Erkan Dogan
2166262
2023-06-11

EE449 HW3

Youtube Link: https://youtu.be/SQecAYX6gro

Note: When I observed my model with StartGameModel, I see my model goes to stage 2 after 2 hours. But although I recorded many steps, I did not observe it in my recordings. I am sharing my best recording after many tries.

1. Basic Questions

Term	Reinforcement Learning	Supervised Learning
Agent	Agent is our subject that uses actions, observe the environment state, and collect rewards or penalties to maximize the total reward to reach optimal policy.	In supervised learning, there is no corresponding term for agent.
Environment	Environment is the context where the agent follows the actions that are defined with the policy. With the actions that agent takes, environment provides feedback to the agent so that agent knows its state.	In supervised learning, there is no corresponding term for environment.
Reward	Reward (or punishment for negative reward) is the resulting value from the agent's action. Agent's goal is to maximize the cumulative reward value.	In supervised learning, there is no corresponding term for reward. However, the goal to minimize error can be used as an example for reward correspondence.
Policy	Policy is mapping of states with the action probability distributions to the other states.	In supervised learning, the corresponding term can be model classifiers since with classifiers we create a relation between input and output features.
Exploration	Exploration is where the agent tries different actions to find better options for the policy.	In supervised learning, there is no corresponding term for exploration.
Exploitation	Exploitation is where the agent uses its previous experience reward information to find best rewards.	In supervised learning, there is no corresponding term for exploitation.

2. Experimental Work (No questions asked)

PPO_1: Default parameters given in the homework.

PPO_2: n_steps changed (256->512)

PPO_3: VecFrameStack changed (4->3)

DQN_4: Default parameters given in the homework.

DQN_5: batch_size changed (192->384)

DQN_6: VecFrameStack changed (4->5)

