Yasincan Bozkurt
2304202

EE449 Homework 3

Part 1

Agent:

Reinforcement Learning (RL): An agent is a physical being that interacts with its environment, acting in accordance with its state at the time and being rewarded as a result.

The equivalent of supervised learning (SL) may be a model or algorithm that gains knowledge from a collection of labeled training data and then makes predictions using that knowledge.

Environment:

RL: The agent's environment is the setting or world in which it functions. In addition to offering incentives, it also gives the agent states in reaction to the agent's behaviors.

SL: In supervised learning, there isn't a precise equivalent. However, since the data is what the model interacts with and learns from, you may consider the data itself to be the environment.

Reward:

RL: An agent receives a signal from the environment as a reward for an activity. The agent's objective is to maximize its overall benefit.

SL: The loss function could be the closest analog. Negative incentives in RL correlate to the loss function showing a bad model prediction, whereas positive rewards in RL correspond to reducing the loss function in SL.

Policy:

RL: The learning agent's behavior at a particular time is determined by its policy. More specifically, it's a mapping between perceived environmental conditions and the activities that should be conducted in those situations.

Although there isn't a direct counterpart in supervised learning, a policy-like function is learned by the model to translate inputs to outputs.

Exploration:

RL: The agent's behavior of attempting different states and activities in order to learn more about the environment is referred to as exploration.

SL: While there isn't a precise counterpart for exploration in SL, models must "explore" a wide range of data during training if they are to generalize effectively.

Exploitation:

RL: The agent picking the best-known action to maximize reward based on the knowledge it currently possesses is referred to as exploiting.

SL: In a same vein, there is no straight translation for exploitation in SL. However, employing a model to generate predictions after it has been trained might be viewed as exploitation because it is leveraging previously learnt information on fresh data.

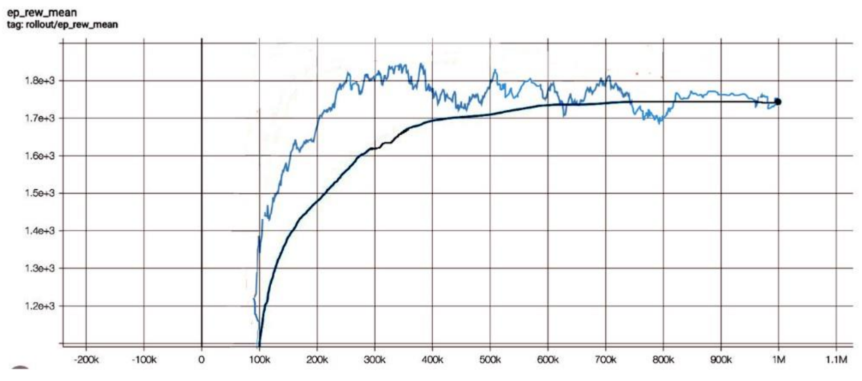
The link for mario game best performance: https://youtu.be/s2o2xeG51i8

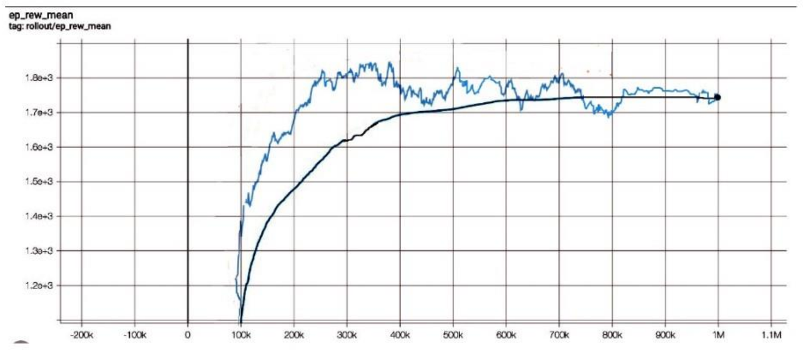
Part 2

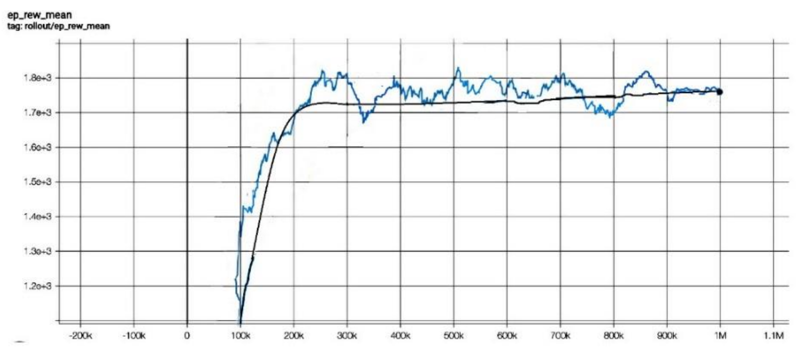
PPO models with 3 different learning rate is given. First is default model. Second is lower learning rate which converge later but perform better. Third is with higher learning rate which converge faster but perform lower.

