Introduction

In this homework assignment, we will delve into the exciting field of **Reinforcement Learning** (RL) and its application in training an agent to excel in Atari game environments. Specifically, the focus will be on developing an RL model capable of conquering the first stage of the iconic Super Mario Bros game. To accomplish this, we will utilize various tools and libraries, including **PyTorch**, **OpenAI Gym**, **NES Emulator**, and **Stable-Baselines3**.

To track the progress of our agent and gain insights into its performance, we will leverage TensorBoard, a powerful visualization tool. Throughout this homework, we will explore advanced RL algorithms such as **Proximal Policy Optimization** (PPO) and **Deep Q-Network** (DQN). These algorithms will be trained for an extensive duration of 1 million iterations.

During the training process, we will experiment with different preprocessing methods to enhance the agent's learning capabilities. Furthermore, we will evaluate the performance of the PPO and DQN algorithms by analyzing entropy loss and episode mean reward values. To gain a comprehensive understanding of their performance trends, we will plot these values as functions of training steps.

By undertaking this homework, our objective is to gain valuable insights into the potential of Reinforcement Learning. We aim to observe the impact of different preprocessing methods on the agent's performance and thoroughly evaluate the capabilities of the PPO and DQN algorithms within the Super Mario Bros environment.

1. Basic Questions

Agent:

An autonomous learner or decision-maker that engages with the environment is referred to as an agent in **reinforcement learning**. It keeps track of the environment's state, makes decisions in accordance with its policy, and then gets feedback in the form of incentives or punishments. The agent's objective is to discover an ideal course of action that maximizes its long-term cumulative payoff. Since the learning system is trained using labeled data, there is often no explicit concept of an agent in **supervised learning.** The algorithm instead learns from a dataset with input-output pairs provided by a supervisor rather than actively interacting with an environment. Learning a function that can precisely translate inputs into appropriate outputs is the objective.

Environment:

The external system or context that the agent operates in is represented by the environment in **reinforcement learning**. It includes the world's current state, the game's rules, and the dynamics that control how the state shifts in response to agent activities. The environment influences the agent's learning process by giving the agent feedback in the form of rewards or punishments. In **supervised learning**, the environment is not explicitly considered as the primary focus lies in understanding patterns and relationships within labeled training data. The emphasis is on analyzing input-output pairs present in the dataset, which serve as representative examples of the target function or phenomenon under investigation. Unlike in other learning approaches, the

aspect of interacting with a dynamic environment is not a central aspect in supervised learning.

Reward:

In **reinforcement learning**, an agent receives a reward, which is a scalar value, from the environment as a result of its actions. It acts as a gauge of the behavior of the agent's quality or desirability. The agent's goal is normally to learn a policy that produces bigger rewards in order to maximize the cumulative reward over time. Moreover, the idea of a reward is not frequently used in **supervised learning**. The learning method instead seeks to reduce a predetermined loss function based on the differences between anticipated outputs and ground truth labels. The loss function measures the discrepancy between the expected and observed outcomes and directs learning toward increasing prediction accuracy.

Policy:

In **reinforcement learning,** a policy refers to a mapping that links states to actions, serving as a representation of the agent's behavior or strategy. It outlines how the agent makes decisions by selecting appropriate actions based on the current state it observes. The primary objective of the agent is to acquire an optimal policy that maximizes the anticipated total reward over time.
It is uncommon to employ the idea of a policy in **supervised learning**. Instead, without explicitly taking into account decision-making processes, the emphasis is on learning a mapping from input features to output labels. The learned model tries to produce precise predictions or classifications and to generalize well to unknown inputs.

Exploration:

In **reinforcement learning,** exploration is the agent's habit of actively seeking out and attempting novel behaviors in order to learn more about their surroundings. It aids the agent in learning a more advantageous policy and identifying potential better actions. The agent can better comprehend the dynamics of the environment and enhance its performance over time by investigating various actions. Since the learning algorithm is given labeled data, exploration is not a unique concept in **supervised learning** because active exploration is not required to obtain information. The emphasis is on using the readily available labeled samples to uncover the underlying trends and connections in the data.

Exploitation:

In **reinforcement learning,** the agent's behavior of taking what it now believes to be the best course of action based on its learned policy is referred to as "exploitation." It concentrates on maximizing immediate benefits in light of the agent's present knowledge. The agent seeks to make the best decisions possible and to take advantage of its knowledge base in order to maximize rewards by utilizing the learnt policy. Since the objective of **supervised learning** is to apply the learnt model to provide correct predictions or classifications rather than determining trade-offs between exploration and exploitation, exploitation is not a distinct concept in this process. The main goal is to use the learnt model to forecast additional, unforeseen inputs.

