CS 4180/5180: Reinforcement Learning and Sequential Decision Making (Spring 2024) – Final Project

1 General Information

The project offers an opportunity to apply concepts/techniques learned in class on a substantial problem that interests students. Whereas most of the course has been focused on breadth of topics, the project provides a counterpoint by requiring students to study a problem in depth.

Topics

Your final project can be on anything that has some relation to RL and/or sequential decision making. Your project will likely involve MDPs, although there can be exceptions for related frameworks (e.g., bandits, POMDPs, etc.). It does not necessarily have to use any of the methods we cover, although it is likely that you will be using at least some of the material.

Types

This semester, we offer two types of projects:

- **Practical**: Find and formulate a problem, develop/implement/apply algorithms to solve it. Recommended if you want to escape the toy domains we have studied so far and actually build something substantial and potentially applicable in the real world. You would likely learn how to use new RL-related software tools and hone algorithmic / data-scientific skills. If successful, the project would be a great addition to your portfolio. Also a great chance to do the RL-related hobby project that you have always dreamed of but never got started on.
- **Theoretical**: Do an independent study of an advanced topic not covered in the course. Recommended if you have enjoyed the topics so far and just want to see more of it, or go significantly deeper into something we briefly touched on. You will learn about new problems / models / algorithms. This is recommended if you plan to continue studying RL in the future, and/or if you are interested in getting involved in RL-related research. Could also be used to deeply study / reproduce a research paper.

Teams

Teams should have 1–3 members. Our recommendation:

- Practical project: We strongly recommend forming teams of 2, which in past experience has led to much more interesting projects. Overall, we expect each team member to spend about 60–80 hours on the project (about 10 hours per week). This is why teams of two are recommended – you can accomplish much more with 140 total hours compared to 70.
- Theoretical project: Teams of 1 or 2 are recommended. If in a team of 2 or more, the chosen topic should be broader, and there should be a rationale as to why more people are needed.

Deviating from the above recommendations will require a written rationale in the proposal, and potentially approval from / further discussion with Prof. Wong.

Expectations

We are mostly interested in the **process** of your project, and not the final results.

- Practical project: We do not expect everyone to achieve state-of-the-art results on the problem they choose – although that would be a pleasant surprise if it happened. Your team is expected to find a problem of interest, formulate it well and precisely, develop/apply an algorithm to solve the problem, analyze your results, and communicate that to the class (and us).
- Theoretical project: You should learn the proposed topic and understand it well enough to the point that you can potentially teach an hour-long lecture on it. We will not actually ask you to deliver this lecture unless you really want to. Instead, you will learn and digest the chosen topic, and produce a tutorial on it, such as in the form of a report, slides, illustrative examples, or other pedagogical materials. There is more flexibility on the deliverables.

Timeline

All deadlines, except for the project presentation, are at **11:59 PM ET**.
Submissions should be uploaded via Canvas.

- The project proposal is due 3/8 (Fri).
- There is one self-proposed milestones, which will be due 4/12 (Fri).
- Project **poster** presentations will happen in the final **three** classes on 4/17 (Wed), 4/22 (Mon), and 4/24 (Wed). Presentation slots will be assigned on a semi-random basis, after taking conflicts into account. Most presentations will occur on 4/22 or 4/24; however, since 4/17 is the final day of class for the CS 4180 section, any teams that need to present on 4/17 will be offered the opportunity to do so.
- The draft project report (practical) / deliverables (theoretical) is due on 4/19 (Fri).
- **The final project report / deliverables is due on 4/25 (Thu), 11:59 PM ET.**
  This is rather strict, because final grades are due soon after that, and we need time to ensure every project receives a proper assessment.

Typical project process (practical track)

Overall, we expect each team member to spend about 60–80 hours on the project (about 10 hours per week). This is why teams of two are recommended – you can accomplish much more with 140 total hours compared to 70.

