METU EE 449

Computational Intelligence

Saving the Princess with a Reinforcement Learning Agent

Homework 3 - Reinforcement Learning

Due: 23:55, 03/06/2023

Late submissions are welcome, but penalized according to the following policy:

- 1 day late submission: HW will be evaluated out of 70.
- 2 days late submission: HW will be evaluated out of 40.
- 3 or more days late submission: HW will not be evaluated.

You should prepare your homework by yourself alone and you should not share it with other students, otherwise you will be penalized.

Introduction

In this homework, you will train a Reinforcement Learning agent in atari environment to win games. More specifically, you will create an RL model to play the first stage of the Super Mario Bros game. The implementations will be in Python language and you will be using PyTorch [1], OpenAI Gym [2], NES Emulator and Stable-Baselines3 [3]. You can visit the link provided in the references [1–3] to understand the usage of the libraries. You will also use TensorBoard to track the progress of your agent. You can download necessary libraries (except PyTorch, whose download method is introduced before) using **pip install gym_super_mario_bros==7.3.0 nes_py stable-baselines3[extra]**. Make sure PyTorch is already installed before installing Stable Baselines3, otherwise Stable Baselines3 may automatically download CPU version of PyTorch, regardless whether you have GPU or not.

This homework consists of some more advanced algorithms (PPO, DQN that are not provided in EE449 course material. **You do not need to worry about** the content of these algorithms as these algorithms are directly given to you in Stable Baselines library. However, you are encouraged to study these algorithms as they could be beneficial for you in the future. Also, some helper codes are provided to you under *HW3* folder in *ODTUClass* course page.

Homework Task and Deliverables

In the scope of this homework, you will use reinforcement learning algorithms to train an agent to play an Atari game. Your task will be to provide the agent with a few consecutive frames of the game as input, and train it to output a single action that maximizes its long-term cumulative reward. You will use OpenAI Gym to access the Atari game environment and the PyTorch library to implement the algorithms. By the end of the assignment, you will gain a better understanding of how reinforcement learning can be used to solve complex problems such as playing Atari games. Also, you will learn how to track your training using **TensorBoard**.

The homework is composed of 3 parts. In the first part you will answer some basic questions about RL. In the second part you will use Proximal Policy Optimization (PPO) [4] and Deep Q-Network (DQN) [5] to train your agent. In this part, you will also need to tune the hyperparameters of the algorithms and evaluate the performance of your agents using metrics such as average episode reward and win rate. Finally, you will compare the two algorithm visually and quantitatively and interpret the results by your own conclusions.

You should submit a single report in which your answers to the questions, the required experimental results (performance curve plots, visualizations etc.) and your deductions are presented for each part of the homework. Moreover, you should append your Python codes to the end of the report for each part to generate the results and the visualizations of the experiments. Namely, all the required tasks for a part can be performed by running the related code file. The codes should be well structured and **well commented**. The non-text submissions (e.g. image) or the submissions lacking comments will not be evaluated. Similarly answers/results/conclusions written in code as a comment will not be graded.

The report should be in portable document format (pdf) and named as *hw3_name_surname_eXXXXXX* where *name*, *surname* and *X*s are to be replaced by your name, surname and digits of your user ID, respectively. You do not need to send any code files to your course assistant(s), since everything should be in your single pdf file.

Do not include the codes in *utils.py* to the end of your pdf file.

1 Basic Questions

Compare the following terms in reinforcement learning with their equivalent terms in supervised learning (if any) and provide a definition for each, in your own wording:

- Agent
- Environment
- Reward
- Policy
- Exploration
- Exploitation

2 Experimental Work

In this part, you will experiment the performance of Proximal Policy Optimization (PPO) and Deep Q-Network (DQN) over an atari game. You may use the default hyperparameters and preprocessing methods given below to start, but you are expected to tune them to increase your performance (without adding too much computational load).

You may simply use the following code to start a game environment. Don't forget to load `startGameRand` from *utils.py*.

```
# Import environment libraries

import gym_super_mario_bros
from nes_py.wrappers import JoypadSpace
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT

# Start the environment

env = gym_super_mario_bros.make('SuperMarioBros-v0') # Generates the environment
env = JoypadSpace(env, SIMPLE_MOVEMENT) # Limits the joypads moves with important moves

startGameRand(env)
```

If the game runs properly, you may go on with the preprocessing steps. Also, you may save your gameplay as a video using `saveGameRand` from *utils.py*. Don't forget to install FFmpeg to your computer. There may be easier download methods for FFmpeg depending on your OS, so you can Google it.

```
# Import preprocessing wrappers

from gym.wrappers import GrayScaleObservation
from stable_baselines3.common.vec_env import VecFrameStack, DummyVecEnv, VecMonitor
from matplotlib import pyplot as plt

# Apply the preprocessing

env = GrayScaleObservation(env, keep_dim=True) # Convert to grayscale to reduce dimensionality
env = DummyVecEnv([lambda: env])
# Alternatively, you may use SubprocVecEnv for multiple CPU processors

env = VecFrameStack(env, 4, channels_order='last') # Stack frames
env = VecMonitor(env, "./train/TestMonitor") # Monitor your progress
```

Don’t forget to create `"./train/’` directory (or some alternative name) as your `CHECKPOINT_DIR` and `"./logs/’` directory as `LOG_DIR`.