Erkan Dogan
2166262
2023-06-11

3. Benchmarking and Discussions

3.1. Benchmarking

3.1.1. Plot 3 different PPO scenario for ep_rew_mean value in one figure.

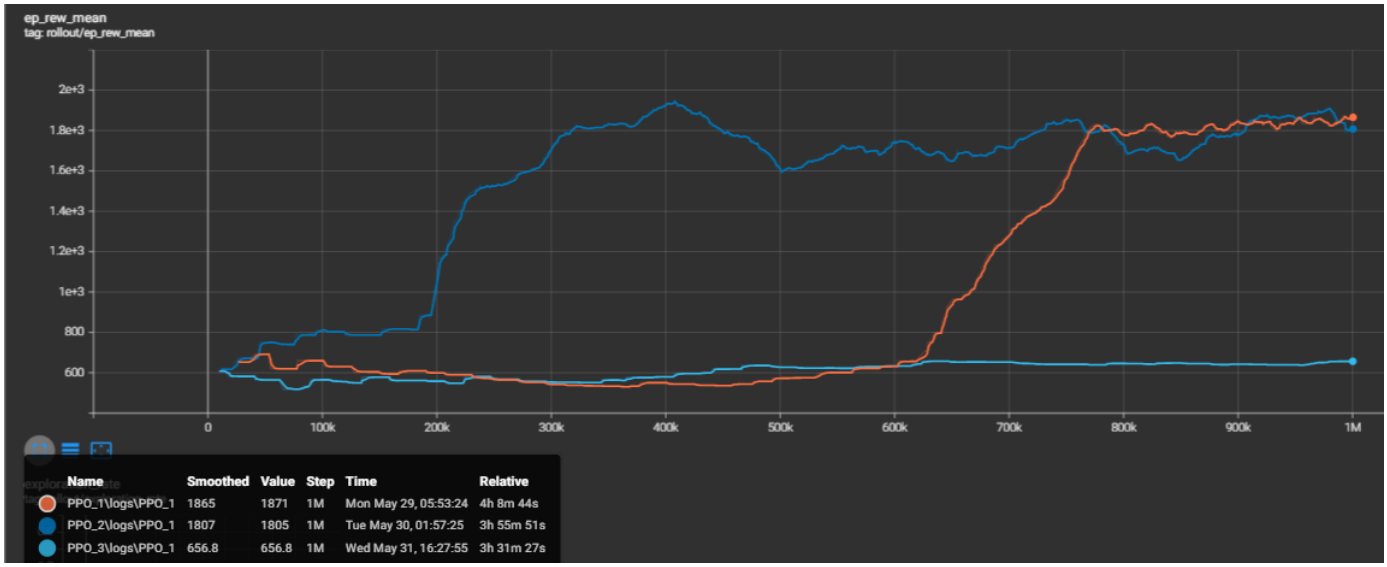

Figure 1 ep_rew_mean vs time step graph for PPOs (smoothing = 0)

3.1.2. Plot 3 different PPO scenario for entropy_loss value in one figure

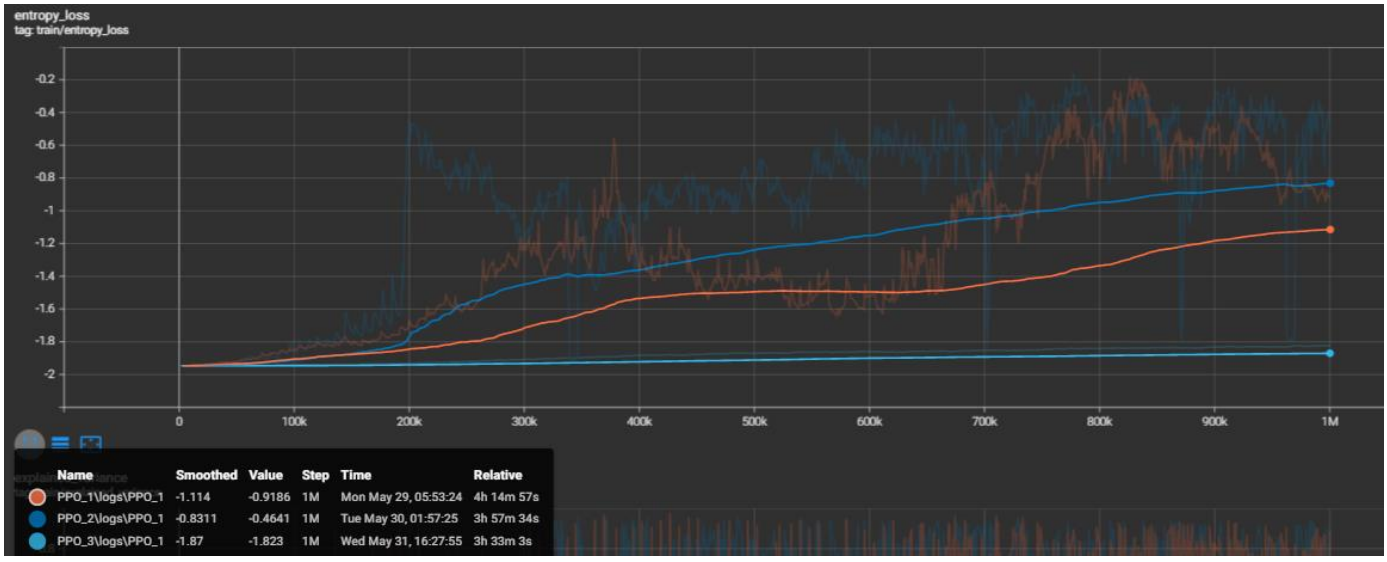

Figure 2 entropy_loss vs time step graph for PPOs (smoothing = 1)