2. Experimental Work

Trained Models:

1. PPO_1

    - Preprocessing Method:
        - **keep_dim = True**, in GrayScaleObservation
        - **n_stack = 4**, in VecFrameStack

    - Model parameters:
        - Algorithm: PPO
        - Policy: 'CnnPolicy'
        - Environment: env
        - Verbosity (verbose): 1
        - Tensorboard Log Directory (tensorboard_log): LOG_DIR
        - Learning Rate (learning_rate): 0.000001
        - Number of Steps (n_steps): 512

2. PPO_2

    - Preprocessing Method:
        - **keep_dim = True**, in GrayScaleObservation
        - **n_stack = 4**, in VecFrameStack

    - Model parameters:
        - Algorithm: PPO
        - Policy: 'CnnPolicy'
        - Environment: env
        - Verbosity (verbose): 1
        - Tensorboard Log Directory (tensorboard_log): LOG_DIR
        - Learning Rate (learning_rate): 0.00001
        - Number of Steps (n_steps): 512

3. PPO_3

    - Preprocessing Method:
        - **keep_dim = False**, in GrayScaleObservation
        - **n_stack = 3**, in VecFrameStack

    - Model parameters:
        - Algorithm: PPO
        - Policy: 'MlpPolicy'
        - Environment: env
        - Verbosity (verbose): 1
        - Tensorboard Log Directory (tensorboard_log): LOG_DIR
        - Learning Rate (learning_rate): 0.0001
        - Number of Steps (n_steps): 512
        - Entropy Coefficient (ent_coef): 0.01
        - Value Function Coefficient (vf_coef): 0.5

4. DQN_1:

    - Preprocessing Method:
        - **keep_dim = True**, in GrayScaleObservation
        - **n_stack = 4**, in VecFrameStack

    - Model Parameters:
        - Algorithm: DQN
        - Policy: 'CnnPolicy'
        - Environment: env
        - Batch Size: 192
        - Verbosity (verbose): 1
        - Learning Starts: 10000
        - Learning Rate (learning_rate): 5e-3
        - Exploration Fraction: 0.1
        - Exploration Initial Epsilon: 1.0
        - Exploration Final Epsilon: 0.1
        - Train Frequency: 8
        - Buffer Size: 10000
        - Tensorboard Log Directory (tensorboard_log): LOG_DIR

5. DQN_2:

    - Preprocessing Method:
        - **keep_dim = False**, in GrayScaleObservation
        - **n_stack = 2**, in VecFrameStack

    - Model Parameters:
        - Algorithm: DQN
        - Policy: 'MlpPolicy'
        - Environment: env
        - Batch Size: 192
        - Verbosity (verbose): 1
        - Learning Starts: 10000
        - Learning Rate (learning_rate): 5e-3
        - Exploration Fraction: 0.1
        - Exploration Initial Epsilon: 1.0
        - Exploration Final Epsilon: 0.1
        - Train Frequency: 8
        - Buffer Size: 10000
        - Tensorboard Log Directory (tensorboard_log): LOG_DIR

6. DQN_3:

    - Preprocessing Method:
        - **keep_dim = True**, in GrayScaleObservation
        - **n_stack = 4**, in VecFrameStack

    - Model Parameters:

- Algorithm: DQN
- Policy: 'CnnPolicy'
- Environment: env
- Batch Size: 256
- Verbosity (verbose): 1
- Learning Starts: 10000
- Learning Rate (learning_rate): 1e-4
- Exploration Fraction: 0.2
- Exploration Initial Epsilon: 1.0
- Exploration Final Epsilon: 0.05
- Train Frequency: 16
- Buffer Size: 50000
- Tensorboard Log Directory (tensorboard_log): LOG_DIR

