CSE 4/546: Reinforcement Learning
Spring 2025

Instructor: Alina Vereshchaka

Assignment 3 - Actor-Critic

Checkpoint: April 17, Thu, 11:59pm

Due Date: April 24, Thu, 11:59pm

1 Assignment Overview

The goal of the assignment is to help you understand the concept of policy gradient algorithms, to implement the actor-critic algorithm and apply it to solve Gymnasium environments. We will train our networks on multiple RL environments.

For this assignment, libraries with in-built RL methods cannot be used. Submissions with used in-built RL methods (e.g. stable-baselines, keras-RL, TF agents, etc) will not be evaluated.

Part I [20 points] – Introduction to Actor-Critic

The goal of this part is to help you build a deeper understanding of the structure and mechanics behind Actor-Critic algorithms. You will explore how neural networks are adapted for different environment setups, how Actor and Critic components are architected, and how common practices like normalization and gradient clipping can stabilize training. This section will also prepare you to design more generalizable and reusable RL agents that can adapt to various environments.

Refer to the Jupyter notebook for more detailed instructions: UBlearns > Assignments > Assignment 3 > `a3_part_1_UBIT1_UBIT2.ipynb`

You are required to complete the following five tasks. Each task should be fully implemented in your notebook and supported with comments and brief explanations where appropriate.

TASKS

1. **[5 points] Implement Actor-Critic Neural Networks**
   Create two versions of the actor-critic architecture:
   (a) Completely separate actor and critic networks with no shared layers.
   (b) A shared network with two output heads: one for the actor and one for the critic.

   Discuss the motivation behind each setup and when it may be preferred in practice.

2. **[5 points] Auto-generate Actor-Critic Networks Based on Environment Spaces**
   Implement a general-purpose function `create_shared_network(env)` that dynamically builds a shared actor-critic network for any Gymnasium environment. Your function should handle variations in:
   - Observation spaces: e.g., discrete, box (vector), image-based.
   - Action spaces: discrete, continuous (Box), and multi-discrete.

Test your implementation on the following environments:

- `CliffWalking-v0` (Observation: integer → one-hot encoded)
- `LunarLander-v3`
- `PongNoFrameskip-v4` (use gym wrappers for preprocessing)
- `HalfCheetah-v5` (MuJoCo)

Your code will be further tested with additional environments by the course staff. The design must be generalizable.

3. **[5 points] Normalize Observations for Bounded Environments**
   Write a function `normalize_observation(obs, env)` that normalizes the input observation to a $[-1, 1]$ range using the environment's `observation_space.low` and `high` values. This normalization should be applied only for environments with `Box` observation spaces and defined bounds. Test this function using:

   - `LunarLander-v3`
   - `PongNoFrameskip-v4`

4. **[5 points] Implement Gradient Clipping**
   Demonstrate how to apply gradient clipping in your training loop using PyTorch's `torch.nn.utils.clip_grad_norm_`. Print the gradient norm before and after clipping to validate that the operation was applied correctly. You may use any of the models built in earlier tasks.

Submission Format:

- Submit your Jupyter Notebook as `a3_part_1_UBIT1_UBIT2.ipynb`.
- Ensure all cells are executed and saved with output before submission.
- Include inline comments and explanations within your code.
- No separate PDF report is required for this part.

Part II [50 points] - Implementing Advantage Actor Critic (A2C/A3C)

In this part we will implement an Advantage Actor Critic (A2C/A3C) algorithm and test it on any simple environment. A2C is a synchronous version of the A3C method.

1. Implement the A2C algorithm. You are welcome to implement A2C or A3C version of the algorithm. Any other variations will not be evaluated. You may use any framework (Tensorflow/PyTorch). **Implement at least 2 actor-learner threads.**

