EARIN
Laboratory report
EXERCISE 6: Reinforcement learning

Bartłomiej Mastej & Paweł Borsukiewicz

Warsaw University of Technology, Warsaw, Poland

1 Introduction

The aim of the experiment was to train reinforcement learning models based on CarRacing-v0 environment and subsequently assess and compare their results. There was prepered the environment using the openAI gym for the 'CarRacing-v0' problem. Further, there was created the Proximal Policy Optimization model which was further tuned with the usage of the Grid Search algorithm. Finally, the results are assessed and concluded.

2 Implementation

First of all, there was prepared environment using the openAI gym environment. It was decided to use 'CarRacing-v0' gym instead of 'CarRacing-v1' due to high availability of libraries supporting reinforcement learning.

2.1 Prerequisites

At the very beginning, there was a need to configure the environment to work properly with the *gym* module as well as with the *stable_baselines*3. Furthermore, in order to make all modules work, the proper version of each of them had to be installed. The final configuration of the environment is presented on the Fig. 1.

2.2 Model & learning procedure

It was decided to use the PPO - Proximal Policy Optimization model from the *stable_baselines*3 python module due to the ease of implementation and tuning. Furthermore, this module is implemented with the usage of the $TensorFlow$ module, hence the learning can be done with the GPU computational support. The implementation of the model creation with the parameters tuning is presented on the Listing. 1.1. Further, there was used the *learn* method of the module which take the timestep as the parameter. The duration of the model learning is being saved as shown on the Listing. 1.1. Finally, there is a need to

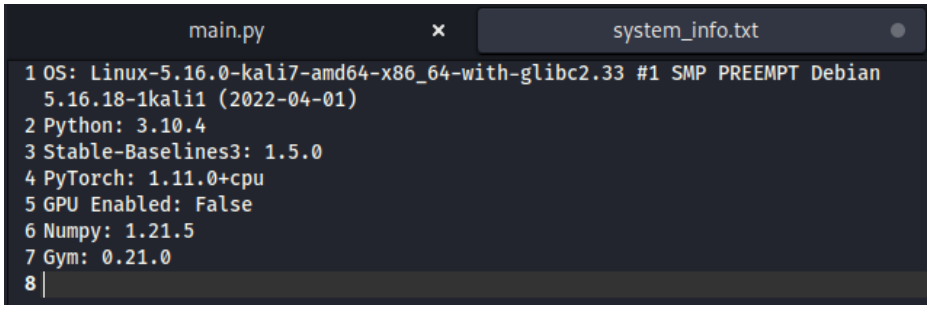

Fig. 1: Configured environment parameters & libraries

give a reward for the given simulation as well as the standard deviation. Therefore, there was used the *evaluate_policy* function also from the *stable_baselines*3 module. As described in the subsection 3 the best model is being saved to a zip file.

The positive reward is being given to a car while it drives through the tiles on the track, and the negative reward is being given if the car drives through the tiles off the track.

Listing 1.1: Model creation

```
#Prepare model
model = PPO(params['policy'], \
            env,\
            verbose=0, \
            learning_rate=params['l_rate'], \
            tensorboard_log=log_path, seed=2137, \
            n_steps=params['n_steps'], \
            n_epochs=params['n_epochs'])

#Perform learning procedure
print("LEARNING", iter, "/", len(grid))
start = timeit.default_timer()
model.learn(total_timesteps=params['timesteps'])
stop = timeit.default_timer()
l_time = "{0:0.3f}".format(stop - start)

#Evaluate learning outcomes
print("EVALUATING", iter, "/", len(grid))
#Returns (Mean reward, Standard deviation)
#Change render to true to see results
mean, std = evaluate_policy(model, env,\
                            n_eval_episodes=3, \
                            render=False)
print("Result:", mean)
```