3. Benchmarking and Discussions

3.1. Benchmark

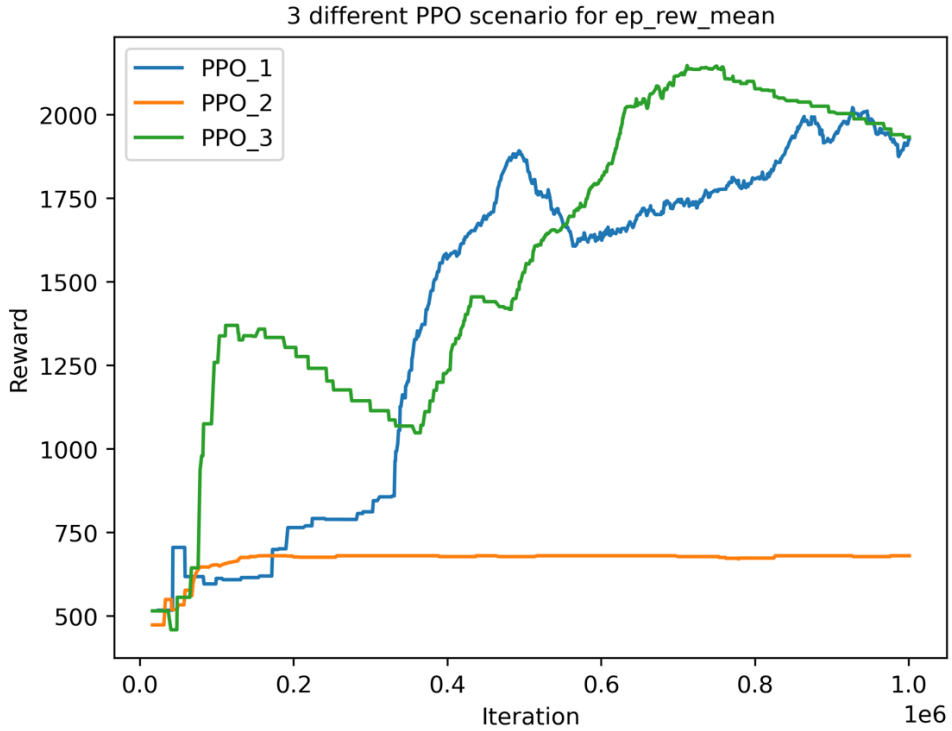

FIGURE 1. THREE DIFFERENT PPO SCENARIO FOR EPISODE MEAN REWARD.

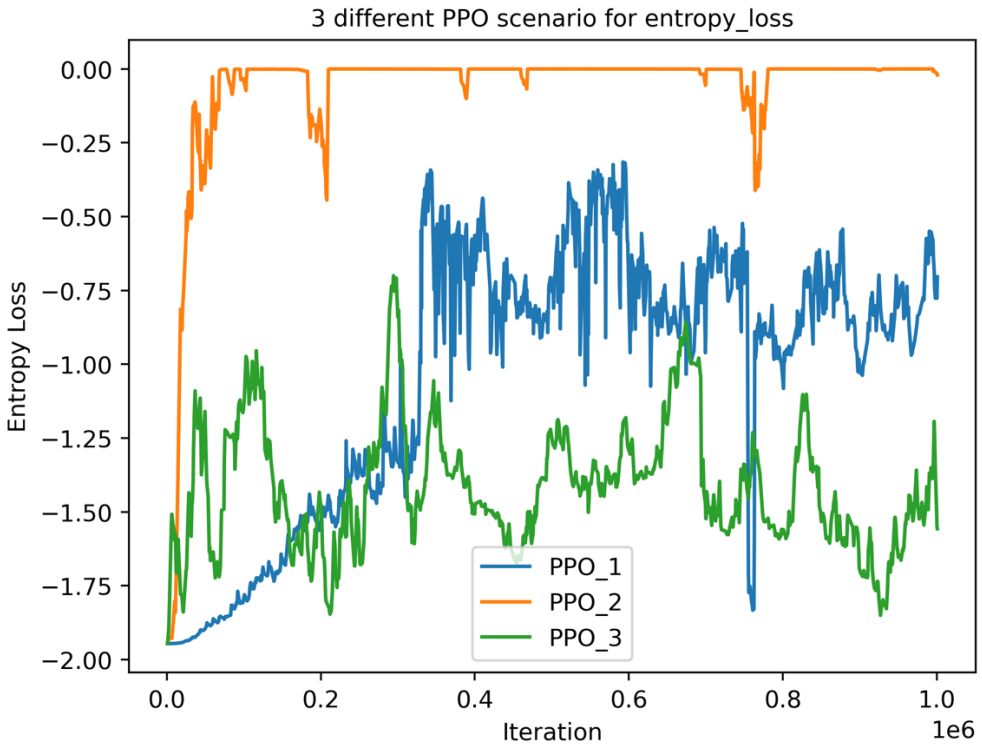

FIGURE 2. THREE DIFFERENT PPO SCENARIO FOR ENTROPY LOSS.

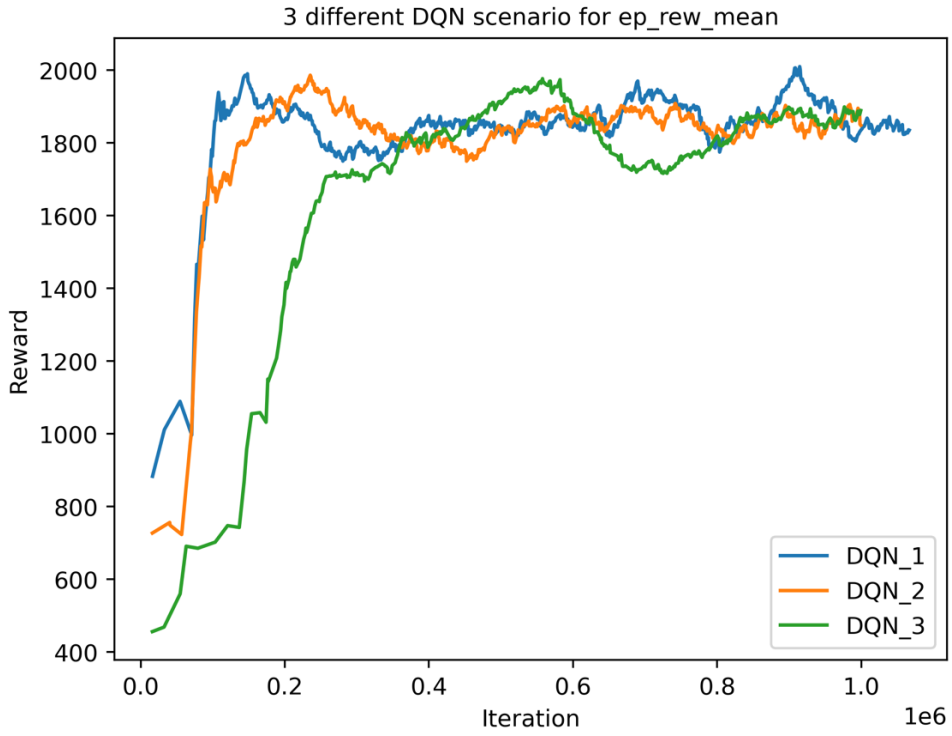

FIGURE 3. THREE DIFFERENT DQN SCENARIO FOR EPISODE MEAN REWARD.