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$
// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal $s_t$ or $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
until $T > T_{max}$
```

2. Train your implemented algorithm on any environment. Check "Suggested environments" section.
3. Show and discuss your results after applying the A2C/A3C implementation on the environment. Plots should include the total reward per episode.
4. Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

Important Implementation Note: Multi-threaded Actor-Learner Setup

You are required to implement a multi-threaded version of the A2C or A3C algorithm that uses **at least 2 actor-learner threads**. These threads should operate independently but either synchronize periodically (A2C) or update asynchronously (A3C) using a shared model.

- Each thread must maintain its own environment instance.
- Each thread should collect experience independently (e.g., observations, actions, rewards).
- Actor and Critic gradients must either be:
  - Computed and applied **synchronously** across threads (as in A2C), or
  - Computed and applied **asynchronously** by each thread to a shared model (as in A3C).

For A2C (Synchronous Version):

- Threads run in parallel to collect experience.
- After a fixed number of steps or episodes, threads synchronize and share their gradients.
- A global model is updated using the average of these gradients.

For A3C (Asynchronous Version):

- Each thread maintains a local copy of the global model.
- Threads collect experience and compute gradients independently.

- The global model is updated asynchronously with gradients from each thread.
- Local models are periodically synced with the updated global model.

Implementation Tips:

- Use Python's `multiprocessing` or `torch.multiprocessing` libraries to spawn processes (Gym environments are not thread-safe).
- Ensure the global model is safely shared and updated using shared memory or locks.
- Start with 2 threads for simplicity and debugging, then scale up if desired.
- Consider printing from each thread to confirm parallel execution and model updates.

What to Submit:

- Your code must clearly show the use of multiple threads or processes for actor-learners.
- In your report, explain how the threads interact with the global model and how synchronization is handled.
- Include plots showing the reward per episode for each thread, or the average across all threads.
- (Optional) Discuss any tradeoffs or bottlenecks you observed when increasing the number of threads.

In your report for Part II:

1. Discuss:
    - What are the roles of the actor and the critic networks?
    - What is the "advantage" function in A2C, and why is it important?
    - Describe the loss functions used for training the actor and the critic in A2C.
2. Briefly describe the environment that you used (e.g. possible actions, states, agent, goal, rewards, etc). You can reuse related parts from your previous assignmnets.
3. Show and discuss your results after applying your A2C/A3C implementation on the environments. Plots should include epsilon decay and the total reward per episode.
4. Provide the evaluation results. Run your agent on the three environments for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.
5. Run your environment for 1 episode where the agent chooses only greedy actions from the learned policy. Render each step of the episode and verify that the agent has completed all the required steps to solve the environment. Save this render and include it in your report as clearly ordered screenshots or as a clearly named video file in your submission.
6. Provide your interpretation of the results.
7. Include all the references that have been used to complete this part

Part II submission includes

- Report (as a pdf file)
- Jupyter Notebook (.ipynb) with all saved outputs. Do not combine with the Juypter notebook from Part I.
- Saved weights of the trained models as pickle files, .h5 or .pth for each model and each environment.
- Saved video file or screenshots of one evaluation episode.
- If you are working in a team of two people, we expect equal contribution for the assignment. Provide contribution summary by each team member.

Part III [Total: 30 points] - Solving Complex Environments

In this part, test the A2C/A3C algorithm implemented in Part I on any other two complex environments.

1. Choose an environment. At least one of the environments has to be among "Suggested environments - Part II". The environment with multiple versions is considered as one environment.
2. Apply the A2C/A3C algorithm to solve it. You can adjust the neural network structure or hyperparameters from your base implementation.
3. Show and discuss your results after applying the A2C/A3C implementation on the environment. Plots should include the total reward per episode.
4. Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.
5. Go to Step 1. In total, you need to provide the results for TWO environments.