Erkan Dogan
2166262
2023-06-11

3.1.3. Plot 3 different DQN scenario for entropy_ep_rew_mean value in one figure

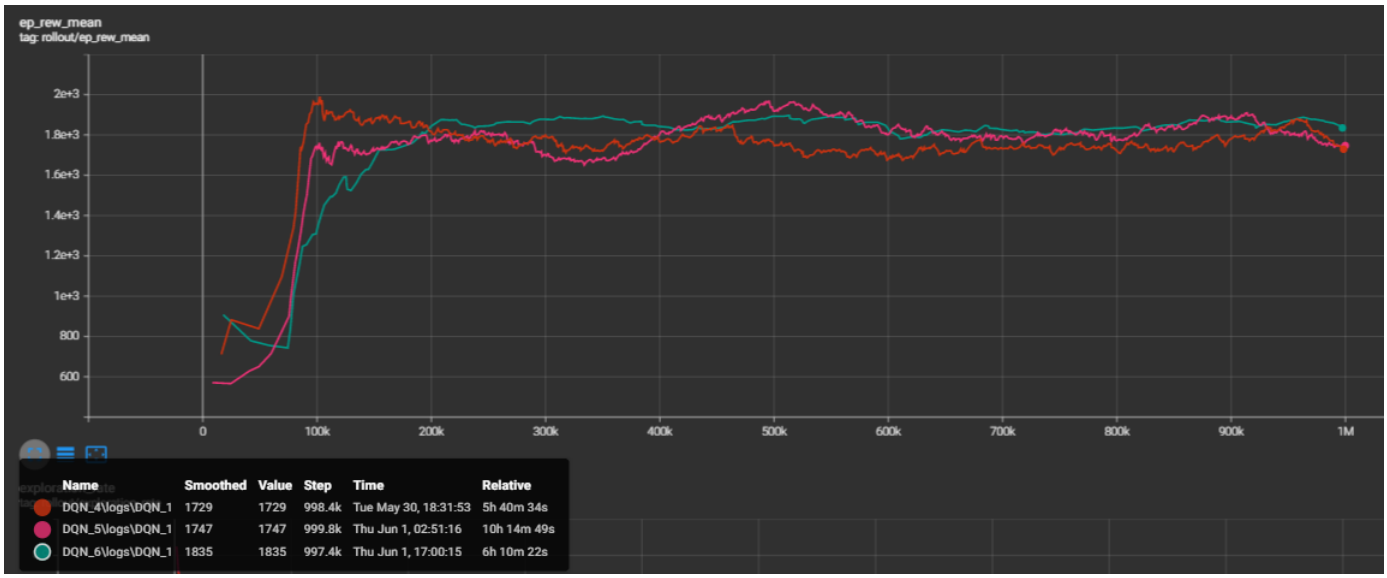

Figure 3 ep_rew_mean vs time step graph for DQNs (smoothing = 0)

