EARIN
Introduction to Artificial Intelligence

Exercise 6: Reinforcement Learning

Julia Czmut, 300168
Laura Ploch, 300176

1. Introduction

Reinforcement learning is a machine learning training method based on rewarding desired behaviours and/or punishing undesired ones. In this approach, a reinforcement learning **agent** is able to perceive and interpret its environment, take actions and learn through trial and error.

Rewarding desired behaviours and punishing undesired ones is done by assigning positive values to desired actions and negative values to undesired actions. This programs the agent to seek long-term and maximum overall reward to achieve an optimal solution.

A common application of reinforcement learning are games, which is also the topic of this task. The goal of this task was to set up an openAI gym environment for the Car Racing game, train the model with different policies and then compare the obtained results.

2. Methodology

To help us train the model, we used the **StableBaselines** module which contains implementations of reinforcement learning algorithms in Pytorch.

Because our task is to compare results of training the models with different policies (suggested actions that the agent should take for every possible state), we have chosen one of the on-policy algorithms – **PPO**.

On-Policy learning algorithms evaluate and improve the same policy, which is being used to select actions, so Target Policy is the same as Behaviour Policy. A2C and PPO are examples of this type of RL algorithms.

Off-Policy learning algorithms evaluate and improve a policy that is different from the policy used for action-selection, so Target Policy is different from Behaviour Policy. Q-Learning is an example of such RL algorithm.

Preparation of the environment

The openAI gym environment we are using is **'CarRacing-v1'** since the previous version has been deprecated in the current version of the gym library.

Preparation of the model

Our model is trained using the **PPO** algorithm.

"**Proximal Policy Optimization** algorithm combines ideas from A2C (having multiple workers) and TRPO (it uses a trust region to improve the actor).

The main idea is that after an update, the new policy should be not too far from the old policy. For that, PPO uses clipping to avoid too large update."

[source: https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html]

PPO generally takes less time to train than other algorithms and additionally, it gives stable results, which is why we think it is a good choice for this task. It will give us more secure conditions for us to focus on comparison between the two chosen policies.

Policies used for comparison

1. CNN Policy

2. MLP Policy

3. Implementation and results

Initialization of the environment

```
import gym
from stable_baselines3 import PPO
from stable_baselines3.common.vec_env import VecFrameStack
from stable_baselines3.common.evaluation import evaluate_policy

env = gym.make('CarRacing-v1')
```

Training the CNN model

learning rate = 0.0001
number of epochs = 20
total timesteps = 100 000

```
env.reset(seed=2022)
model_cnn = PPO('CnnPolicy', env, verbose=1, learning_rate=0.0001, n_steps=2048, n_epochs=20)
model_cnn.learn(total_timesteps=100000)
print("Saving to file")
model_cnn.save('ModelCnn')
```

Training the MLP model

learning rate = 0.0001
number of epochs = 10
total timesteps = 100 000

```
env.reset(seed=2022)
model_mlp = PPO('MlpPolicy', env, verbose=1, learning_rate=0.0001, n_steps=2048, n_epochs=10)
model_mlp.learn(total_timesteps=100000)
print("Saving to file")
model_mlp.save('ModelMlp')
```

Calculating mean reward and std reward for CNN model

```
mean_reward_cnn, std_reward_cnn = evaluate_policy(model_cnn, model_cnn.get_env(), n_eval_episodes=3)
print("CNN policy")
print("Mean reward = ", mean_reward_cnn)
print("Std reward = ", std_reward_cnn)
```

```
CNN policy
Mean reward =  432.201396766666667
Std reward =  74.3333333333333334
```

Calculating mean reward and std reward for MLP model

```
mean_reward_mlp, std_reward_mlp = evaluate_policy(model_mlp, model_mlp.get_env(), n_eval_episodes=3)
print("MLP policy")
print("Mean reward = ", mean_reward_mlp)
print("Std reward = ", std_reward_mlp)
```

```
MLP policy
Mean reward =  -69.60057766666667
Std reward =  1.7101932468118644
```

Observations

Based on the results of the reward metrics we can see that the model trained with the CNN policy achieved a far better score – the mean reward is equal to about 432 and the standard deviation of the reward is equal to about 74, as opposed to a negative mean reward in case of MPL policy.

The parameters (learning rate, number of steps, number of epochs and total number of timesteps) were almost the same for both performed policies, except for the number of epochs. However, our previous experiments proved that increasing the number of epochs in MPL model training doesn't yield better results, but takes longer for the model to learn, so we decided to stick to 10 epochs.

During the simulation we could observe that the CNN agent had a careful approach, riding on medium speed and staying consistently on the track, while the MLP agent often drifted off the track, sometimes falling off the map or riding around the map completely missing the road, doing drifts and running in circles occasionally.

4. Summary

The model trained with the **CNN policy** network proved to be **incomparably better** for this task than the one trained with the **MLP policy**. It can be concluded both by looking at the evaluation metrics**,** as well as the simulation of the trained agents playing the car racing game.

The reason for this difference between the policies' performance might be that the CNN policy network is designed for images, while the MLP policy network is generally for other types of input features. So, because this task specifically had image inputs, it could be expected from the start that the CNN policy would perform better.