2.3 Grid Search parameters tuning

For the scope of tuning there was used Grid-Search process the following parameters were checked: learning rate, policy, number of steps, number of epochs, and the timestep. The implementation of the Grid Search was done with the usage of the *sklearn* module and it can be seen on the Listing. 1.2. It was decided to use two different policies: *MlpPolicy* (multi-layer perceptron, which was described in detail in the previous laboratory report) and *CnnPolicy* (convolutional neural networks which the most common application is to analyze visual imagery, hence it was decided to test them with the problem). Moreover, there are different learning rates applied so as to find the more optimal solution (the duration of learning process vs optimal weights changing). Next parameters is *n_steps* are and *n_epochs* where the fist one determines the number of steps to process the one batch of data and the second one determines the number of complete passes through the training data set. Finally, the *timesteps* is the number of steps during each the single action will take place and the reward is being given.

Listing 1.2: Grid Seach parameters

```
#Variables for grid search
param_grid = {'l_rate': [0.0001, 0.001, 0.01, 0.1], \
              'policy' : ["MlpPolicy", "CnnPolicy"], \
              'n_steps': [1024, 2048, 4096], \
              'n_epochs': [5, 10, 20], \
              'timesteps': [1000, 10000]}

grid = ParameterGrid(param_grid)
```

Further, there was created the main loop which iterated through all the grid search parameters. The code that is presented on the Listing. 1.1 and Listing 1.4 are inside that loop.

2.4 Results saving/loading

Before main loop of the Grid Search was performed, there was created the *results.csv* file that is presented on the Listing. 1.3 in which there are saved the parameters of the given iteration as well as the mean reward value and the network learning time. The save of the current results takes place just after learning outcomes are achieved Listing. 1.4. Furthermore, it was decided to save only the best pretrained model of the given program execution as can be seen on the Listing. 1.4. The model is being saved in the zip format. The pretrained network can be easily loaded from the file as presented on the Listing. 1.5.

Listing 1.3: Log files creation

```
#Save logs to file
log_path = ('Logs')
filename = "Logs/Results3.csv"
```

Listing 1.4: Pretrained model and parameters saving

```
#Save results
with open(filename,"a") as my_csv:
    csvWriter = csv.writer(my_csv,delimiter=';')
    csvWriter.writerow([params['l_rate'], \
                        params['policy'], \
                        params['n_steps'], \
                        params['n_epochs'], \
                        params['timesteps'], \
                        l_time, mean])

#Check if it is currently the best model
if mean > highscore:
    highscore = mean
    print("New highscore: ", highscore)
    #Save the best model to file in a .zip format
    print("SAVING TO FILE")
    model.save('Logs/Model')
```

Listing 1.5: Loading pretrained network

```
model.load('Logs/Model')
```

3 Resuls & conclusions

Firstly, as described the results from each iteration were saved to a file in a *.csv* format. Further, the data was sorted descending starting with the highest mean score of the reward. The results for the best 30 networks are presented on the Fig. 2. As it can be easily concluded the *CnnPolicy* gives significantly better results than the *MlpPolicy* as in the best 30 results there are only 4 *MlpPolicy* based results, none of which are in the best 10. The best result for the *CnnPolicy* is 3 times better then the best result obtained with the *MlpPolicy* which indicates that it is much better policy for this task. Furthermore, as it can be observed on the Fig. 3 the highest results for the whole data sets are for the number of step equal to 2048, however, more compressed results are obtained for the number of steps equal 4096. On the Fig. 4 there is presented mean score vs learning rate and on the Fig. 5 there is presented learning rate vs time chart. The conclustions from those two figures are that for this task the best learning rate is the smallest one equal 0.0001 as not only it gives the best results but also it does not differ significantly from the duration of the execution from 100 higher learning rate which brings the worst results. As it can be easily concluded with the smaller learning rate the results significantly improve, however, as a consequence the execution time extends.

