CS 7642: Reinforcement Learning and Decision Making **Project #3**

Football

1 Problem

1.1 Description

For the final project of this course, you will develop a multi-agent reinforcement learning algorithm with function approximation, and train it in a modified version of the "Football" multi-agent environment originally developed by Google Research (Kurach et al. 2019). You will be provided with three baseline teams (details below), and your objective is to implement an algorithm/solution of your choosing that improves upon some aspect(s) of the baseline teams. This project serves as a capstone to the course and as such we expect much of it to be self-directed. Our expectation is that you have learned what is significant to include in this type of report from the previous projects and the material we have covered so far. It is thus up to you to define

- the direction of your project including which aspect(s) you aim to improve upon,
- how you specify and measure such improvements,
- how to train your agents, as well as
- how to structure your report and what graphs to include in it.

Your focus should be on demonstrating your understanding of the algorithm(s)/solution(s), clarifying the rationale behind your experiments and analyzing their results. Your focus should not be on building algorithms from scratch, and therefore you are free to use/extend any RL algorithm, library or package. Nonetheless, all submitted code should be your own (meaning do not copy and paste from a Medium article, GitHub repository or something similar).

1.2 Football Environment

In this project you will be training a team of agents to play the game of Football (Soccer in the USA). The original environment created by Google Research is a full 11v11 game. To reduce the computational cost, you will be working with a smaller version for this project. However, the core idea for the game is still the same: to train a team to maximize their chances of scoring and minimize their chances of conceding goals against an opposing team, by learning to cooperate with each other while competing against the skilled opponent team.

The modified environment you will be working with is a 3v3 game in which each team has two field players and a goalkeeper. Your goal will be to train the two field players on your team. This game is stochastic by nature, meaning the same player in the same state taking the same action (shooting the ball for example) may not necessarily achieve the same result. Additionally, the game terminates on one of three conditions: (1) a goal is scored, (2) the ball goes out of bounds, or (3) the game reaches the maximum time-step limit of 500. The Offside rule is disabled; *i.e.*, players can score from offside positions. The environment alternates the team starting with ball possession (and thus team on defense) between games.

1.3 Baseline Teams

We will provide you with three separate AI baseline teams, all trained using the Proximal Policy Optimization (PPO) algorithm. Each of the baseline teams is unique in the way they were trained: it could be the model architecture, the hyper-parameters used, whether the team uses a single policy (*i.e.*, sharing neural network weights), whether the model architecture contains an LSTM layer or not, whether the LSTM layers used the reward or the previous action as an input, and so on. How they were trained is not as important as the fact that these teams will have different behaviors. You will compare your team's win rate and behavioral metrics with each of the baselines. You will collect metrics of your choosing in episode rollouts generated by each of the baselines and your team. You are required to compare each baseline's performance against the hard-coded

game AI with the performance of your team against the hard-coded game AI. Your team doesn't have to play against each of the baselines as a direct opponent, though you're welcome to do so if you choose. Also, your team doesn't have to achieve a higher win rate than all three baselines or even higher values for the metrics you choose to get a good grade—though it helps, of course. Here are some definitions that may help you understand some distinctions:

- **Player**: The "unit" or "platform" that plays in the simulation. This being a 3v3 environment, it has 3 players per side.
- **Game AI**: Hard-coded logic that controls players. Typically, the game AI is built into the simulation, so from the perspective of RL, this logic is part of the environment and not really an AI—now, tell that to a game developer! In our 3v3 environment, your team has 1 player (the goalkeeper) that is controlled by game AI, which again, is part of the simulation, as well as the 3 players on the opponent team.
- **Control**: Players that are not controlled by game AI, and still need to be controlled from outside the simulation. This is Google Research Football specific definition (agent controls), though it also applies to other simulations.
- **Agent**: This is the true AI from the perspective of RL practitioners. An agent is a learning unit with decision-making abilities. An agent hooks to one or more controls. It's important to understand that while you may think agents and players have a one-to-one mapping, in reality one may have a single agent conducting centralized control of all players (even the two sides, check out self-play).
- **Policy**: A policy is a function that outputs actions. "Agent" and "policy" are often used interchangeably, however, know that some agents, for instance a Q-learning agent, only has an implicit policy that is extracted from an action-value function. On the other hand, agents trained with policy-based methods, such as a PPO, have an explicit policy model that map states to actions.
- **Model**: A model, in the context of deep reinforcement learning, is a neural network architecture. A policy can be represented with a wide variety of models: from number of layers, to number of nodes in layers, to types of layers, *etc.* For instance, a PPO agent's policy can be represented with a neural network model that includes an LSTM layer for memory, or maybe the input of the model includes information that other models don't, such as the reward, or the previous action. These models would be different ways of representing a policy.
- **Checkpoint**: It is common practice to save the policy at several times during training. Each time you save a policy, you create a checkpoint. A checkpoint can be saved, restored, stored, moved between computers, *etc.* Typically, a checkpoint contains all information needed to instantiate the architecture of a neural network model and restore its weights.
- **Baseline Team**: When we refer to a baseline team, we mean the team that gets formed after restoring one of the checkpoints that we provide to you. Restoring a baseline team checkpoint will load one or more policies that will map to the available controls (2 controls). The game AI will still control 1 player on the baseline team, and the other 3 players on the opponent team. All baseline teams where trained using an algorithm/agent called PPO, but they should produce different behaviors due to differences in training and architecture mentioned above.