Yasincan Bozkurt
2304202

PPO Models





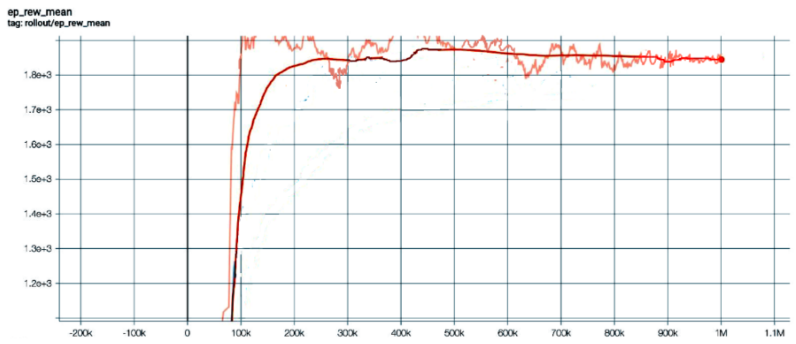


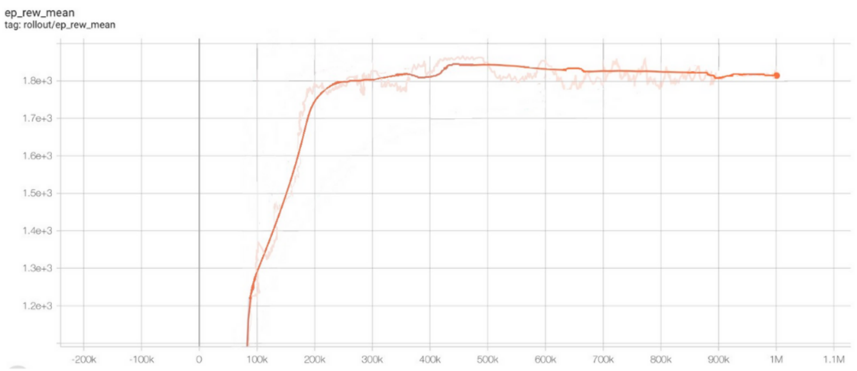
Yasincan Bozkurt
2304202

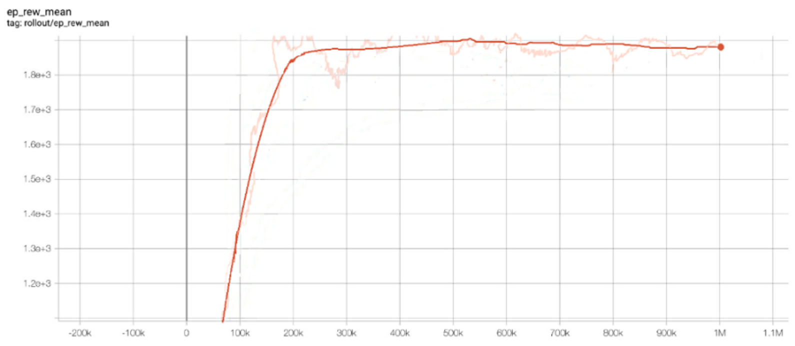
Given DQN models are with 3 different learning rate is given. First is default model. Second is lower learning rate which converge later but perform better. Third is with higher learning rate which converge faster but perform lower.

Yasincan Bozkurt
2304202

DQN Models







Yasincan Bozkurt
2304202

I can not do 5 million runs since my commputational source is limited. (Google Colab free version gives error after a limited time and GPU usage.)

