Assignment # 2 - Linear Function Approximation and Feature Design for Tackling a New RL Problem *from Scratch*

Author: Prabhat Nagarajan, Marlos C. Machado CMPUT 628, Deep RL: Winter 2025

Instructions and General Notes

- **Due Date**: February 7, 2025.
- **Late Policy**: See syllabus.
- **Marking**: There will be 150 total marks.

Collaboration and Cheating

Do not cheat. You are graduate students. You can discuss the assignment with other students, but do not share code with one another. Do not use language models to produce your solutions.

General Advice

We highly recommend **starting the assignments early**. In machine learning and reinforcement learning, training agents or learning systems requires a different skillset from traditional programming and software engineering. Oftentimes the process of debugging is far more time-consuming, as code may be only "somewhat" wrong and so discovering why your agent is underperforming is not a straightforward task. Moreover, debugging often involves having agents run for periods of time, and training can be both expensive in terms of both time and compute. Building agents is also not merely validating program correctness, but involves a lot of trial and error on the programmer's part (not just the agents). This assignment can end up being particularly difficult due to the many components you will need to design and parameter tuning.

Software and Dependencies

This assignment will use:

- Python 3.11
- `numpy`
- `matplotlib`
- `gymnasium[classic-control]`
- `jumping-task@git+https://github.com/prabhatnagarajan/jumping-task@gymnasium`

The files contain all the required imports. You should not need to import any additional libraries or packages. You are welcome to import additional libraries for debugging purposes, but your final submission should not include any new imports.

Submission

Please submit a zipped folder titled `a2_<ccid>.zip`, where you replace <ccid> with (can you guess?) your ccid. E.g., a2_machado.zip. The unzipped folder should be `a2_<ccid>`. **Failure to do so will cost you 5 marks.** This zip file should contain:

- pdf titled a2_<ccid>.pdf.
- All the python files from a1.zip containing your solutions in `a2_<ccid>.zip`.
- Do not include the test code in your submission.

Assignment

In this assignment, we will learn the basics of reinforcement learning learning with linear function approximation. This assignment will also require you to fully understand the learning process, not only the impact features have on it, but also things like exploration and credit assignment. Quite often, we are used to developing a reinforcement learning agent in an environment where someone already succeeded and paved the way regarding algorithmic solutions. It is easier to walk a paved way. This assignment will require you to pave the way. This sets the stage for our transition to deep reinforcement learning with nonlinear function approximation through multi-layered neural networks and, hopefully, will help you develop skills that allow you to use RL algorithms (deep or not) to tackle entirely new problems.

In particular, in this assignment, you will implement:

- **Linear Semi-gradient SARSA**: You will implement Linear Semi-gradient SARSA (see sections 10.1 and 10.2 by Sutton & Barto (2018)).
- **Feature design**: You will design your own features for a jumping task and will train it using your Semi-gradient SARSA agent.
- **Algorithmic Improvements**: You may (and very likely will have to) improve your Semi-gradient Sarsa implementation.

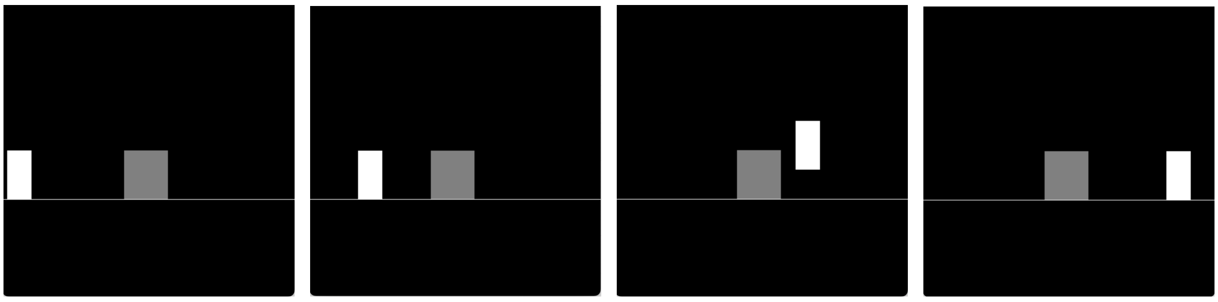
This assignment comprises two parts:

Mountain Car

In the first part of this assignment, you will implement Semi-gradient SARSA on the Mountain Car task (see Example 10.1 in Sutton & Barto's book) using tile coding and linear function approximation. The idea here is for you to do what is the best practice in terms of development. When tackling a new problem, it is better to validate your solution as much as possible in a well-established problem so you know it works at least there.

Jumping Task

The second part of this assignment involves feature design. You will use the Semi-gradient SARSA implementation as a starting point. In the second part of this assignment, you will use linear temporal-difference learning algorithms and will design your own features (i.e., not using tile coding necessarily) for the jumping task, and you will implement a learning algorithm that is capable of learning, online, with those features.



(a) Going Right (b) Going Right (c) Post-jump (d) Approaching the end

Figure 1: **Jumping Task.** At each time step, the agent receives the game screen as observation and a reward signal. The reward signal is $+1$ at each time step, 0 if the agent collides with the obstacle, and 100 if the agent reaches the rightmost part of the screen. The agent has access to two actions: `right` and `up`. We recommend you familiarize yourself with the environment through `jumping_task_play.py`.

Tests

- This assignment will have few tests, all in Mountain Car, and as such no source code for the tests will be provided. We will, however, provide the binaries for all the tests (again, there are tests only in Mountain Car). To run all the tests, run the provided binary file, pointing it to the directory with all your code.
  E.g. `./a2_tests /Users/machado/assignments/a2/a2_machado/`

Agent Environment Interface: `agent_environment.py` [5 marks]

These marks are easy! Since you will be reusing this functionality, you will be re-marked on the last assignment's code (**except at 2.5 marks per function instead of 10**). If you got full marks on the previous assignment, you may simply copy your solutions and receive free points. If you did not get full points last assignment, then this is your opportunity to fix your code!