3.1.4. Plot 3 different DQN scenario for loss value in one figure

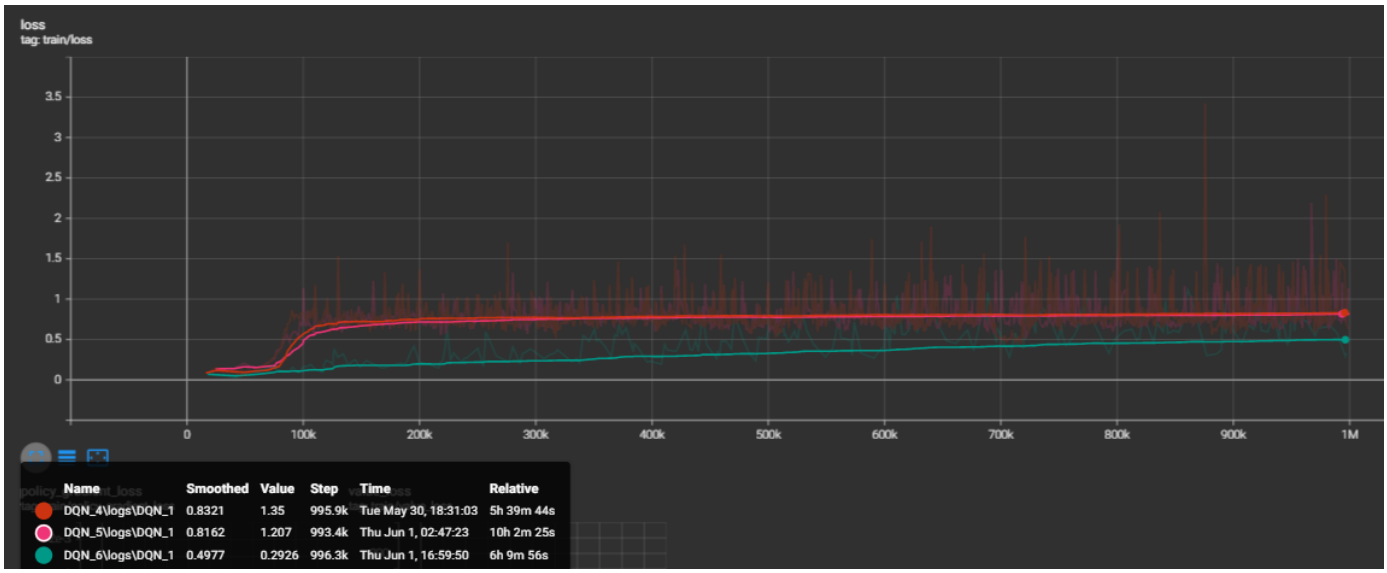

Figure 4 loss vs time step graph for DQNs (smoothing = 1)

Erkan Dogan
2166262
2023-06-11

3.1.5. Plot PPO vs DQN comparison for ep_rew_mean where they are using same preprocessing methods in one figure

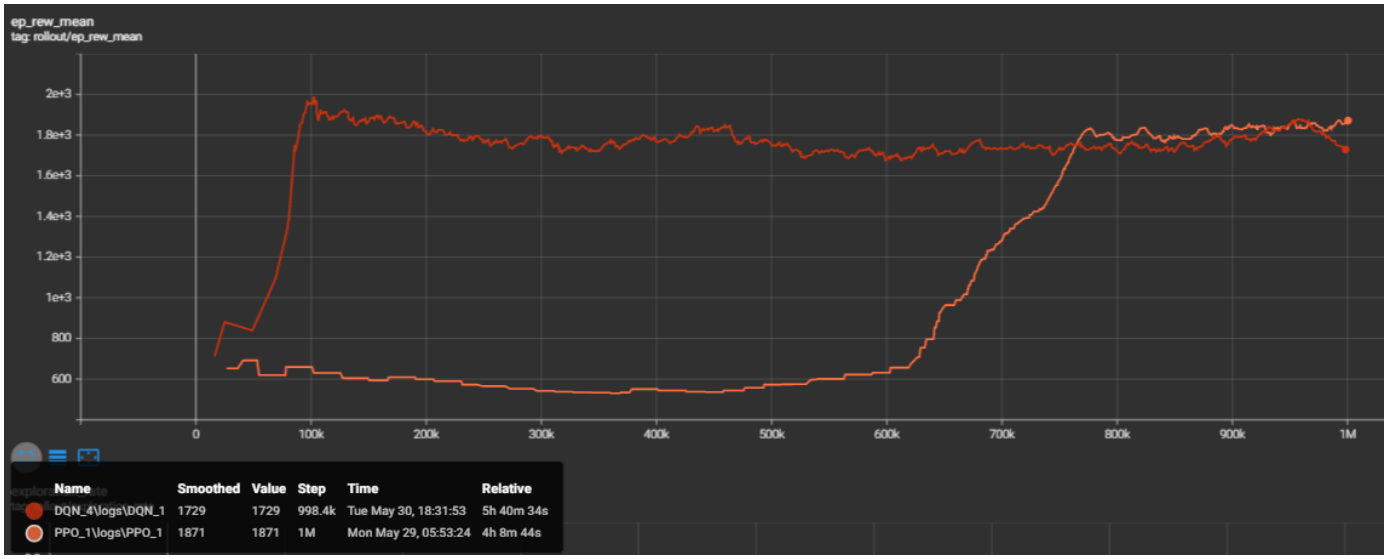

Figure 5 ep_rew_mean vs time step graph for PPO_1 and DQN_4 (smoothing = 0)