2.1 PPO

Import your callback function from *utils.py*. Please note that, each weight file keeps around 250-350 MBs, so decide your save freq accordingly, not exceeding 100000. Also, import PPO function from Stable-Baselines3. Then, you may start training. Train for at least 1 million timesteps. Make sure Tensorboard is logging `ep_rew_mean` and `entropy_loss` properly.

```
from utils import SaveOnBestTrainingRewardCallback
from stable_baselines3 import PPO

callback = SaveOnBestTrainingRewardCallback(save_freq=10000, check_freq=1000,
    chk_dir=CHECKPOINT_DIR)

model = PPO(’CnnPolicy’, env, verbose=1, tensorboard_log=LOG_DIR, learning_rate=0.000001,
    n_steps=512)

model.learn(total_timesteps=4000000, callback=callback)
```

After training, it’s time to test it out. You may load your best model, or any model you want. Also, you may save your gameplay as a video using `saveGameModel` from *utils.py*.

```
model = PPO.load(’./train/best_model’)

startGameModel(env, model)
```

2.2 DQN

Repeat 2.1 with DQN algorithm.

```
from stable_baselines3 import DQN

model = DQN(’CnnPolicy’,
                    env,
                    batch_size=192,
                    verbose=1,
                    learning_starts=10000,
                    learning_rate=5e-3,
                    exploration_fraction=0.1,
                    exploration_initial_eps=1.0,
                    exploration_final_eps=0.1,
                    train_freq=8,
                    buffer_size=10000,
                    tensorboard_log=LOG_DIR
                )

model.learn(total_timesteps=4000000, log_interval=1, callback=callback)
```

3 Benchmarking and Discussions

3.1 Benchmarking

Play with the hyperparameters of the two abovementioned RL algorithms in order to get the best result. Also play with the preprocessing methods given in Gym and Stable-Baselines3 libraries. Make sure you are comparing PPO and DQN with the same preprocessing methods at least once (try to coincide their best results). Train each PPO and DQN scenario with 3 different hyperparameter, preprocessing methods for at least 1 million timesteps. Using Tensorboard module,

- Plot 3 different PPO scenario for `ep_rew_mean` value in one figure,
- Plot 3 different PPO scenario for `entropy_loss` value in one figure,
- Plot 3 different DQN scenario for `ep_rew_mean` value in one figure,
- Plot 3 different DQN scenario for `entropy_loss` value in one figure,
- Plot PPO vs DQN comparison for `ep_rew_mean` where they are using same preprocessing methods in one figure,
- Plot PPO vs DQN comparison for `entropy_loss` where they are using same preprocessing methods in one figure,

Then, decide on your best algorithm, hyperparameter and preprocessing triplet and train your model with 5 million timesteps in total. Plot this final figure for `ep_rew_mean` and `entropy_loss`. Additionally, using the `saveGameModel` function provided in *utils.py*, create a video of your best game and upload it to YouTube as unlisted. Share the YouTube link in your homework PDF file.

Finally, change your environment using to `gym_super_mario_bros.make('SuperMarioBrosRandomStages-v0')`. This will generate a random stage of the game in each time. Use your best models in each PPO and DQN to play in this environment for couple of times. Observe the generalizability of your models.

3.2 Discussions

Answer the following questions:

1. Watch your agent's performance on the environment using saved model files at timesteps 0 (random), 10000, 100000, 500000 and 1 million. Could you visually track the learning progress of PPO and DQN algorithms? When did Mario be able to jump over the longer pipes? What are the highest scores your agent could get (top left of the screen) during those timesteps?
2. Compare the learning curves of the PPO and DQN algorithms in terms of the average episode reward over time. Which algorithm learns faster or more efficiently?
3. How do the policies learned by the PPO and DQN algorithms differ in terms of exploration vs exploitation? Which algorithm is better at balancing these two aspects of the learning process?
4. Compare the performances of the PPO and DQN algorithms in terms of their ability to generalize to new environments or unseen levels of the game. Which algorithm is more robust or adaptable?
5. How do the hyperparameters of the PPO and DQN algorithms affect their performance and learning speed? Which hyperparameters are critical for each algorithm, and how do they compare?
6. Compare the computational complexity of the PPO and DQN algorithms. Which algorithm requires more or less computational resources, and how does this affect their practicality or scalability?
7. Considering what you know about Neural Networks by now, if `'MlpPolicy'` was used instead of `'CnnPolicy'`, how it would affect the performance of the algorithms in terms of handling high-dimensional input states, such as images? Which policy is better suited for such tasks, and why?

4 Remarks

1. On a newer generation GPU, a training process with 1 million steps takes around 2.5 hours. On an average GPU (including the ones in Google Colab), it takes around 4-5 hours. Hence you are strongly advised not to do your homework on the last day of submission.

2. Do not expect the perfect gameplay in each time. Your agent may fail lots of times, you can share the ones where your agent successfully completes the level.

3. Tensorboard is a tool for tracking your learning progress in ML experiments. It should be automatically installed with Stable Baselines3. You may check how to use it from this link. It is very simple.

References

[1] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library." `https://pytorch.org`, 2019.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[3] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.