1. ($\sim$ 5 hours) Decide on a project topic, do some background reading, and make an initial problem formulation. Make a rough plan for the project itself (this is the proposal).
2. ($\sim$ 10 hours) Learn the additional knowledge necessary to undertake the project. The amount of time this takes may vary depending on your chosen topic. It is very likely that you will need to learn at least one or two new concepts/algorithms for your project.
3. ($\sim$ 10 hours) Figure out the simplest way to solve your problem (i.e., a basic algorithm), and implement it / use an existing implementation. Apply the simple algorithm to your problem.
4. ($\sim$ 5 hours) There are usually two results of step 3: Either you have completely solved the problem you chose, or it doesn't work too well (at least on some cases). If the former, analyze why it did so well, and come up with some interesting extensions. If the latter, analyze what went wrong, and figure out how to address that.
5. (Remaining time: $\sim$ 30 hours?) Iterate steps 3-4, each time using the analysis in step 4 to identify some potential area to improve on, and try again (step 3).
6. ($\sim$ 10 hours) Summarize the overall process: the problem, algorithm, results, and insightful analyses. Present your story to the class / in your final report.

Typical project process (theoretical track)

This is more fluid and harder to predict. It depends on the chosen topic and its scope.

1. ($\sim$ 20–30 hours) Read the textbook and/or other tutorial material, possibly including watching recorded lectures, to gain a broad understanding of the topic and its context.
2. ($\sim$ 20–30 hours) Work on problems / code simple examples and experiment with variations, to solidify and deepen the initial understanding, and to sort out confusions and misconceptions.
3. ($\sim$ 20 hours) Distill your understanding into project deliverables (such as in the form of a report, slides, illustrative examples, or other pedagogical materials).

Throughout the project, there may be more frequent check-ins with Prof. Wong to ensure that your independent study is on the right track, and to jointly define what the final results of the project may look like.

2 Potential Topics

This section provides some general topic categories and specific suggestions, but you are welcome to choose anything else that interests you.

Topic categories

- (Practical) Find a decision-making problem in real life that is worth automating. When we consider agents, the first thing that comes to mind may be game-playing agents and robots, which are certainly great applications. However, many other things can (and do) use automated decision making, e.g., elevator scheduling, transportation systems (traffic lights, lanes, etc.), electric grids, logistics and warehousing, computer processor and memory control, etc. RL has actually been applied to many of these; go explore.
- (Practical) Furama Foundation Gymnasium – This platform originally from OpenAI (Gym) contains many interesting environments for training reinforcement learning (RL) algorithms. It is quite popular for RL enthusiasts and is often used to benchmark RL algorithms. DeepMind Control Suite is a another such platform focusing on continuous-control tasks. There is an increasing number of similar platforms. Note that it is insufficient to simply apply an existing codebase (e.g., DQN) on an existing benchmark domain (e.g., Atari) – you should implement your own algorithms and/or extend existing methods.
- (Theoretical) Learn extensively about an problem / model / algorithm that is just beyond the scope of the course. Develop / apply and convey the understanding on illustrative examples. Produce pedagogical materials about the topic. See below for some suggested specific topics.
- (Theoretical) Find a paper that tackles an interesting problem, and try to re-implement it (initially without referring to existing code, if any is available). This can be quite challenging because you may find that some crucial details (e.g., particular settings of constants or hyperparameters) may not be present in the paper!
- (Theoretical) If you have ideas about how to substantially improve and liven pedagogical materials used in current class topics, and want to improve the learning experience of future students, you can implement them in a project. Substantial improvement means the introduction of illustrative examples, visualizations, interactive demonstrations, or other engaging material. This will require frequent discussion with Prof. Wong.

Potential topic areas not covered in the course (non-exhaustive)

- More on deep reinforcement learning (see recent papers at ICLR / ICML / NeurIPS / etc.)
- Robot learning (see recent papers at CoRL, RSS, ICRA, IROS, etc.)
- RL in X (e.g., healthcare, education, logistics, NLP, games, robotics)
- Imitation learning / learning from demonstration
- Temporal abstraction / state abstraction in MDPs and RL
- Exploration / intrinsic motivation / curiosity / curriculum learing / safety in RL
- Multi-agent RL / Bayesian RL / Partially observable RL / POMDPs

Projects that are *not* recommended

- Apply an existing algorithm from an existing codebase to an existing domain/benchmark and report the results. This is not enough – you need to analyze the results, see what was good and what could be improved, and iterate.
- Projects on unrelated topics – if you are unsure, ask us!
- Projects that are too broad. 60–80 hours (or 120–160) is not a lot of time. Start small (very small), and if you succeed early, extend and iterate from there. If there is an interesting problem that is likely to take too much time (this is true for many interesting problems), identify the first step in the problem and make that your project.