3.1.6. Plot PPO vs DQN comparison for loss where they are using same preprocessing methods in one figure

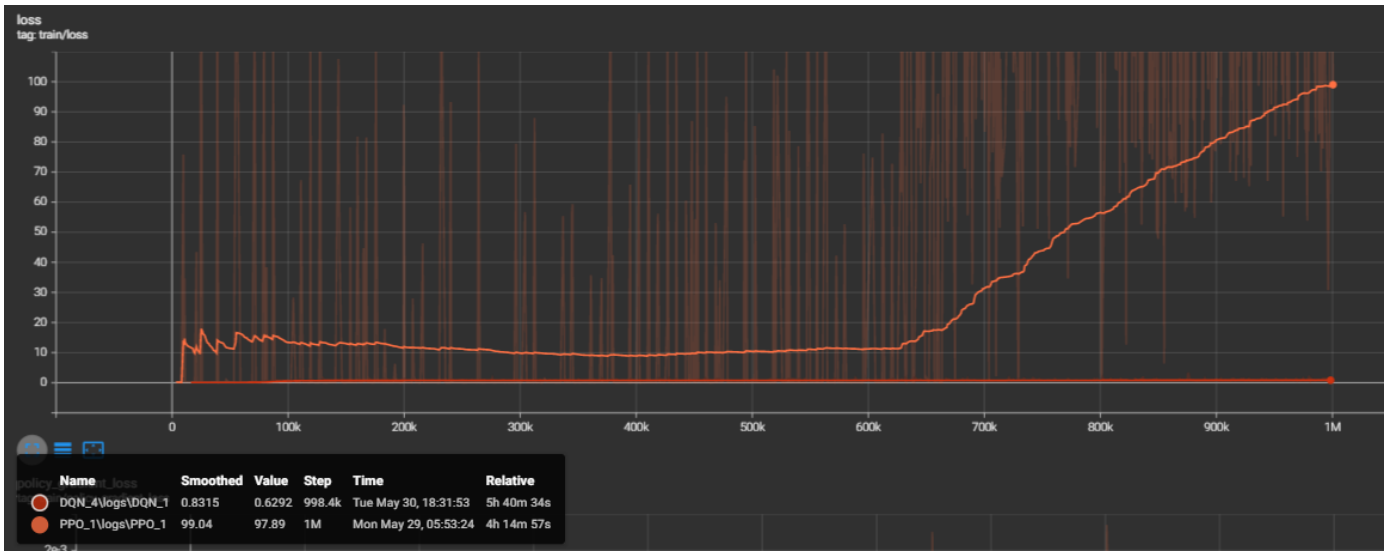

Figure 6 loss vs time step graph for PPO_1 and DQN_4 (smoothing = 1)

