilemma (see Figure 1 for the payoff tables used in this assignment). Each of these games belongs to a category of games, i.e. social dilemma, zero-sum or coordination game. Zero-sum games correspond to situations in which the total sum of gains and losses of all players involved is equal to zero; social dilemmas are known as games in which several agents participate and there is a tension between what is optimal to do at the individual level (in the short term) and what is beneficial to do at the group-level (in the long term). Famous examples of the former include the *Rock-Paper-Scissors* game and *Poker*, and of the latter include the *Prisoner's Dilemma* game and the *Tragedy of the Commons*.

Goal

Independent learning in benchmark matrix games: Implement/use basic reinforcement learning algorithms and experiment with the four benchmark games. Include at least ε-greedy and Lenient Boltzmann Q-learning. Investigate and report on whether the learning algorithms converge to Nash equilibria, and whether these are Pareto optimal (note that both agents should be using an RL algorithm; this can be self-play where both agents use the same RL algorithm). Plot the empirical learning trajectories.

Dynamics of learning in benchmark matrix games: Start by implementing/using the basic form of replicator equations (selection mechanism only) and plot the directional field plots for the same games. Compare these to the learning trajectories (you may plot the field plots and trajectories together). Figure 2a illustrates an example phase plot of replicator dynamics in the matching pennies game, with the Nash equilibrium at $(0.5, 0.5)$ (learning trajectory is not shown in this example). Now make your own implementation of 2-player lenient Boltzmann Q-learning dynamics in OpenSpiel, and do the same analysis, tune the involved parameters and report about their values. You can find the dynamics of lenient Boltzmann in [4].

[2] http://www.writing.utoronto.ca/advice/specific-types-of-writing/literature-review

You should be able to answer questions about what the plots show and why certain behavior occurs.

[*With this task you can earn 8 out of 20 points of your overall mark.*]

Task 3: Minimax for small Dots-and-Boxes

The Dots-and-Boxes game

Dots-and-Boxes is a popular combinatorial game that can be played with only paper and pencil. It is played on a rectangular grid of dots that represents $r \times c$ squares ("boxes"). The two players take turns in drawing a horizontal or vertical line between a pair of neighboring dots. If a player closes a box, that box is assigned to that player and she must draw another line. The player who owns the most boxes wins the game.[3] While the game has simple rules, it has a large number of possible states (more specifically, $|S| = 2^{r(c+1)+(r+1)c}$ states), and an even larger number ($\log_2(|S|)!$) of unique games can be played. No general strategy for optimal play is known. Currently, a 4×5 game is the largest game that is solved [1].[4]

Goal

Solve a tiny version of Dots-and-Boxes using the minimax algorithm. The minimax algorithm is a basic algorithm in game theory to decide what is an optimal move for the current player. By backtracking the entire tree of possibilities, the minimax algorithm computes, for each state, a value indicating how good it would be to reach that position. You can start from a naive template available in the repository. You will notice that this game has a very wide game tree making minimax already slow for tiny grids. Your task is to optimize the provided template for the minimax algorithm. Such optimization requires two steps:

1. *Implement transposition tables for the minimax algorithm.* Transposition tables are a standard technique to improve the minimax algorithm. The intuition is that the same state can be reached by different trajectories. Caching the value of a state will then avoid re-searching the game tree below that same state along a different trajectory. Pay particular attention to *what* is a state in Dot-and-boxes and how to represent it (i.e. what is the key you will use for the cache?).

2. *Exploit symmetries.* The previous improvement exploited an equivalence among different trajectories to improve the search. Now, we want to exploit the fact that multiple states are actually completely symmetric. Think for instance of two states, which are one the rotation of the other. They will necessarily share the same value as one can simply play the same strategy with the actions rotated accordingly. By exploiting such symmetries, you can avoid re-searching states that are symmetric.

Use these two optimizations and compare with the provided template algorithm. For various game board sizes, plot both the execution times and the size of the transposition tables and the number of keys.

[*With this task you can earn 4 out of 20 points of your overall mark.*]

Task 4: Full Dots-and-Boxes

Learning strategies

The Dots-and-Boxes games used in the previous section are too small to be interesting to play. In this task you are expected to find a more advanced strategy that works for any size of game. The following

[3] More details and the history of the game can be found on `https://en.wikipedia.org/wiki/Dots_and_Boxes` and `http://wilson.engr.wisc.edu/boxes/`.

[4] "Solved" here means that an optimal strategy for each player is known; this implies that it is known whether the first player to move will win, lose, or draw, if both players play optimally.

approaches are some starting points:

1. **Reinforcement Learning** (RL) [3, 11, 12] can be used to learn a policy for playing games. Often, given the very large number of state-action combinations, standard model-free techniques such as Q-learning, with a tabular representation of state-action pairs will not work; instead some kind of generalization is needed (predicting the Q-value of unseen state-action pairs), often done using deep learning. There will be a need to design or learn features describing states and actions that correlate with the quality of state-action pairs.