1.4 State Representation and Action Space

The observation space for this environment is high-dimensional and continuous, requiring the use of methods similar to that seen in Project 2 but with a multi-agent twist. Google Research implemented this game with three different state representations, as described in Kurach et al. 2019. The multi-agent environment you will be working with is based on the 115-dimensional vector representation of the state. This observation vector summarizes many aspects of the original 11v11 environment such as the player's coordinates, ball possession and direction, active player, game mode, *etc.* Many of these variables are going to be meaningless in the modified environment you will be working with due to the missing positions from the missing players. We have provided you with a wrapper that eliminates these dimensions, reducing the state to a 43-dimensional vector. You **should not** augment or modify this vector in any way. These dimensions, in order, are:

- 6 - (x, y) coordinates of left team players;
- 6 - (x, y) direction of left team players;

- 6 - (x, y) coordinates of right team players;
- 6 - (x, y) direction of right team players;
- 3 - (x, y, z) ball position;
- 3 - (x, y, z) ball direction;
- 3 - one-hot encoding of ball possession (none, left, right);
- 3 - one-hot encoding of active player; and
- 7 - one-hot encoding of game mode (normal, kick-off, goal kick, free kick, corner, throw in, penalty). Note that some of the game modes don't apply for this project (*e.g.*, throw in, *etc.*), yet are still part of the observation vector.

The action space for Football is discrete, and consists of 19 actions including those pertaining to a player's movement (in 8 directions), different ways to kick the ball, and whether to sprint, slide tackle, or dribble. Some of these actions stop executing a previously selected "sticky" action. For example the "Stop-Sprint" action would disable the previously selected "Sprint" action. For more information about the action space, check out this document.

1.5 Reward Function

The base reward for this environment depends primarily on scoring or conceding a goal. However, the environment also includes reward shaping to help with the fact that the base reward is so sparse. Should a team score a goal, it receives a $+1$ reward while the opposing team receives -1. The shaped reward is based on when a player in possession of the ball moves across the opponent's side of the field, allowing them to earn up to 10 additional $+0.1$ rewards for passing distance checkpoints between the center of the field and the opponent's goal. This amounts to a possible additional reward of $+1$ for each player on a team. This reward, once achieved, does not change in a game: if a player moves from center field to the opponent's goal, they cannot circle back around and repeat the path to obtain the reward again nor do they lose that reward if they move backwards. Should a player score a goal before achieving this additional reward, the player will receive it upon scoring. For example, a player receives the $+0.1$ reward if it passes the first checkpoint, and if it scores before passing the other 9 checkpoints, then it will receive those rewards along with the reward for scoring (for a total of $+1.9$). While the reward is sparse, it is sufficient to solve this environment without any additional measures.