3.1.7. Plot best algorithm (DQN_4-> DQN_7) ep_rew_mean

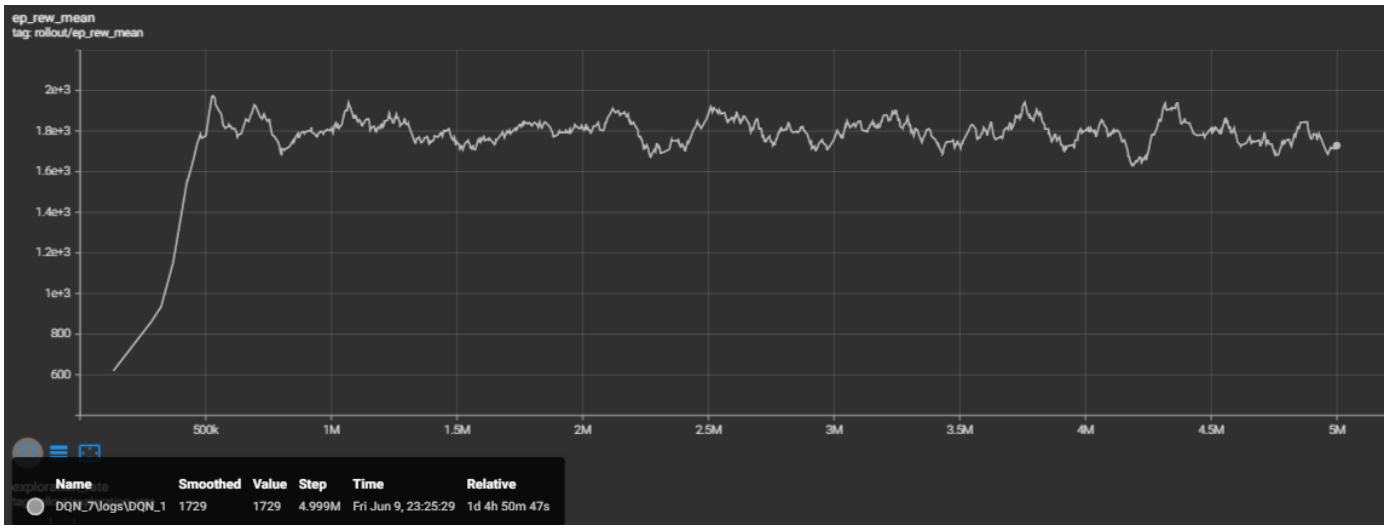

Figure 7 ep_rew_mean vs time step graph for DQN_7 (smoothing = 0)

3.1.8. Plot best algorithm (DQN_4-> DQN_7) loss

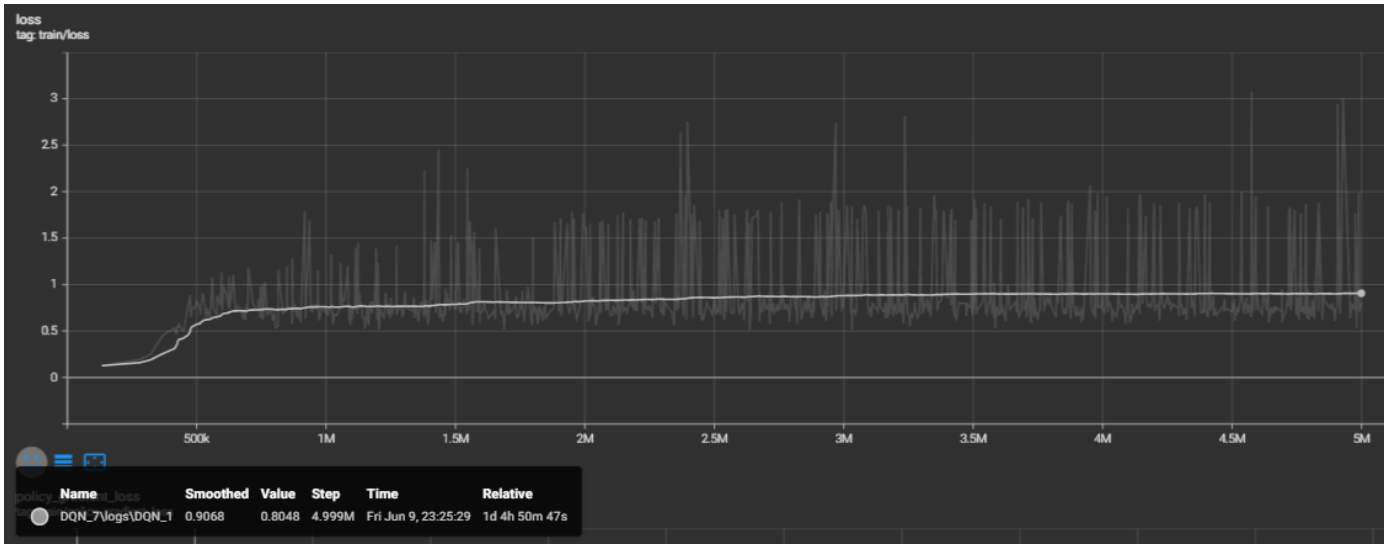

Figure 8 loss vs time step graph for DQN_7 (smoothing = 1)

3.1.9. Generalizability of PPO and DQN models for Random Stages

It was observed that PPO performs better in a random stage because it is policy based meanwhile the DQN uses the action result values in a trained environment.

Erkan Dogan
2166262
2023-06-11

3.2. Discussions

3.2.1. 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?