Comments regarding implementation (code)

- We expect *all* projects will have some implementation component, even if it is just on illustrative toy domains (e.g., if you are investigating a new algorithm as part of a theoretical project). You can use whatever programming language you like. This choice may be dependent on any existing code/libraries that you build upon.
- If there is an existing implementation, you can use it (with proper acknowledgment), but you do not have to. It depends on what you want to get out of the project. If you want to extend an existing method, then you will probably save time by building on existing implementations (especially if this is some non-RL related infrastructure, e.g., a game engine). If you want to have the experience of writing from scratch and deeply understanding the existing approach, you are welcome to do a re-implementation (although you should not refer to the existing "answer" in this case in your initial attempt). If you choose to re-implement, you may find that you cannot replicate existing performance, at which point you can compare against the existing implementation and see what went wrong – is it a bug on your end, or are there some 'magic numbers' that make the algorithm work? Analyzing these hidden / unexpected bits can make a very interesting project.
- Keep in mind that RL methods take a while to run (recall experience in assignments), and modern deep RL methods take a long time to run (e.g., on the order of multiple days is not uncommon, even with very high-end CPUs and GPUs). Make sure you have a plan to start small and iterate quickly. If you do plan on eventually scaling up, make sure you have access to high-end computing resources, e.g., your own server, the Discovery cluster, cloud computing, etc. Note that we will not be able to provide technical/financial support for any of these external resources, so make sure you are comfortable using them early on. There are many RL projects that do not require such intensive computing resources.

3 Project Proposal

Each team submits one proposal. The project proposal is a short document that organizes your team's intentions and communicates that to the course staff, such that we can provide appropriate guidance. It is not a contract – we expect that projects may change course after you start working on them. As such, the proposal will not be graded; however, if you do not submit a proposal, it may negatively affect your project grade. There is no specific page limit, but we expect that a 1-2 pages (12-pt single spacing) should suffice. Just make sure to include all the following items:

- Who is on the team? If you are working in a team of two, is there a clear division of labor? If team is smaller/larger than recommended, provide a rationale.
- Describe the problem you are trying to address, and provide a formulation of it. If it involves an algorithm (i.e., any implementation-based project), describe the input to the algorithm and the desired output.
- What is the ideal outcome of the project? What do you expect to show?
- What algorithms do you expect to use?
- What topics / libraries / platforms, if any, will you have to learn in order to undertake your project? Provide references where applicable.
- What domain(s) will you be working on? Is there a simulator available? If not, will you making a new one? If the latter, do you have the resources / data to do so?
- Define a halfway milestone for your project. This will be due on 4/12 – note that this is only about a week before your draft report is due. Milestones can include learning about certain topics / algorithms, acquiring and processing datasets, implementing an algorithm, analyzing results, etc. Again, we will not penalize you if you do not achieve your specific milestone. At the due date, we will request a short progress report (see next page) that addresses progress toward the milestones, and if they were not achieved, what turned out to be more challenging than expected. You may also find that your initial milestones were inappropriate, and you can choose different milestones, work toward those, and report on them. That is completely fine too – the point is to decompose your project into smaller pieces and ensure that you are making consistent progress over the next few weeks.
- Provide a week-by-week plan for your project.

Theoretical track

The proposal should be similar to the above, but certain items above may not be relevant (e.g., libraries/platforms, dataset). Additionally, provide an initial list of learning material you will study to understand the topic, and provide a list of proposed deliverables that you would like to develop (such as in the form of a report, slides, illustrative examples, or other pedagogical materials).

4 Project Milestones

As previously mentioned, the proposal and the proposed milestones are not treated as contracts, but rather as your best guess of how things will turn out. Therefore, you will not be penalized if you fail to achieve your proposed milestones. In fact, project milestones are rarely reached on time, so we do not expect that of you either.

Why set milestones when they are unlikely to be achieved? It would be nice to accomplish them, so if it does actually work out, good for you. More importantly, as the first module of the course showed, planning is the basis of rational decision making. The milestones you set (and the week-by-week plan requested in the proposal) are supposed to guide your actions, and allow you to determine the feasibility of your proposed project, in terms of the time and effort required to achieve your goals. (Does there actually exist a solution to reach your goal that respects time and resource constraints?) Of course, there is uncertainty in the outcomes, so technically you need to plan under uncertainty ...