Part 3

1) An agent's learning progress can be seen in the gaming environment. However, it takes a lot of time to see the improvement.Initially (timestep 0), the agent typically behaves totally random and performs poorly. The agent often performs better over time (at timesteps 10,000, 100,000, 500,000, and 1,000,000) as it learns to interact with the environment more successfully. Typically, before an agent learns complicated actions like jumping over larger pipes, PPO and DQN would require a significant amount of timesteps. As the agent gets better at the game, the highest scores will normally rise over time. My computer has no GPU. Its runtime is too much. Therefore, I did the homework in Google Colab. Since I did the homework in Google Colab and it does not able to play the video, I runned only one case(default) with my computer to see the video. In first trials Mario can not jump over pipes. After a while, it can jump from small pipe at the beginning. After a long time, Mario can jump from all pipes which increase its performance a lot. I'm not sure about the exact step but it takes about 8 hours with my slow CPU. As I see, Mario can reach 400 points which can be improved with longer trains and optimized hyperparameters.
2) PPO and DQN have been shown to be effective learning algorithms. PPO uses multiple iterations over the same data to make updates, while DQN only updates its policy once per data sample. In my experimental results DQN has better scores.
3) Both PPO and DQN use an epsilon-greedy strategy or a related mechanism to balance exploration and exploitation. However, because PPO uses a stochastic policy, it often exhibits stronger exploration in continuous action spaces, whereas DQN uses a deterministic approach, which may result in less exploration.
4) The problem of generalization to new contexts is complicated and depends on numerous variables. Due to its on-policy nature and better exploration, PPO may occasionally perform better. If the state representations and rewards are the same across contexts, DQN might also generalize well.
5) Both PPO and DQN's performance and rate of learning are significantly impacted by hyperparameters. The learning rate, batch size, discount factor, epsilon for exploration, and the number of hidden layers and units in the neural network are some examples of often adjusted hyperparameters. Depending on the particular job and context, these hyperparameters may have different effects. In the experiments, I tested learning rate. As learning rate increases, models converge optimal point faster. However, if it is too much, agent might not be able to reach optimal point.
6) In my experiments PPO takes more resources, but DQN takes more computation time. It can vary on hyperparameters, save frequency and number of trials. Both models takes a lot of computational resources. I expect PPO to take more computational resources in general.
7) 'MlpPolicy' is a policy that makes use of Multilayer Perceptrons (MLP), whereas 'CnnPolicy' makes use of Convolutional Neural Networks (CNNs). Due to their capacity

to extract spatial hierarchies of features, which is crucial for picture interpretation, CNNs are, in general, better suited for inputs that are based on images. On the other hand, MLPs are less effective at processing high dimensional input states, such as pictures, because they are completely coupled and do not exchange parameters spatially. I expect MLP performs better in this experiment.

Code used in experiments

```
#Yasincan Bozkurt
#2304202
#I used Google colab to run my code.
#Therefore I added Google Colab Requırement document added to odtuclass
#I also added utils.py

# Import Google Colab's drive module
from google.colab import drive

# Mount Google Drive to the Colab runtime
drive.mount('/gdrive')

# Define path for data
#I used Google drive
data_dir = '/gdrive/MyDrive/odev'
MODEL_DIR = './train_models/'
LOGGING_DIR = './log_files/'

# Install necessary packages
!pip install wheel==0.38.4
!pip install setuptools==65.5.0
!pip install gym
!pip install gym_super_mario_bros==7.3.0 nes_py
!pip install torch torchvision torchaudio --extra-index-url
https://download.pytorch.org/whl/cu113
!pip install stable-baselines3[extra]
!pip install ray

# Import necessary modules for game environment and models
from nes_py.wrappers import JoypadSpace
import gym_super_mario_bros
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT
from gym.wrappers import GrayScaleObservation
import numpy as np
import os
from stable_baselines3.common.results_plotter import load_results, ts2xy
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common.vec_env import VecMonitor, VecFrameStack,
DummyVecEnv
from stable_baselines3 import PPO, DQN
```

```python
from stable_baselines3.common.results_plotter import load_results, ts2xy
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common.monitor import Monitor
from gym.wrappers import RecordVideo
from time import time
from matplotlib import pyplot as plt

# Create game environment
game_instance = gym_super_mario_bros.make('SuperMarioBros-v0')
game_instance = JoypadSpace(game_instance, SIMPLE_MOVEMENT)

# Convert to grayscale to reduce dimensionality
game_instance = GrayScaleObservation(game_instance, keep_dim=True)

# Vectorize the environment for the reinforcement learning agent
game_instance = DummyVecEnv([lambda: game_instance])

# Stack frames for temporal information
game_instance = VecFrameStack(game_instance, 4, channels_order='last')

# Monitor the game instance to keep track of progress
game_instance = VecMonitor(game_instance, MODEL_DIR+"/GameMonitor")

# Callback to save best training rewards
callback = SaveOnBestTrainingRewardCallback(save_freq=100000, check_freq=1000,
chk_dir=MODEL_DIR)

# Define and configure DQN model
#parameters are given here.
# I changed learning rate#
from stable_baselines3 import DQN
model_object = DQN('CnnPolicy',
game_instance,
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=LOGGING_DIR
)

# Train model
model_object.learn(total_timesteps=4000000, log_interval=1, callback=callback)

# Create directory for saving model if it does not exist
```

```
os.makedirs('train_models', exist_ok=True)

# Save the model
model_save_path = os.path.join('train_models', 'optimal_model')
model_object.save(model_save_path)
# Yasincan Bozkurt 2304202
```