We can visually track our model's performance for the models that ep_rew_mean increases significantly. Mario jumps over the longer pipes when ep_rew_mean is around 1.5e+3. The highest score was 17200.

3.2.2. 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?

DQN learns faster and more efficiently when it is compared to PPO. However, DQN does not improve itself after the exploration rate decreases to the base level meanwhile PPO starts learning later.

3.2.3. 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?

PPO uses probability distribution by taking actions in the environment and uses these distribution values for exploration. It tries to find a balance between exploration and exploitation by updating its policy by using entropy regularization. With this balance improvement, exploration takes more space and it prevents converging to suboptimal policies.

DQN uses a greedy exploration strategy as a value based algorithm. It selects random actions on the environment with predefined probabilities. Due to its value based approach, it is prone to fall into exploitation with relying more on learned Q-values.

Overall, to balance exploration and exploitation, PPO is better because it tries to balance these two values although DQN does not try.

3.2.4. 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?

PPO shows a better performance than DQN because of it is more policy based instead of being value based. It utilizes sampling the environment and updates its policy accordingly. Due to that, PPO can be generalized to new environments better and it is more robust or adaptable. However, DQN uses action values in different states in a trained environment, which limits its good usage to specific environments.

Erkan Dogan
2166262
2023-06-11

3.2.5. 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?

Hyperparameters are important to reach a well trained result in a specific time and there is a tradeoff between them always.

In PPO, the critical parameters are policy name, learning rate, number of steps. Policy name decides which kind of policy is needed such as CNN or MLP which can be important depending on the environment. Learning rate controls the step size for parameter updates, and it affects the convergence and stability. As learning rate increases, learning gets faster. Number of steps are used for collecting experiences before updating the policy and it is important for deciding sample efficiency and stability. As number of steps increases, learning gets slower. As exploration decreases, performance and learning speed decreases.

In DQN, the critical parameters are policy name, learning rate, exploration parameters, learning start, buffer size, batch size, and train frequency. Learning rate has similar affect on performance as it has on PPO. Exploration parameters controls the relationship between exploration and exploitation, by deciding how long and with which fraction the algorithm should explore new values. Learning start parameter decides until which value the exploration will continue initially. As this parameter increases, initial exploration increases. Buffer size stores the number of experiences and as its value increases, the learning gets slower. Batch size lets agent to update its Q-network more frequently and it keeps more experience values to learn in each update. With higher batch size, the converge and improvement can be faster for performance and learning speed. Train frequency decides how many updates per time step should be done. As the train frequency increases, the learning speed increases.

3.2.6. 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?

In my models' learning process, I observed that DQN takes much longer time to train for same number of timesteps. Since PPO is policy based algorithm, it uses a smaller neural network than DQN and provides faster results. For practicality or scalability, complexity is the most important topic every aspect of technological development. Although DQN provides better results, due to the resources, PPO is mostly considered for model training.

3.2.7. 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?

CNN policy are designed for high dimensional input states, which can be images. CNNs can find patterns and local dependencies in the data easily. They use different layer types such as convolutional layers, pooling layers and non linear activations to create a learning representation from pixels.

MLP is fully connected neural networks where which is better for low-dimensional input states where the spatial structure is not important.

Due to their focus different input states, CNN is better for high-dimensional input states because it uses in-between network connection values with spatial structure and gather the important features better.