Finally, the best model was rendered and tested. On the Fig. 6 there can be seen the car being perfectly on track and it's score is increasing, and while

l_rate	policy	n_steps	n_epochs	timesteps	TRAINING TIME	MEAN SCORE
0.0001	CnnPolicy	2048	20	10000	290.057	164.13
0.0001	CnnPolicy	2048	20	10000	192.517	134.14
0.0001	CnnPolicy	4096	5	1000	63.527	117.70
0.001	CnnPolicy	4096	10	10000	286.375	106.82
0.0001	CnnPolicy	4096	10	10000	309.846	94.81
0.0001	CnnPolicy	4096	5	10000	160.375	83.35
0.001	CnnPolicy	2048	5	1000	27.732	82.65
0.0001	CnnPolicy	4096	10	10000	224.657	77.47
0.0001	CnnPolicy	4096	10	10000	224.657	77.47
0.0001	CnnPolicy	2048	10	1000	38.809	77.30
0.0001	CnnPolicy	1024	20	10000	404.265	68.17
0.001	CnnPolicy	2048	20	10000	385.04	65.69
0.001	CnnPolicy	2048	10	1000	38.874	55.26
0.001	MlpPolicy	4096	5	1000	34.068	54.38
0.0001	CnnPolicy	4096	20	10000	367.386	53.29
0.001	CnnPolicy	1024	20	1000	34.554	47.82
0.0001	CnnPolicy	2048	20	1000	57.397	45.50
0.001	CnnPolicy	4096	5	10000	193.82	34.64
0.0001	CnnPolicy	4096	20	1000	76.385	33.92
0.001	MlpPolicy	2048	5	10000	123.38	29.74
0.001	CnnPolicy	1024	20	1000	21.999	29.08
0.001	MlpPolicy	4096	5	1000	49.771	27.37
0.0001	CnnPolicy	4096	10	1000	75.882	17.73
0.0001	CnnPolicy	4096	10	1000	75.882	17.73
0.001	CnnPolicy	2048	10	1000	27.554	16.34
0.01	CnnPolicy	2048	20	1000	38.206	13.25
0.001	MlpPolicy	2048	5	10000	86.056	8.48
0.001	CnnPolicy	2048	20	1000	41.531	8.19
0.001	CnnPolicy	1024	5	10000	100.376	5.19