1.6 Strategy Recommendations

You are free to pursue any strategies on your quest for improving your team. But, make sure to address multi-agent RL specific issues. For instance, you may pursue a novel reward shaping technique, however, make sure that it is relevant to multi-agent RL problems. Below are some examples of possible strategies to consider:

- centralizing training and decentralizing execution (Lowe et al. 2017; J. N. Foerster et al. 2017);
- adding communication protocols (J. Foerster et al. 2016);
- improving multi-agent credit assignment (J. N. Foerster et al. 2017; Zhou et al. 2020);
- improving multi-agent exploration (Iqbal and Sha 2019; Wang et al. 2019)
- learning hierarchically (Makar, Mahadevan, and Ghavamzadeh 2001);
- finding better inductive biases (*i.e.*, choosing the function space for policy/value function approximation) to handle the exponential complexity of multi-agent learning, *e.g.*, graph neural networks (Battaglia et al. 2018; Naderializadeh et al. 2020).
- employing curriculum learning (some single-agent ideas in this dissertation may be interesting and easy to extend to the multi-agent case *e.g.*, Narvekar 2017).
- using reward shaping techniques for improving multi-agent considerations such as collaboration and credit assignment;

1.7 Procedure

This problem is more sophisticated than anything you have seen so far in this course. **Make sure you reserve enough time to consider what an appropriate approach might involve and, of course, enough time to build and train it.**

- Clearly define the direction of your project and which aspect(s) you aim to improve upon. For example, do you want to improve collaboration among your agents?
  - This includes why you think your algorithm/procedure will accomplish this and whether or not your results demonstrate success.
- Implement a solution that produces such improvements.
  - Use any algorithms/strategy as inspiration for your solution.
  - **The focus of this project is to try new algorithms/solutions, not to simply improve hyper-parameters of the algorithms already implemented for you. Nor is it to search for random seeds that happen to work the best.**
  - Justify the choice of that solution and explain why you expect it to produce these improvements.
  - **You can write a solid paper even if your solution does not improve upon the base agents.**
  - Upload/maintain your code in your private repo at `https://github.gatech.edu/gt-omscs-rldm`.
- Describe your experiments and create graphs that demonstrate the success/failure of your solution.
  - You must provide one graph demonstrating the win rate improvement of your team over the three baselines teams.
  - Additionally, you must provide two graphs using metrics you decided on that are significant for your hypothesis/goal.
  - Analyze your results and explain the reasons for the success/failure of your solution.
  - Because these graphs are largely decided by you, they should have **clear** axis, labels, and captions. You will lose points for graphs that do not have any description or label of the information being displayed.
  - **To give you a starting point for thinking about behavioral metrics and ways of graphing the actions your agents take**, DeepMind (Liu et al. 2019) has a great paper on emergent behavior in multi-agent soccer environments that shows many different ways you can evaluate behavior. We hope this can serve as inspiration for defining success or failure in your report.
- We've created a private Georgia Tech GitHub repository for your code. Push your code to the personal repository found here: `https://github.gatech.edu/gt-omscs-rldm`.
- The quality of the code is not graded. You do not have to spend countless hours adding comments, *etc.* But, it will be examined by the TAs.
- Make sure to include a `README.md` file for your repository that we can use to run your code.
  - Include thorough and detailed instructions on how to run your source code in the `README.md`.
  - If you work in a notebook, like Jupyter, include an export of your code in a `.py` file along with your notebook.
  - The `README.md` file should be placed in the project_3 folder in your repository.
- You will be penalized by **25 points** if you:
  - Do not have any code or do not submit your full code to the GitHub repository; or
  - Do not include the git hash for your last commit in your paper.
- Write a paper describing your agents and the experiments you ran.
  - Include the hash for your last commit to the GitHub repository in the header on the first page of your paper.

- Make sure your graphs are legible and you cite sources properly. While it is not required, we recommend you use a conference paper format. For example: https://www.ieee.org/conferences/publishing/templates.html.
- 5 pages maximum—really, you will lose points for longer papers.
- Explain your algorithm(s).
- Explain your training implementation and experiments.
- An ablation study would be a interesting way to find out the different components of the algorithm that contribute to your metric. (See J. N. Foerster et al. 2017.)
- Graphs highlighting your implementations successes and/or failures.
- How does the scores achieved by your algorithm compare to the baselines?
- Explanation of algorithms used: what worked best? what didn't work? what could have worked better?
- Justify your choices.
  - Unlike Project 1, there are multiple ways of solving this problem and you have a lot of discretion over the general approach you take as well as specific design decisions. Explain to the reader why, from amongst the multiple alternatives, you chose the ones you did.
  - **Your focus should be on justifying the algorithm/techniques you implemented.**
- Explanation of pitfalls and problems you encountered.
- What would you try if you had more time?
- Save this paper in PDF format.
- Submit to Canvas!