... which leads to what we expect for the milestones. So far, we have covered the MDP approach to decision-making under uncertainty – if you were really on top of your project proposal game, you would produce a *policy* instead of a plan. There is another common approach, known as replanning, which we will learn by doing here. This approach is as the name suggests – if the outcome is different from what you expected, plan again from your current state. The point of the milestone deliverables is to reflect on what you have done so far, how it went, and to replan as necessary.

More concretely, we expect milestone reports (one report per team) to address the following 3 'R's (not Reading, wRiting, aRithmetic):

1. **Report**: What have you done so far? What is the current state of the project? To ensure best use of time, we recommend treating this question as a partial draft of the final report – ideally, if your project does not change, you can write 1–2 pages for this section in each milestone report, and then copy and paste them into the final report.

2. **Reflect**: Did you achieve your milestone(s)? If not, why not (e.g., some part was challenging because of [*insert challenge here*] and hence took longer, some other task became irrelevant)? Did you do something extra that you did not mention in your proposed milestone(s)?

3. **Replan**: Based on your progress so far and the knowledge you have acquired, what is the upcoming plan (week-by-week)? What is your immediate next step? If your project milestones / goals have changed from the initial proposal (e.g., you underestimated / overestimated the difficulty of the project), propose new milestones / goals.

Good project progress will roughly follow this timeline (some modifications for theoretical track):

- By the milestone (4/12): Have a precise problem formulation (inputs and outputs). Learned the basics of any additional topics necessary to accomplish project. Collected all the necessary external pieces (e.g., simulator / model, external libraries if needed, etc.). High-level description / pseudocode of first algorithm to try, if you haven't tried anything yet; or describe which algorithms you have already tried. Ideally, you would have already implemented and experimented with at least one algorithm / model. If some algorithm is complete, perform some empirical analysis and identify potential areas of improvement. Also, determine an 'end-game' for the project (i.e., a plan for wrapping up within 1–2 weeks).
- By draft report (4/19): Write up the complete project cycle: What is the high-level problem you wanted to tackle, your precise (technical) problem formulation, the methods / algorithms you used (including external libraries / datasets if any), the empirical results you obtained, some analysis about why the described method worked well / not well in your experiments, some future direction to address what did not work well, possibly propose some interesting extensions, and reflection on the project as a whole (e.g., surprises, challenges, etc.). The report should be relatively complete (it should be a "release candidate", in software engineering terms). See next page for more details.
- By final report (4/25): Any further work and wrapping up, if desired. How much more this contains over the presentation and draft report depends on how much you worked on the project by that point, and/or how satisfied you are with the outcome(s). If you feel you have put in sufficient time and effort into the project, have achieved the goals you set for yourself, and have a solid written report, then you do not have to do anything beyond the draft report. For others, treat this extra week as an "extension" / second chance – reflecting on our own projects, we find that just before the deadline, when in the rush to get everything working for the presentation / report, we often uncover many more things that we can and want to improve on, as well as loose ends to tie (many TODO comments in the code...). This week is for that purpose – to polish rough, freshly (under-)baked work into a complete, presentable project that you are proud of (and would willingly show to friends / family / potential employers).

5 Project Presentation

Details on the project poster presentations are forthcoming (details can be expected by 4/15).

6 Draft and Final Report

There is no specific format that the project report should follow. You should choose a structure that best tells the story of your project.

Most reports will probably have these components:

- Describe the problem on a high level, including some motivation for choosing it (just like in your presentation).
- Concrete technical problem statement, model (if applicable), inputs/outputs.
- Describe the dataset / simulator you used, if applicable.
- Methods / algorithms that you used or developed. If you are using algorithms not covered in the class, provide a description of it so we know you understand what you have been using.
- If you are basing your project on someone else's work, explain on a coarse level what they have done, and if you are doing anything different.
- Empirical results, including details on the experimental setup (be precise about what settings / data you ran experiments with). What were your hypotheses / what do you expect to see from your experiments?
- Analysis of empirical results. What worked, what did not, and why?
- Discussion / future directions (if any). What difficulties did you encounter, if any? Did anything not go according to plan? If you had more time to spend on the project, what would you have liked to do next? What advice about the project would you give to future CS 4180/5180 students?