Fig. 2: 30 Highest mean rewarded networks with their parameters

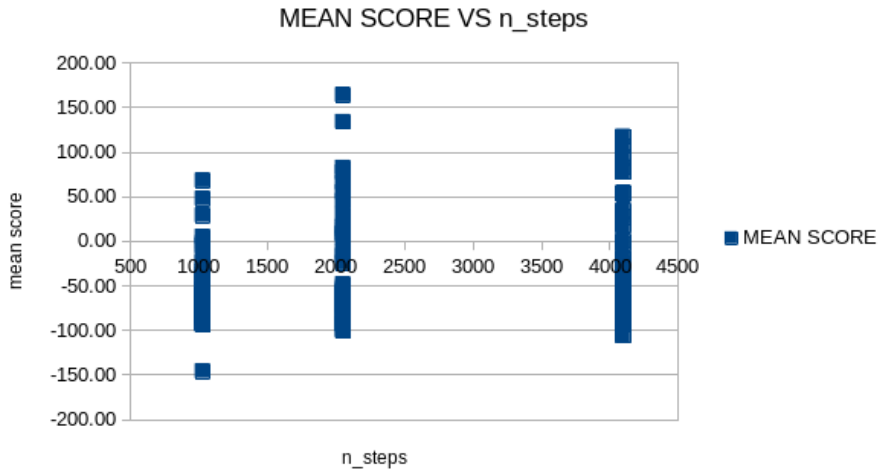

Fig. 3: The mean award value vs number of steps

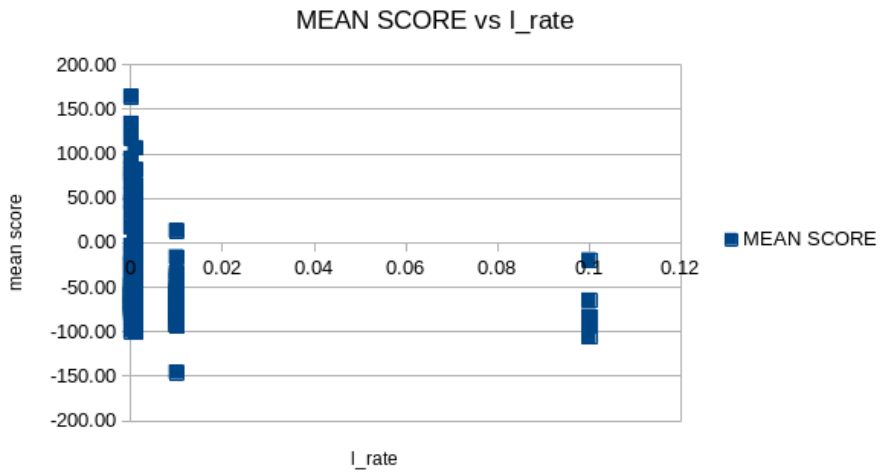

Fig. 4: The mean award value vs learning rate

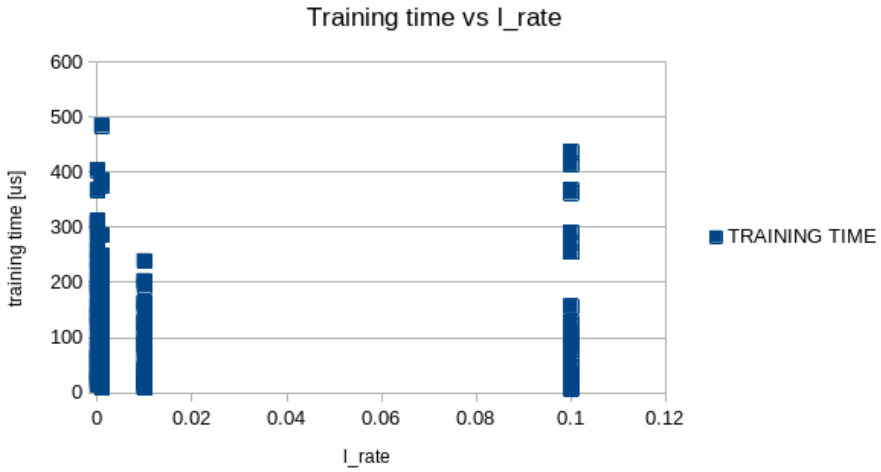

Fig. 5: learning rate vs time

it stays on this track it keeps it's position about in the middle of the track. Nevertheless, it struggles during the turns as it is very likely to go off the track and then spin around as presented on the Fig. 7, where although it has relatively high score, which was gained while being on track, it is lost. However, if it finally return on track it quickly stablize and keeps it's position in the middle of the track.