1.8 Installation Notes

We have setup three different approaches to assist you in getting started with this project, using the football environment, visualizing the game, and training an agent using RLlib—see below. Each approach comes with its own benefits and issues, so we'd recommend examining each of them and carefully considering which would be best given the hardware you possess, your technical skills setting up services like Docker, and your ability to adapt/understand others code. As always, the teaching staff cannot diagnose installation issues for every student so if you encounter issues we invite you to post to the forums and collaborate with your fellow students to resolve any hiccups encountered along the way.

- **Docker image (recommended):** This installation option has a more complex setup than the previous, has a greater system memory requirement, and will put a greater strain on your hardware. However, it does have the benefit of giving you additional tools and starter code implemented by the TAs to get you started quickly. Among other things, it includes code to produce several interesting graphs out of the gate using TensorBoard. Instructions for this installation can be found here.
- **Native Installation:** This installation runs on your local machine, but requires less technical know-how to setup than the Docker installation. Note that the Docker installation, while more challenging up-front, may be beneficial in the long run. The specific instructions for the native install can be found here. As a broad overview, first, install PyTorch. Next, install RLlib version 1.6, in which you'll find a great number of algorithms and tools to help you train your agents. Finally, follow the instructions here to install the Google Research Football simulation engine and environments.
- **Google Colab:** The primary purpose of this "installation" method is to serve as a final resort if you are **certain** your hardware cannot handle the computations involved in this project. We encourage you to at least test out the Native implementation first, before automatically resorting to this method. This installation method could also provide a path for those wanting to pursue a "cloud" solution. Unfortunately, Google Colab has limits in the resources that you can use, and therefore, we cannot recommend this path to all students. However, some students may still be able to leverage these instructions and use them in other ways. A Google Colab notebook can be found here with specific instructions on how to set everything up using your Google Drive. In short, after copying this notebook to your own Colab files, you will need to clone the Project 3 code from the Docker repository, mount your Google Drive, and select a GPU run-time. After that, you will be able to start playing the Notebook. But, again, due to the nature

of Google Colab, and the fact that run-times for this project will be longer than their 12-hour limit, we recommend you only try this as a last resort, and look into the premium options before getting started. Pro and Pro+ are $10 and $50 per month respectively, both of these options can stay connected for up to 24 hours and have more lenient idle times. While there's benefits to both of the premium subscriptions, they also have some limits (*e.g.*, vCPU) that you should look into to decide whether either is right for you.

1.9 Resources

1.9.1 Lectures

- Lesson 11A: Game Theory
- Lesson 11B: Game Theory Reloaded
- Lesson 11C: Game Theory Revolutions

1.9.2 Readings

- J. N. Foerster et al. 2017
- Lowe et al. 2017
- Rashid et al. 2018
- Kurach et al. 2019

1.9.3 Talks

- Factored Value Functions for Cooperative Multi-Agent Reinforcement Learning
- Counterfactual Multi-Agent Policy Gradients
- Learning to Communicate with Deep Multi-Agent Reinforcement Learning
- Automatic Curricula in Deep Multi-Agent Reinforcement Learning

1.10 Documentation

- Google Research Football paper
- PyTorch Docs
- RLlib Documentation
- RLlib Concepts and Building a Custom Algorithm Tutorial
- Multi-Agent Methods Offered by RLlib
- RLlib Callbacks and Custom Metrics
- Football Environment

1.11 Submission Details

The due date is indicated on the Canvas page for this assignment. Make sure you have set your timezone in Canvas to ensure the deadline is accurate.
Due Date: **Indicated as "Due" on Canvas**
Late Due Date [**20 point penalty per day**]: **Indicated as "Until" on Canvas**
The submission consists of:

- Your written report in PDF format (Make sure to include the git hash of your last commit)
- Your source code in your personal repository on Georgia Tech's private GitHub

To complete the assignment, submit your written report to Project 3 under your Assignments on Canvas: https://gatech.instructure.com

You may submit the assignment as many times as you wish up to the due date, but, we will only consider your last submission for grading purposes. Late submissions will receive a cumulative 20 point penalty per day. That is, any projects submitted after midnight AOE on the due date get a 20 point penalty. Any projects submitted after midnight AOE the following day get a 40 point penalty and so on. No project will receive a score less than a zero no matter what the penalty. Any projects more than 4 days late and any missing submissions will receive a 0.