In your report for Part III:

1. Briefly describe TWO environments that you used (e.g. possible actions, states, agent, goal, rewards, etc). You can reuse related parts from your previous assignmnets.
2. Show and discuss your results after training your Actor-Critic agent on each environment. Plots should include the reward per episode for TWO environments. Compare how the same algorithm behaves on different environments while training.
3. Provide the evaluation results for each environments that you used. Run your environments for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.
4. If you are working in a team of two people, we expect equal contribution for the assignment. Provide contribution summary by each team member.

Part III submission includes

- Report (as a pdf file) – combined with your report from Part II
- Jupyter Notebook (.ipynb) with all saved outputs. It can be combined with the Jupyter notebook from Part II. Do not combine with the Jupyter notebook from Part I.
- Saved weights of the trained models as pickle files, .h5 or .pth for each model and each environment.

Suggested environments

Part I

- Your grid world defined in A1 or A2
- CartPole
- Acrobot
- Mountain Car
- Pendulum
- Lunar Lander

Part II

- Any multi-agent environment
- Car Racing
- Bipedal Walker
- MuJoCo Ant
- Any Atari env
- Any other complex environment that you will use for your Final Project

Extra Points [max +12 points]

Implement a Different Version of Actor-Critic [7 points]

STEPS:

1. Choose one of the following more complex actor-critic algorithms: PPO, TRPO, DDPG, TD3, or SAC. Implement this algorithm.
2. Apply your implementation of the chosen advanced actor-critic algorithm to the same THREE environments that you used earlier in this assignment.
3. In the report, create a new section titled "Bonus: Advanced Actor-Critic". Include the following:
   - Present three reward dynamic plots, one for each environment. Each plot should show the learning curves for both the A2C/A3C algorithm you implemented previously and the new, improved actor-critic algorithm.
   - Compare the performance of the two algorithms (A2C/A3C vs. the advanced version) across the three environments based on the plots.
   - Provide a detailed analysis discussing the observed differences in performance, potential reasons for these differences, and any insights gained from comparing the two algorithms.

SUBMISSION:

Submit your results as a Jupyter Notebook file named: $a3\_bonus\_advanced_ac\_TEAMMATE1\_TEAMMATE2.ipynb$

Solve MuJoCo Environment [5 points]

STEPS:

1. Choose one environment from the MuJoCo suite (e.g., ‘HalfCheetah-v3‘, ‘Ant-v3‘, ‘Hopper-v3‘).
2. Implement any Actor-Critic algorithm to solve the chosen MuJoCo environment. You can either use your existing A2C/A3C implementation or implement a more advanced version like PPO, TRPO, DDPG, TD3, or SAC.
3. Train your chosen Actor-Critic agent on the selected MuJoCo environment. Evaluate its performance using appropriate metrics (e.g., average reward over episodes).
4. In the report, create a new section titled "MuJoCo Environment". Include the following:
    - Present a reward dynamic plot showing the learning curve of your agent on the MuJoCo environment.
    - Describe the specific MuJoCo environment you chose and the Actor-Critic algorithm you implemented.
    - Provide an analysis of the results, discussing the performance achieved and any challenges encountered during training.

SUBMISSION:

Submit your results as a Jupyter Notebook file named: $a3\_bonus\_mujoco\_TEAMMATE1\_TEAMMATE2.ipynb$

2 References

- NeurIPS Styles (docx, tex)
- Overleaf (LaTex based online document generator) - a free tool for creating professional reports
- Gymnasium environments
- Atari Environments
- Lecture slides
- Asynchronous Methods for Deep Reinforcement Learning

3 Assignment Steps

1(a). Register your team (Due date: April 8)

- You may work individually or in a team of up to 2 people. The evaluation will be the same for a team of any size.
- Register your team at UBlearns > Groups. You have to enroll in a team on UBLearns even if you are working individually. Your teammates for A2 and A3 should be different.

1(b). For a team of 2 students (Due date: April 12)

- Create a private GitHub repository for the project and add our course GitHub account as a collaborator: @ub-rl
- Each team member should regularly push their progress to the repository. For example, you can sync the repository daily to reflect any updates or improvements made

- In your report include a link to the repository along with the contribution table. Additionally, add screenshot(s) of your commit history.

- **Report:** The report should be delivered as a separate pdf file, and it is recommended for you to use the NIPS template to structure your report. You may include comments in the Jupyter Notebook, however you will need to duplicate the results in the separate pdf file. Name your report as:
  a3_report_TEAMMATE1_TEAMMATE2.pdf
  (e.g. *a3_report_avereshc_nitinvis.pdf*)