Teams doing more theoretical projects / literature surveys / tutorials will have a somewhat different format. If you have questions about the format, please ask the course staff on Piazza.

Length

There is no set length / page limit for the report. It would probably take at least 8–10 pages (including figures) to include all of the above. You can write as much as you wish, but do not be excessive – just get to the point and provide a complete description of the project, including the components described above.

Draft report

The draft report is due on 4/19, less than a week before the final report. We *do not* expect the report to be fully written, or even contain all the results. However, we would like to see a paper with the overall structure and sections in place, some filled sections (e.g., if you wrote applicable things in the proposal / milestone reports, you can copy and paste them into the draft), possibly some sections with bullet points, and maybe even some blank or "TODO" sections. The idea is to have a top-down view of the project report (instead of bottom-up based on milestone reports), so we all have a clear picture of what is left to do in the project / write-up. Essentially, the sections that have yet to be filled in the draft becomes your to-do list in the following week while working toward the final report. We can then use this draft report as the basis of our discussion during team interviews.

7 Final Deliverables (Theoretical track)

This is more fluid and harder to predict. It depends on the chosen topic, its scope, and your team's interests. Some ideas from the past:

1. A set of lecture notes, written in a tutorial format suitable for self-learning.
2. A set of lecture slides, suitable for teaching fellow CS 4180/5180 students.
3. Illustrative examples/visualizations that aid in understanding concepts/models/algorithms regarding the chosen topic.
4. Simple and well-documented code examples for algorithms studied.
5. Jupyter notebook (or similar tutorial formats) for showing some of the above items.

Most of the past theoretical projects have included some combination of the above items (but not all!). Suggestions not included in the above list are completely welcome and encouraged – be creative! Throughout the project, there may be more frequent check-ins with Prof. Wong to jointly define what the final results of the project may look like.

8 Evaluation criteria

Since every project is different, there's no single way to grade projects, unlike most other assignments. Instead, we look at multiple dimensions that we believe are important abilities as a practitioner in RL and sequential decision making. We consider at least 5 primary dimensions:

1. Formulation: Did you clearly and precisely describe your problem as a sequential decision making problem? What approach(es) did you consider? Are you able to articulate the approach(es) you used, demonstrating that you understand the underlying techniques and algorithms?

2. Learning: What (if anything) did you have to learn beyond the course material while completing the project? This includes technical material (theory, algorithms, models), implementation (code libraries/frameworks, existing repositories, game engines), domain-related knowledge (rules/strategies of games, dataset interpretation), and more. Tell us what you had to learn and roughly how long it took you.

3. Implementation: What (if anything) did you implement? How involved was it? What parts were existing, and what was your contribution? Although it's unlikely we will actually run your code, including your code (or a GitHub link) provides evidence of implementation.

4. Results: What were the outputs/results? How was your system evaluated, and how well did it perform? For theoretical projects, how extensive are your deliverables (quality + quantity)?

5. Analysis: Did the results you obtained and their interpretation make sense? Were there any interesting findings? What further experiments did you perform/consider? If limitations were identified, what steps were taken to address them? For theoretical projects, how extensive are your explanations / comparisons (if applicable)? Is pedagogical intuition provided? Strive for greater depth on the topic you are covering than breadth.

6. Other: Size of group, etc.

We *do not* expect you to attain maximum scores on all dimensions – for example, some projects may have more learning involved (e.g., approach not covered in class, theoretical project on a new topic), while others may have more implementation (e.g., using an algorithm covered in class but implementing it in a new domain). There are many ways to achieve a successful project.

Further notes:

- Based on reports in the past, the problem/approach(es) can usually be described in greater detail. Show that you understand what you are doing. Sometimes an example helps to illustrate an abstract problem formulation / solution approach (even in practical projects).

- We are *much* more interested in the process (that reflects technical critical thinking) than in the actual final results, especially for practical projects. Many of you probably encountered unexpected challenges along the way. Do not omit these, otherwise we would not know about it! Tell us about the issues you encountered, what steps you took to address them, and why. Again, we want to see the path you took, not just the final destination.

- For theoretical projects, aim to make your materials self-contained, suitable for self-study by curious CS 4180/5180 students (or similar). Do not just provide a summary of some topic – teach them (and us). If you are submitting Jupyter notebooks, apart from the `.pynb` files, please save and upload the output as a PDF so we do not have to load the notebook itself.