ϵ-greedy exploration: `epsilon_greedy_explorers.py` [5 marks]

Just like above, these marks are easy! Since you will be reusing this functionality, you will be re-marked on the last assignment's code (**except at 1.25 marks per function instead of 10**). If you got full marks on the previous assignment, you may simply copy your solutions and receive free points. If you did not get full points last assignment, then this is your opportunity to fix your code!

Linear Semi-Gradient SARSA: `semi_gradient_sarsa.py` **[25 marks]**

Your implementations should:

- Perform the correct updates under truncation, termination, and nonterminal transitions.

In both files you should implement the `act`, `update_q`, and the `process_transition` functions. `act` should, amongst other things, utilize the explorer to produce the agent's action in a state. The `update_q` method should actually update the `self.q` estimate. The `process_transition` method should process the consequences of the previous decision, including calling the `update_q` method with the appropriate parameters. You are free to make modifications/introduce variables within the scope specified for your code. The tests include:

- Correctly computing Q-values given weights and a feature vector (`compute_q_values`) **[5 marks]**.
- A standard linear semi-gradient Sarsa update with bootstrapping (`update_q`) **[10 marks]**.
- A standard linear semi-gradient Sarsa update with a transition to a terminal state (`update_q`) **[10 marks]**.

Mountain Car: `tile_coding_mountain_car.py` **[20 marks]**

Once the previous files have been completed correctly, you should be able to run `tile_coding_mountain_car.py` successfully, which should produce `sarsa_mountain_car.png`.

You should not modify this file, other than replacing the `CCID` **variable with your ccid**. Submit the file `sarsa_mountain_car.png` (where you use your ccid).

- With your `sarsa_mountain_car.png`, assuming adequate performance (indicating a correct implementation), you should receive marks accordingly. **[10 Marks]**
- Answer the following question about exploration: ***What exploration strategy are we using here, and why does it work?*** **[10 marks]**

Jumping Task `feature_extraction.py`**,** `train_jumping_task.py` **[125 marks]**

In this part of the assignment, you will play with three variants of the Jumping Task. You may personally play the different tasks using

```
python jumping_task_play.py --config [1,2,3] --num-episodes <num-episodes>
```

where the `config` flag specifies the environment and the `num-episodes` flag specifies how many episodes you would like to play.

Each task has some different configurations. The first task requires the agent to jump over an obstacle. The floor and obstacle may change locations between episodes. The second task is similar, but with a different agent shape, obstacle shape, and jump height. The third task has two obstacles.

Your task is to write (in code) feature extractors for these tasks (without using machine learning in the loop, i.e., don't learn features). You may write feature extractors for each task individually or write a single set of features that apply to all three tasks.

Your feature extractor should be written in `feature_extraction.py` and the code should be run from `train_jumping_task.py`. Unlike the previous assignments and the tile coding example for Mountain Car, you may modify both of these files. In particular, you can modify the main function of `train_jumping_task.py`, and *add* any code (excluding imports) to the other parts of the file. You are free to *add* any code (excluding imports not in the assignment) to `feature_extraction.py`. You may not include additional imports (though feel free to do so for debugging purposes) other than the ones provided or in other files written for the assignment. All of your code for this portion of the assignment must be written in these two files.

You have the freedom to write new exploration strategies and agents (in these files) to solve this task. Your constraints are that the agent must be some variant of linear semi-gradient Sarsa. You are free to come up with clever exploration strategies, but are not allowed to use excessive domain knowledge (e.g., hardcoding policies to perform well). You are free to set the discount factor γ as you see fit.

You may have different hyperparameter settings for each task. You should specify how to reproduce that from the command line. For any hyperparameters that are needed, add parseargs arguments for then feel free to come up with an `explorer` (not too much domain knowledge).

Your explanation/report for the Jumping task should be **no more than two pages**.

- *Performance* **[30 marks]**
  - Add plots for each environment configuration to `a2.pdf`.
  - Eventually you should reach an average success rate (across a minimum of 5 seeds) of 90% in the last ~ 500 episodes of the task. You will receive 10 marks for achieving at least 90% mean success for each config, 5 points for at least 40% performance, and 0 points otherwise. In particular, we will look at the last 100 episodes in your learning curves. Your training runs should have at most 100K episodes (hopefully you need nowhere close to this many timesteps; our solution succeeds in less than 5k timesteps).
  - Provide the command line commands to run to reproduce (qualitatively) each learning curve. If you have different hyperparameters for each configuration, then be sure to add these hyperparameters to the argument parser.
- *Features* **[20 marks]**
  - Describe your features and how/why you produced them.
  - Did you have features that worked on only one task at first?
  - Did you find one set of features that works on all tasks?
  - Did you use three different sets of features?
  - What considerations did you have to make when designing and experimenting with features?

- Describe your full solution (though features have already been explained) in terms of algorithms, exploration mechanisms, hyperparameter choices, etc. Write how your solution (perhaps including features) touches on the elements of exploration, generalization, and credit assignment. What are some challenges you faced and how did you address them? **[35 marks]**
- What challenges do you think might arise when we have different colors and different sprites. Do you think your feature extractor would work or not? **[10 marks]**

Acknowledgements

- We used the tile coding software from `http://incompleteideas.net/tiles/tiles3.py-remove`
- We forked our jumping task code from `https://github.com/google-research/jumping-task`. It was originally proposed in the following reference:

  Tachet, R., Bachman, P., & van Seijen, H. (2018). Learning Invariances for Policy Generalization. In Int. Conference on Learning Representations (ICLR), Workshop Track.