2. **Monte Carlo tree search (MCTS)** [14] is a stochastic version of minimax search. It has been found to work better than standard minimax with α-β pruning. Many modern game-playing programs use it. Contrary to reinforcement learning, MCTS explores the states space during play. Like minimax, MCTS may use an evaluation function V that models how promising a state is (e.g., in AlphaGo they used deep learning to represent this function [10]).

For all approaches you can use implementations from OpenSpiel (DQN, policy gradient, MCTS, etc.) or make implementations of your own. For all strategies you will need to represent states and state-action pairs. Simple games such as matrix games are stateless models, but for more complex settings you will need to use models that can generalize states. To this end, you will have to think about features that can accurately represent the game state and that allow for learning models that generalize well. The features can be designed by an expert, or learned automatically. The Q or V function will be expressed in terms of these features. Learning this function can be done using almost any of the machine learning techniques you have seen in the machine learning course.

Goal

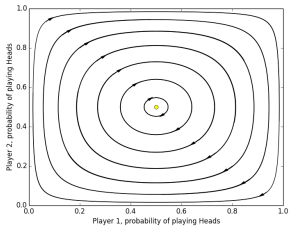
You will develop an agent that has learned to play Dots-and-Boxes, making use of the code infrastructure provided in OpenSpiel. Your agent should work for any board size. If this is too challenging, you can train an agent that can play a 7x7 board (this reduces your maximal score by 2 points). You are free to use any technique you prefer (e.g., deep Q-learning, Monte-Carlo tree search, minimax).

In the case of Dots-and-boxes it won't be possible anymore to evaluate your agent exactly with respect to Nash optimality or other metrics due to the size of the game. Therefore, we expect you to evaluate your agent head-2-head against simple baselines including but not limited to random play, first open edge, ... Use the results of your evaluation and literature to motivate your design choices. Being able to run and experiment with your agent (and for us to be able to reproduce this) is part of the evaluation. Thus explicitly describe your model (e.g., network structure, input tensor and exploited symmetries), what types of gameplay you observe (e.g., does your model learn to recognize the concept of chains, a popular strategy in Dots-and-Boxes [2]), and discuss the learning statistics you used.

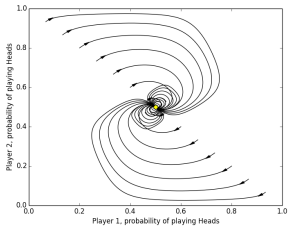
Tournament

For the final evaluation of your agent we will play a round-robin tournament, in which each agent will play many games (pairwise) against all other agents. The tournament is played with all submitted agents and a range of standard baseline agents. This will be used to assess that your agent knows how to play the game. Two tournaments will be played: one on a 7x7 board, one on a non-disclosed board size.

To participate in the tournament, a copy of your code and agent needs to be available in a directory on the departmental computers that will be provided and your agent needs to follow a predetermined template. See `https://github.com/ML-KULeuven/ml-project-2022-2023` for technical instructions. Include all the code you used and have written to perform the experiments and explain how to



(a) Evolutionary dynamics of the Matching Pennies game.



(b) Average Time Evolutionary dynamics of the Matching Pennies game.

Figure 2: Example of dynamics

use it in a readme file. If you work in a team, choose the directory of one team member. Test your (preliminary) code as early as possible on the departmental computers. An implementation that does not run on the server reduces your score.

[*With this task you can earn 7 out of 20 points of your overall mark.*]

References

[1] Joseph Kelly Barker and Richard E Korf. "Solving Dots-And-Boxes." In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. 2012.

[2] Elwyn R Berlekamp. *The Dots and Boxes Game: Sophisticated Child's Play*. AK Peters/CRC Press, 2000.

[3] Hendrik Blockeel. *Machine Learning and Inductive Inference (course notes)*. KU Leuven, 2017.

[4] Daan Bloembergen et al. "Evolutionary Dynamics of Multi-Agent Learning: A Survey". In: *J. Artif. Intell. Res. (JAIR)* 53 (2015), pp. 659–697.

[5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A Comprehensive Survey of Multiagent Reinforcement Learning". In: *IEEE Trans. Systems, Man, and Cybernetics, Part C* 38.2 (2008).

[6] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[7] Marc Lanctot et al. "OpenSpiel: A Framework for Reinforcement Learning in Games". In: *ArXiv* abs/1908.09453 (2019).

[8] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[9] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009. URL: http://www.masfoundations.org/mas.pdf.

[10] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. 2nd. Cambridge, MA: MIT Press, 2017. URL: http://incompleteideas.net/book/the-book-2nd.html.

[12] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2010. URL: https://sites.ualberta.ca/~szepesva/RLBook.html.

[13] Karl Tuyls and Gerhard Weiss. "Multiagent Learning: Basics, Challenges, and Prospects". In: *AI Magazine* 33.3 (2012), pp. 41–52.

[14] Y. Zhuang et al. "Improving Monte-Carlo tree search for dots-and-boxes with a novel board representation and artificial neural networks". In: *IEEE Conf. on Comput. Intel. and Games (CIG)*. 2015.