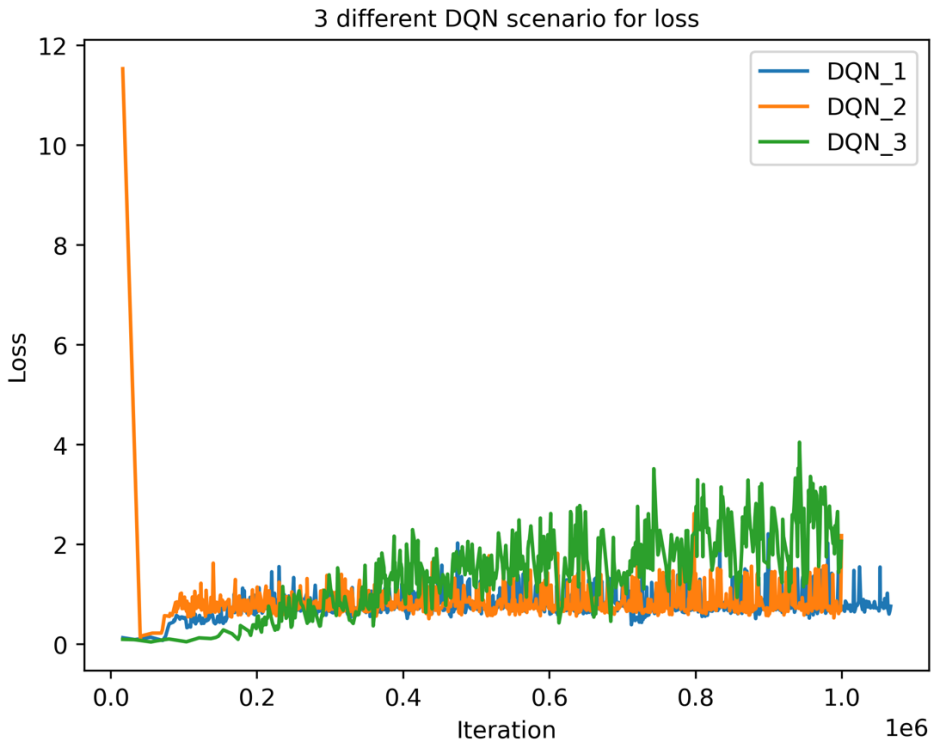

FIGURE 4. THREE DIFFERENT DQN SCENARIO FOR TRAIN LOSS.

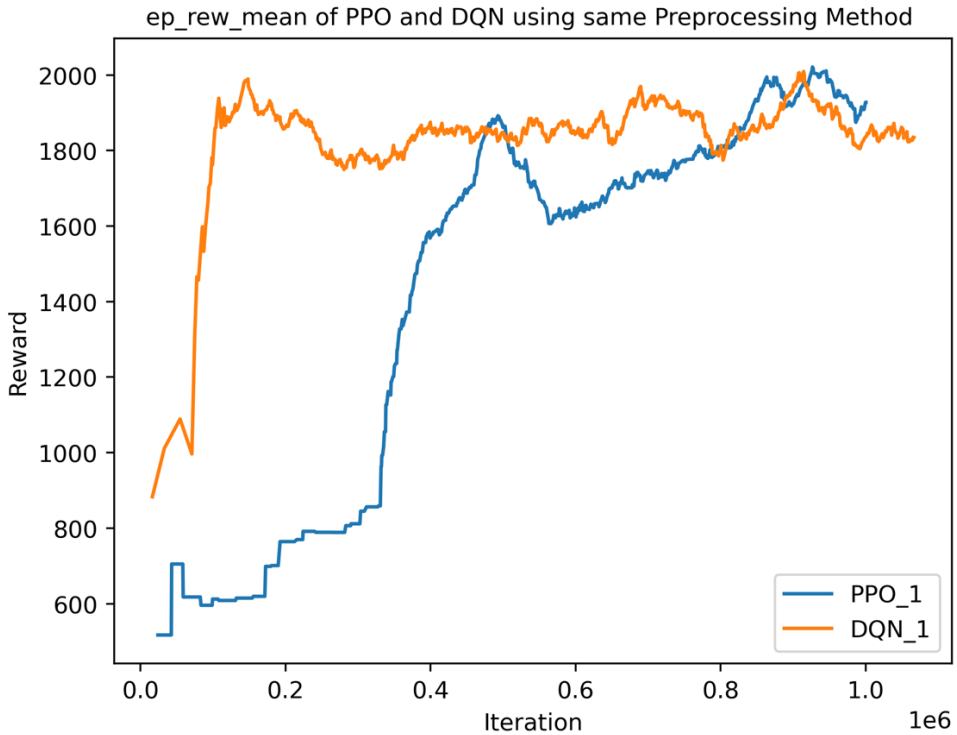

FIGURE 5. EPISODE MEAN REWARD OF PPO_1 AND DQN_1 USING SAME PREPROCESSING METHOD.