Please be aware, if Canvas marks your assignment as late, you will be penalized. This means one second late is treated the same as three hours late, and will receive the same penalty as described in the breakdown above. **Additionally, if you resubmit your project and your last submission is late, you will incur the penalty corresponding to the time of your last submission.**

Finally, if you have received an exception from the Dean of Students for a personal or medical emergency we will consider accepting your project up to 7 days after the initial due date with no penalty. Students requiring more time should consider taking an incomplete for this semester as we will not be able to grade their project.

1.12 Grading and Regrading

When your assignments, projects, and exams are graded, you will receive feedback explaining your errors (and your successes!) in some level of detail. This feedback is for your benefit, both on this assignment and for future assignments. It is considered a part of your learning goals to internalize this feedback. This is one of many learning goals for this course, such as: understanding game theory, random variables, and noise.

If you are convinced that your grade is in error in light of the feedback, you may request a regrade within a week of the grade and feedback being returned to you. A regrade request is only valid if it includes an explanation of where the grader made an error. **Create a private Ed Discussion post titled "[Request] Regrade Project 3"**. In the Details add sufficient explanation as to why you think the grader made a mistake. Be concrete and specific. We will not consider requests that do not follow these directions.

It is important to note that because we consider your ability to internalize feedback a learning goal, we also assess it. This ability is considered 10% of each assignment. We default to assigning you full credit. If you request a regrade and do not receive at least 5 points as a result of the request, you will lose those 10 points.

1.13 Words of Encouragement

We understand this is a daunting project with several directions in specific design choices. As Master's Students in Computer Science, these projects are key within the program which allow you to challenge and test your skills in a practical and low-stakes manner. These projects are ideal for the knowledge you have garnered throughout the course and apply to a difficult problem many times faced in the current reinforcement learning industry. From historical data, after completing the course, a project like this is important to highlight during job recruiting, show off to current employers, or add to a (new) section on your resume. Many students report back the notable positive interactions when discussing the project, leading to job offers or promotions in their current workplace. However, please remember not to publicly post your report or code, the project would be a good **talking** point and you would be fine sharing it with a potential employer if you so desired but you should not violate the GT Honor Code. We encourage you to start early and dive head-first into the project to try as many options as possible as you will only grow and learn through the successes and failures throughout the project.

The teaching staff is dedicated to help as much as possible. We are excited to see how you will approach the problem and have many resources available to help. Over the next several Office Hours, we will be discussing various approaches in detail, as well as dive deeper into any approach on Ed Discussions. We are here to help you and want to see you succeed! With all that said:

Good luck and happy coding!

Figure 1 from Kurach et al. 2019 demonstrating the soccer environment in action.

References

[Bat+18] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[Foe+16] Jakob Foerster et al. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 2137–2145.

[Foe+17] Jakob N. Foerster et al. "Counterfactual Multi-Agent Policy Gradients". In: *CoRR* abs/1705.08926 (2017). arXiv: 1705.08926. URL: http://arxiv.org/abs/1705.08926.

[IS19] Shariq Iqbal and Fei Sha. "Coordinated Exploration via Intrinsic Rewards for Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1905.12127* (2019).

[Kur+19] Karol Kurach et al. "Google Research Football: A Novel Reinforcement Learning Environment". In: *CoRR* abs/1907.11180 (2019). arXiv: 1907.11180. URL: http://arxiv.org/abs/1907.11180.

[Liu+19] Siqi Liu et al. "Emergent coordination through competition". In: *arXiv preprint arXiv:1902.07151* (2019).

[Low+17] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *CoRR* abs/1706.02275 (2017). arXiv: 1706.02275. URL: http://arxiv.org/abs/1706.02275.

[MMG01] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. "Hierarchical multi-agent reinforcement learning". In: *Proceedings of the fifth international conference on Autonomous agents*. 2001, pp. 246–253.

[Nad+20] Navid Naderializadeh et al. "Graph Convolutional Value Decomposition in Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:2010.04740* (2020).

[Nar17] Sanmit Narvekar. "Curriculum Learning in Reinforcement Learning." In: *IJCAI*. 2017, pp. 5195–5196.

[Ras+18] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *CoRR* abs/1803.11485 (2018). arXiv: 1803.11485. URL: http://arxiv.org/abs/1803.11485.

[Wan+19] Tonghan Wang et al. "Influence-Based Multi-Agent Exploration". In: *International Conference on Learning Representations*. 2019.

[Zho+20] Meng Zhou et al. "Learning implicit credit assignment for multi-agent actor-critic". In: *arXiv e-prints* (2020), arXiv–2007.