Homework Assignments - PA230

Overview of Assignments

For the homework assignments, you will work on implementing several reinforcement learning algorithms and evaluating them on simple environments from Gymnasium. There will be four tracks in total, each with different algorithms, environments, and rulesets. **You may work on the assignments alone or in pairs.**

The tracks are set up in a tournament-style fashion, in which you will get to compete with your classmates for the best-performing agent. The main goal is to train the strongest agent possible. However, for each track, there is also a set of minimal requirements for you to satisfy in terms of implemented features and score. In other words, you don't need to worry if the performance of your agent is not the best, as long as you have implemented the minimal feature requirements for each task and achieved the score threshold.

A short overview of the tournament tracks is given below, along with the rules for your submissions. For each track, the environment, minimal requirements for your solution, and some pointers for potential algorithmic improvements are given.

Submissions

If you work in a team of two, please make sure only one of you submits the assignments. The other team member should be only listed in the final report.

Weekly Agent Submissions: To submit your agent to the tournament, you have to upload a directory called `agent` into the file vault `Tournament` located at `https://is.muni.cz/auth/el/fi/podzim2025/PA230/ode/tourn/`. Make sure you upload to the directory corresponding to the correct track. For example, if you want to turn in a solution for the first track, upload the folder into the directory `track1_dqn`. The evaluation of your submissions will be performed on a weekly basis, and the results will be uploaded to the tournament dashboard.

Submission Rules The directory `agent` must contain two files:

1. `interface.py` - Script implementing the minimal interface required for evaluation. **Set the name of your team to the variable `TEAM_NAME` at the top of the file.**
2. `weights.pth` - Trained weights that can be loaded by the agent script.

Please submit a **zip archive** of the agent directory. On unzipping the structure should look exactly like above, with no additional parent directories.

We will not run any training for our evaluation, rather relying on the weights you've provided. Your `interface.py` script **must conform to the specified dependencies**, see the installation manual in the example submission.

For your training scripts, you're free to import other libraries, as long as you implement the RL algorithm by yourself. Utilizing third-party pre-trained policies is also forbidden.

We have a virtual environment ready on the nymfe01 machine, located in directory `/var/tmp/PA230`, which you can use to test that your submission satisfies the requirements. See the `README.md` file in the example submission for instructions on how to connect to the machine and use the environment.

Training Code Submission: By the end of the semester, you must submit the code you used for the training. For each track, you can either submit a single Python file or a zip file. The code should be submitted to file vaults `Training/NameOfTheTrack`.

Finally, when you submit your code, please make sure it is written somewhat clearly and cleanly; we would prefer not to feel like scratching our eyes out while reading it.

Training Report: You also have to upload a PDF report to the file vault under `Reports`. Add the team name and **UČO of all team members** to the report. The report should walk us through what you have submitted to the `Training` vault. It should describe the algorithms you implemented for the tournament and explain your design choices.

Do not include general explanations of Q-learning, policy gradients, or other standard methods—we are already familiar with those. Instead, focus on the specific components and mechanics you used, what worked and what didn't, and how you arrived at your parameter settings. Be honest in your descriptions: if you selected a parameter by guessing, simply state that.

The report does not need to be lengthy. Its main purpose is to persuade us that you actually understand your solutions, explain what you struggled with, and what have you learned.

Training your agents

You can find inspiration in the file `train_example.py`. It demonstrates several useful environment wrappers, includes an example of interacting with the environment, and implements TD-style evaluation of a randomized policy with a PyTorch neural network.

Debugging RL algorithms can be tricky, and it's strongly recommended that you first experiment on simpler environments (such as CartPole or Acrobot) before trying to solve the tournament ones. On average, you should be able to attain around 400 and -100 total (undiscounted) reward in these simpler environments to consider them solved.

The number of hyperparameters can also be daunting at first, but we encourage you to start with some reasonable defaults, for example, from Stable Baselines 3. You can then tune these per task once you're sure your implementation behaves correctly.

Tournament Tracks

Note that the minimal payoff refers to the expected sum of undiscounted rewards collected by the agent. We will estimate this value by sampling several episodes from the evaluation environment. For each of the episodes, we will sum all of the rewards and average these sums over the sampled episodes.

Off-Policy Learning with DQN

In this track, you will implement a variant of the classic DQN algorithm.

Environment: LunarLander-v3 (discrete)

Minimal Requirements: **DQN agent**, which utilizes **double-Q learning** to reduce bias, **target networks**, and **epsilon-greedy** exploration. The scheduling of epsilon and the target update is up to you. Soft target updates (Polyak averaging) with $\rho = 0.005$ and linear decay for epsilon are a good starting point.

Minimal Payoff: 100

Potential Improvements: See Rainbow paper for some ideas.

Policy Gradients

In this track, you will implement a policy gradient actor-critic algorithm for the same Lunar Lander environment.

Environment: LunarLander-v3 (discrete)

Minimal Requirements: A **policy gradient** agent that utilizes a **value network as a baseline** to reduce the variance of policy gradient updates. In other words, REINFORCE + value network baseline.

Minimal Payoff: 100

Potential Improvements: You should get a noticeable improvement by implementing all the goodies in common implementations of PPO (e.g., clipping the importance sampling ratio, data normalisation, GAE advantage estimates, etc.)

Racecars from Pixels

In the final two benchmarks, you will try to solve a harder task of driving a racing car from pixel observations. You will evaluate your algorithm on the version of the environment with discrete actions. For this environment, you either have to employ a convolutional architecture for your neural networks or implement an appropriate discretization wrapper for the observations.

For the first track, your algorithm must collect all data through reinforcement learning itself, while in the second track, you can also utilize behavioral cloning for warm-starting the RL training. You can collect the pretraining data in any way you see fit, as long as you implement the data collection process yourself. For example, you can implement a way that lets you drive the car around yourself and collect some expert trajectories.

Environment: CarRacing-v3 (discrete)

Minimal Requirements: There are no minimal requirements for the architecture, except that you have to turn in a solution for **at least one** of the two tracks.

Minimal Payoff: 300