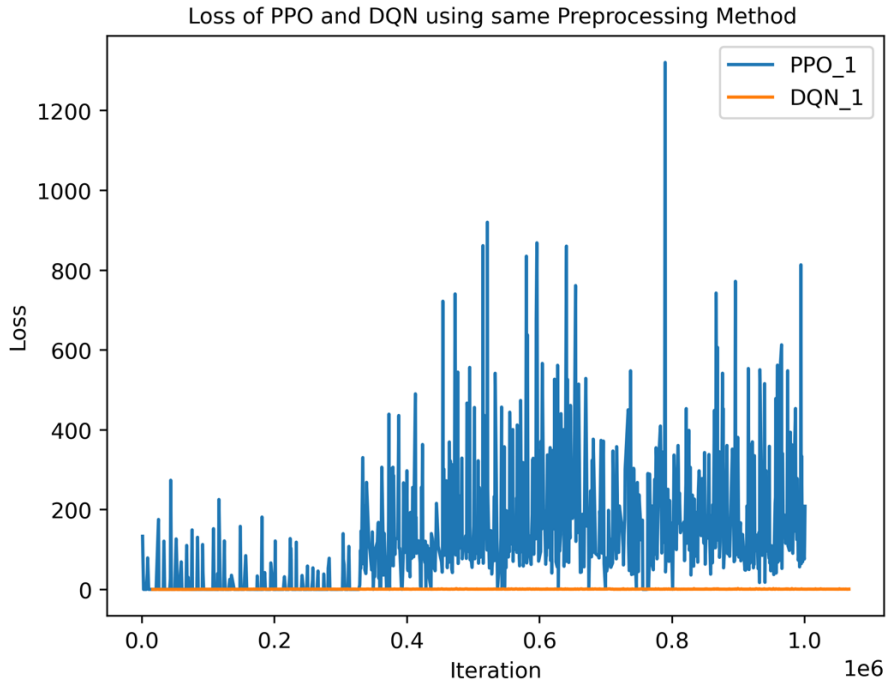

FIGURE 6. TRAIN LOSS OF PPO_1 AND DQN_1 USING SAME PREPROCESSING METHOD.

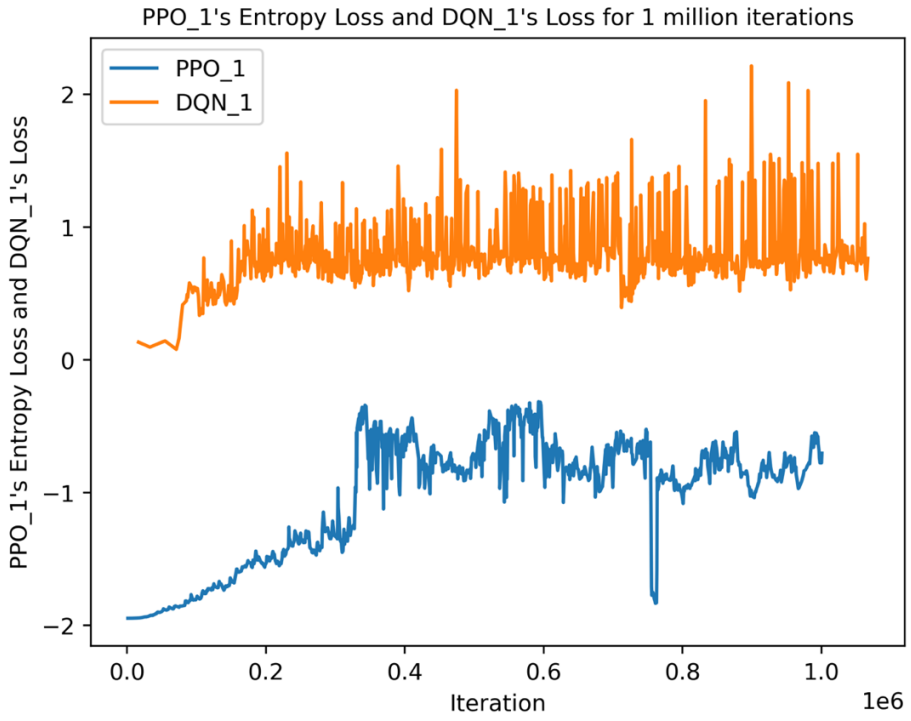

FIGURE 7. ENTROPY LOSS OF PPO_1 AND TRAIN LOSS OF DQN_1 USING SAME PREPROCESSING METHOD.