Erkan Dogan
2166262
2023-06-11

Appendix A: Code

```
# Import environment libraries
import gym_super_mario_bros
from nes_py.wrappers import JoypadSpace
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT
from utils import *
# Import preprocessing wrappers
from gym.wrappers import GrayScaleObservation
from stable_baselines3.common.vec_env import VecFrameStack, DummyVecEnv, VecMonitor,
SubprocVecEnv
from matplotlib import pyplot as plt

from utils import SaveOnBestTrainingRewardCallback
from stable_baselines3 import PPO
from stable_baselines3 import DQN

from matplotlib import pyplot as plt
import torch

SAVE_FREQ = 10000
CHECK_FREQ = 1000
TOTAL_TIMESTEPS = 5000000
BASE_DIR = "C:/Users/erkan/Desktop/EE/e2022_2/EE449/2023/HW3/Code/"

device_cuda = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"{device_cuda} is available")
print(torch.cuda.is_available())
print(torch.cuda.current_device())
print(torch.cuda.device(0))
print(torch.cuda.device_count())
print(torch.cuda.get_device_name(0))


def path_creator(model_name, itr):
    CHECKPOINT_DIR = BASE_DIR + f'{model_name}_{itr}/' + 'train/'
    LOG_DIR = BASE_DIR + f'{model_name}_{itr}/' + 'logs/'
    # Create the necessary directories
    os.makedirs(CHECKPOINT_DIR, exist_ok=True)
    os.makedirs(LOG_DIR, exist_ok=True)
    return CHECKPOINT_DIR, LOG_DIR

def env_generator(CHECKPOINT_DIR, itr):
    # 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)
```

```
    # Apply the preprocessing
    env = GrayScaleObservation(env, keep_dim=True) # Convert to grayscale to reduce
dimensionality
    env = DummyVecEnv([lambda: env])
    #env = SubprocVecEnv([lambda: env])
    # Alternatively, you may use SubprocVecEnv for multiple CPU processors
    if itr == 3 or itr == 6:
        env = VecFrameStack(env, 5, channels_order='last') # Stack frames
    else:
        env = VecFrameStack(env, 4, channels_order='last') # Stack frames
    env = VecMonitor(env, f"{CHECKPOINT_DIR}TestMonitor") # Monitor your progress
    return env

def trainer(model_name, itr):
    CHECKPOINT_DIR, LOG_DIR = path_creator(model_name, itr)
    env = env_generator(CHECKPOINT_DIR, itr)
    callback_func = SaveOnBestTrainingRewardCallback(save_freq=SAVE_FREQ,
check_freq=CHECK_FREQ, chk_dir=CHECKPOINT_DIR)

    if itr == 1:
        model = PPO('CnnPolicy', env, verbose=1, tensorboard_log=LOG_DIR,
learning_rate=0.000001, n_steps=256, device=device_cuda)
    elif itr == 2:
        model = PPO('CnnPolicy', env, verbose=1, tensorboard_log=LOG_DIR,
learning_rate=0.000001, n_steps=512, device=device_cuda)
    elif itr == 3:
        model = PPO('CnnPolicy', env, verbose=1, tensorboard_log=LOG_DIR,
learning_rate=0.0000001, n_steps=256, device=device_cuda)

    elif itr == 4:
        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,
device=device_cuda)
    elif itr == 5:
        model = DQN('CnnPolicy', env, batch_size=384, 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,
device=device_cuda)
    elif itr == 6:
        model = DQN('CnnPolicy', env, batch_size=192, verbose=1, learning_starts=10000,
learning_rate=5e-4, exploration_fraction=0.1, exploration_initial_eps=1.0,
exploration_final_eps=0.1, train_freq=8, buffer_size=10000, tensorboard_log=LOG_DIR,
device=device_cuda)
    elif itr == 7:
        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,
device=device_cuda)

    else:
        return "Invalid Argument"

    model.learn(total_timesteps=TOTAL_TIMESTEPS, log_interval=1, callback=callback_func)
    model.save(f'{CHECKPOINT_DIR}best_model_{model_name}_{itr}')


if __name__ == "__main__":
    #trainer("PPO", 1)
    #trainer("PPO", 2)
    #trainer("PPO", 3)
    #trainer("DQN", 4)
    #trainer("DQN", 5)
    #trainer("DQN", 6)
    #trainer("DQN", 7)
    pass


#startGameModel(env, model)
for i in range(10):
    env =
env_generator("C:/Users/erkan/Desktop/EE/e2022_2/EE449/2023/HW3/Code/DQN_7/train/", 4)
    model =
DQN.load("C:/Users/erkan/Desktop/EE/e2022_2/EE449/2023/HW3/Code/DQN_7/train/best_model")
    saveGameModel(env = env, len = 5000000, model = model)
```