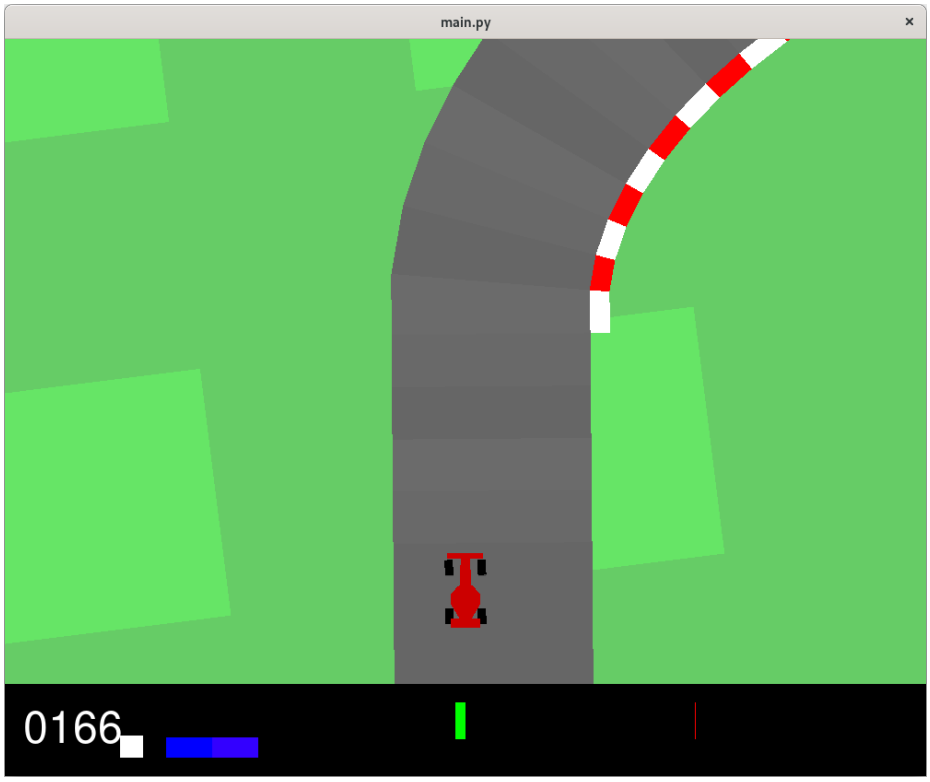

Fig. 6: Initial on track ride

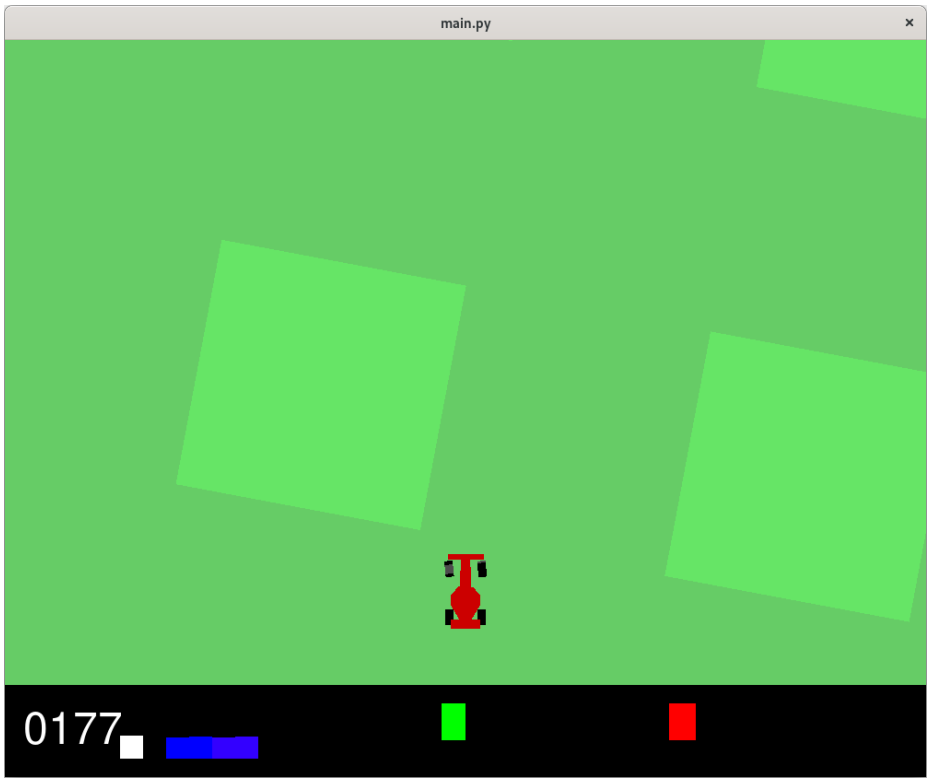

Fig. 7: Off track ride