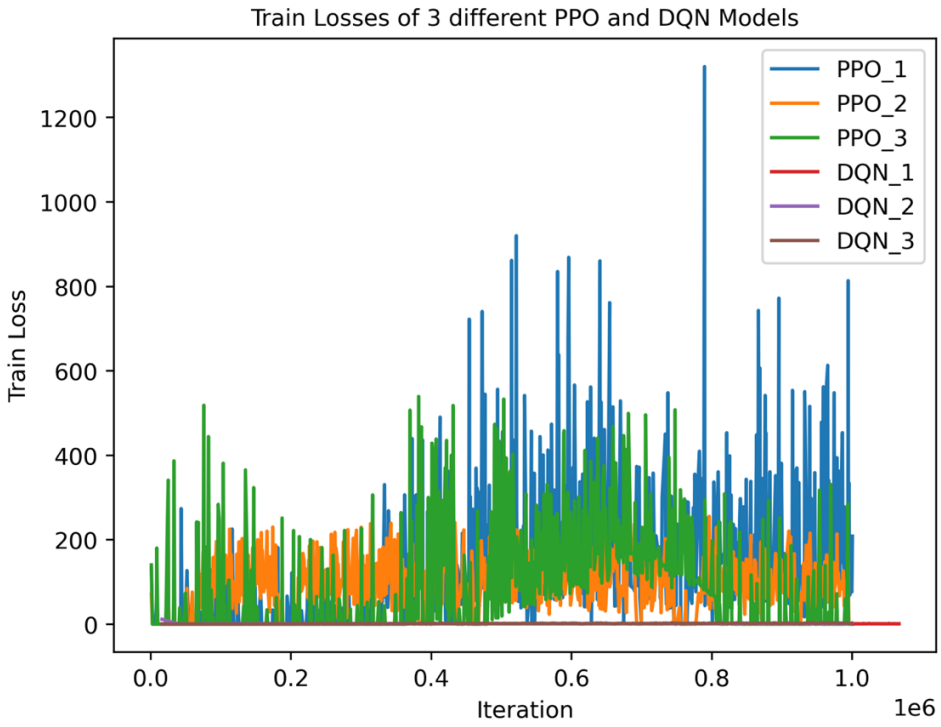

FIGURE 8. TRAIN LOSSES OF EACH TRAINED MODEL.

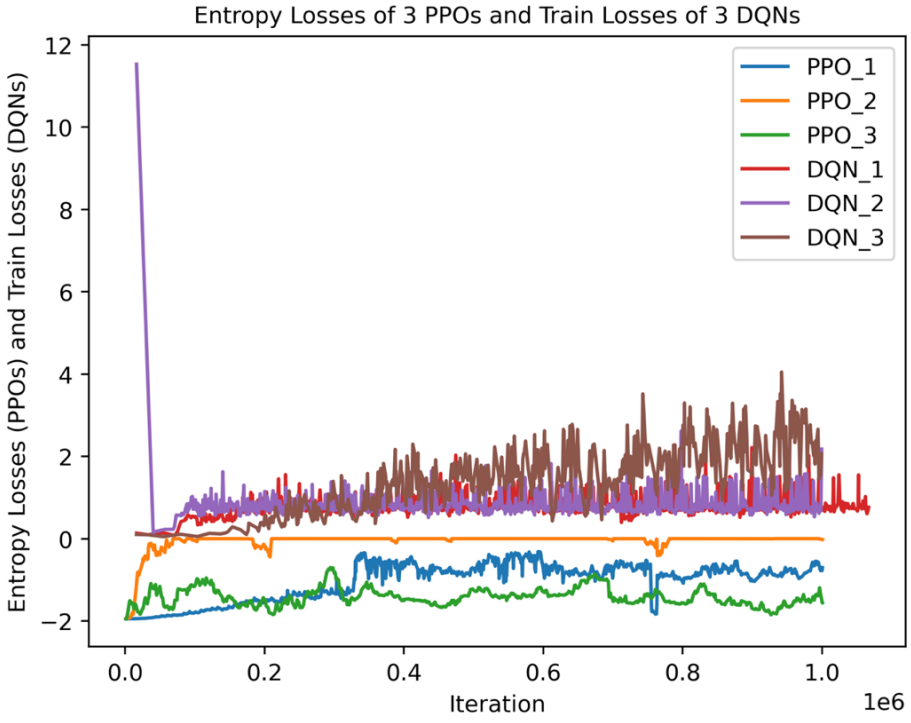

FIGURE 9. ENTROPY LOSSES OF PPO ALGORITHMS AND TRAIN LOSSES OF DQN ALGORITHMS.