- **Code:** Python is the only code accepted for this project. You can submit the code in Jupyter Notebook with the saved results. You can submit multiple files, but they all need to have a clear name. After executing command Jupyter Notebook, it should generate all the results and plots you used in your report and should be able to be printed out in a clear manner.

  Name your Jupyter Notebooks following the pattern:
  a3_part_1_TEAMMATE1_TEAMMATE2.ipynb
  and *a3_part_2_TEAMMATE1_TEAMMATE2.ipynb*
  (e.g. *a3_part_1_avereshc_nitinvis_.ipynb*)

- **Model Parameters:** Saved weights of the model(s) as a pickle file or .h5, so that the grader can fully replicate your results. Name your .pickle, .h5 or .pth files using the following pattern:
  a3_part_2_a2c_cartpole_TEAMMATE1_TEAMMATE2.pickle
  a3_part_2_a2c_lunar$_l$*ander_TEAMMATE1_TEAMMATE2.pickle*

2. Submit Checkpoint (Due date: April 17)

- Complete Part I & Part II.

- Submit your .ipynb with saved outputs, named as
  a3_part_1_TEAMMATE1_TEAMMATE2.ipynb
  a3_part_2_TEAMMATE1_TEAMMATE2.ipynb
  e.g. *a3_part_1_avereshc_nitinvis.ipynb*

- Report for Checkpoint submission is optional

- Submit at UBLearns > Assignments > Assignment 3

3. Submit final results (Due date: Apr 24)

- Fully complete all parts of the assignment

- Add all your assignment files in a zip folder including ipynb files for Part I, Part II, Part III Bonus part (optional) and report.

- Submit at UBLearns > Assignments

- Suggested file structure (bonus part is optional):

  assignment_3_final_TEAMMATE1_TEAMMATE2.zip

  - *a3_part_1_TEAMMATE1_TEAMMATE2.ipynb*
  - *a3_part_2_TEAMMATE1_TEAMMATE2.ipynb*
  - *a3_part_2_a2c_ENV1_TEAMMATE1_TEAMMATE2.h5*
  - *a3_part_3_TEAMMATE1_TEAMMATE2.ipynb*
  - *a3_part_3_a2c_ENV2_TEAMMATE1_TEAMMATE2.h5*

  - *a2_part_3_a2c_ENV3_TEAMMATE1_TEAMMATE2.h5*
  - *a3_report_TEAMMATE1_TEAMMATE2.pdf*
  - *a3_bonus_advanced_ac_TEAMMATE1_TEAMMATE2.ipynb*
  - *a3_bonus_mujoco_TEAMMATE1_TEAMMATE2.ipynb*

- The code of your implementations should be written in Python. You can submit multiple files, but they all need to be labeled clearly.
- Your Jupyter notebook should be saved with the results
- Include all the references that have been used to complete the assignment
- If you are working in a team of two, we expect equal contribution for the assignment. Each team member is expected to make a code-related contribution. Provide a contribution summary by each team member in a form of a table below. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	**Assignment Part**	**Contribution (%)**

4 Grading Details

Checkpoint Submission

- Graded on a 0/1 scale.
- A grade of "1" is awarded for completing more than 80% of the checkpoint related part and "0" is assigned for all other cases.
- Receiving a "0" on a checkpoint submission results in a 30-point deduction from the final assignment grade.

Final Submission

- Graded on a scale of X out of 100 points, with potential bonus points (if applicable).
- All parts of the assignment are evaluated during final evaluation.
- Ensure your final submission includes the final version of all assignment parts.

Important Notes

1. Only files submitted on UBlearns are considered for evaluation.
2. Files from local devices, GitHub, Google Colab, Google Docs, or other locations are not accepted.
3. Regularly submit your work-in-progress on UBlearns and always verify submitted files by downloading and opening them.

5 Academic Integrity

This assignment can be completed individually or in a team of two students. Teams can not be the same for A2 & A3 assignments. The standing policy of the Department is that all students involved in any academic integrity violation (e.g. plagiarism in any way, shape, or form) will receive an F grade for the course. The catalog describes plagiarism as "Copying or receiving material from any source and submitting that material as one's own, without acknowledging and citing the particular debts to the source, or in any other manner representing the work of another as one's own.". Updating the hyperparameters or modifying the existing code is not part of the assignment's requirements and will result in a zero. Please refer to the UB Academic Integrity Policy.

6 Important Information

This assignment can be completed in groups of two or individually. Teams can not be the same for A2 & A3 assignments.

7 Late Days Policy

You can use up to 5 late days throughout the course toward any assignments' checkpoint or final submission. You don't have to inform the instructor, as the late submission will be tracked in UBlearns. If you work in teams the late days used will be subtracted from both partners. E.g. you have 4 late days and your partner has 3 days left. If you submit one day after the due date, you will have 3 days and your partner will have 2 days left.

8 Important Dates

April 8, Thu, 11:59pm - Register your team (UBlearns > Groups)

April 17, Thu, 11:59pm - Checkpoint is Due

April 24, Thu, 11:59pm - Assignment 3 is Due