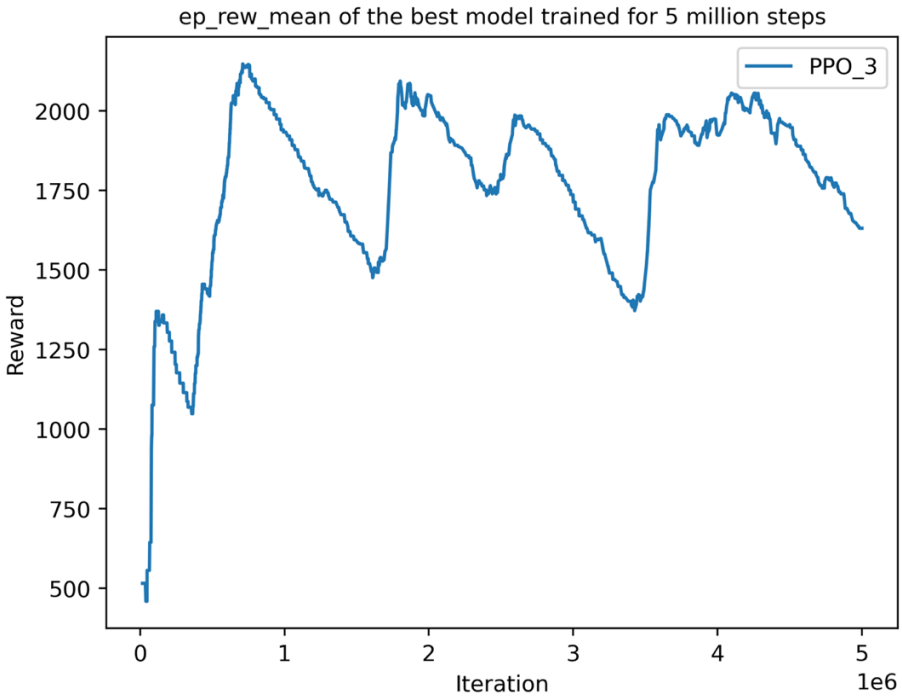

FIGURE 10. EPISODE MEAN REWARD OF THE BEST MODEL TRAINED FOR 5 MILLION ITERATIONS.

YouTube link for the best episode (episode at which the highest score is obtained), using PPO_3 model, utilizing MLPPolicy: https://www.youtube.com/watch?v=eub-oTVlWP8

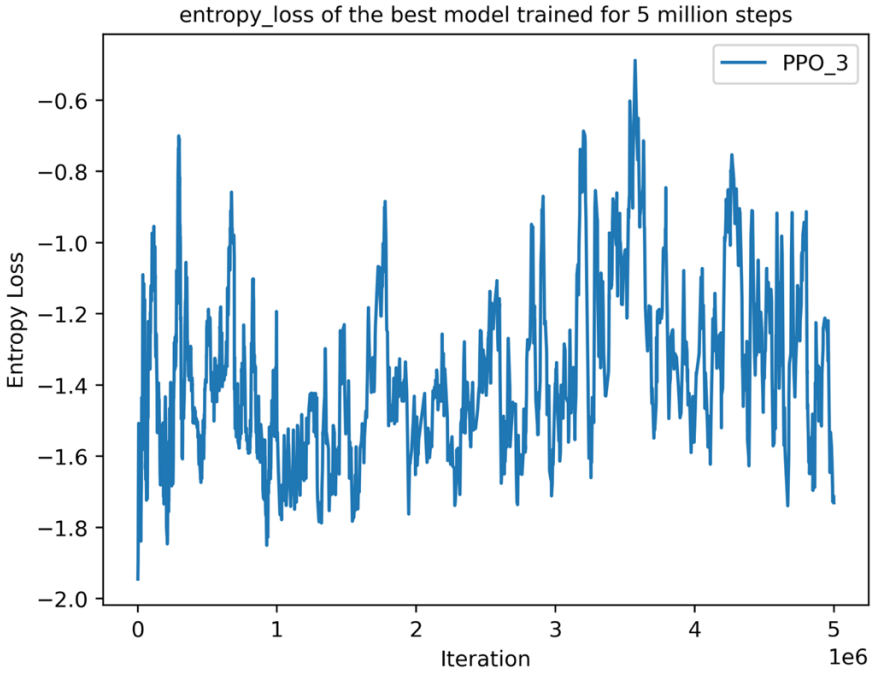

FIGURE 11. ENTROPY LOSS OF THE BEST MODEL TRAINED FOR 5 MILLION ITERATIONS.

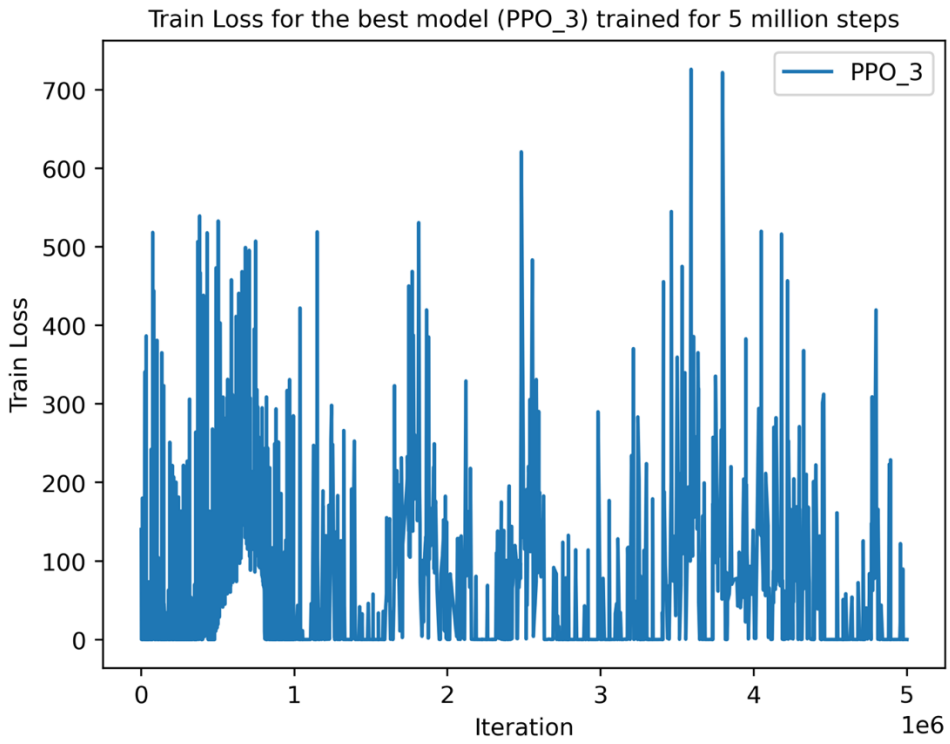

FIGURE 12. TRAIN LOSS FOR THE BEST MODEL (PPO_3) TRAINED FOR 5 MILLION ITERATIONS.

3.2. Discussions

1. During the earlier stages of training (0-100,000 iterations), Mario's performance is limited, and he struggles to progress far in the game. This can be attributed to the initial lack of knowledge and exploration by the agent, which hinders its ability to navigate through the game environment effectively.

   However, as the training progresses and more iterations are completed, Mario's performance gradually improves. After 500,000 iterations, a noticeable advancement is observed, with Mario becoming capable of successfully jumping through long pipes. This signifies that the agent has started to learn and understand the mechanics of the game better.

   The learning process continues beyond the initial stages, and the performance of most models tends to increase over time. This suggests that the agents are continuously acquiring knowledge and improving their gameplay strategies. It is important to note that the performance improvement may vary across different models and training configurations.

   In this particular case, the model labeled as PPO_3 stands out as the best-performing model. It was trained for 5 million steps, and after only 6 episodes, Mario was able to complete the stage, achieving a score of 14550. This accomplishment indicates that

the agent has acquired sufficient skills and knowledge to navigate the environment successfully and reach new stages.

The observations made in this training process highlight the iterative nature of reinforcement learning, where the agent gradually learns and adapts to the environment over time. As more training steps are performed, the agent's performance improves, enabling it to overcome challenges and progress further in the game.

Table 1: Scores at different iterations within the training process.

Model \ Iteration	Highest score at iteration 0	Highest score at iteration 10,000	Highest score at iteration 100,000	Highest score at iteration 500,000	Highest score at iteration 1,000,000
PPO 1	0	0	200	600	800
PPO 2	0	0	100	200	300
PPO 3	0	100	200	1000	1100
DQN 1	0	100	300	500	800
DQN 2	0	0	200	500	700
DQN 3	0	100	200	400	600

2. Which algorithm learns more quickly or effectively can be determined by contrasting the PPO and DQN learning curves in terms of the average episode reward over time. In general, a steeper learning curve denotes quicker learning, whereas a larger average episode reward denotes learning that is more effective. One can see how both algorithms are learning by charting the average episode reward over time for each. It can be said that an algorithm learns more quickly or effectively if it has a greater and steeper learning curve and achieves a higher average episode reward in fewer iterations.

   Inspecting *Figure 5*, episode mean reward of the model with DQN algorithm rises more rapidly than the model with PPO algorithm, using the same preprocessing method, in the earlier stages of the learning process. However, DQN algorithm quickly converges to a value which is not always desired. Learning curve (episode mean reward) of the PPO algorithm gradually increases, and even passes the best value of the DQN algorithm, after some iterations.

3. To avoid sudden policy updates, PPO employs a policy optimization strategy that includes a clipping mechanism. This aids in balancing exploration and exploitation, regulating the learning process. PPO concentrates more on fine-tuning the policy by revising it gradually to guarantee a smooth transition.

   The Q-values for state-action pairs are learned by DQN using a value-based method. It uses an epsilon-greedy exploration approach and keeps a replay buffer to sample experiences. Epsilon-greedy with increasing epsilon, DQN first prefers to explore more and gradually transitions to exploitation as it learns better Q-value estimates.

It is more reasonable to make a comparison between the PPO and DQN algorithms when they had been gone through the same preprocessing method. Hence, plot from the *Figure 5*, can be investigated. Since Episode Mean Reward of PPO algorithm has a slowly increasing trend, it can be stated that PPO has a more balanced aspect regarding exploration-exploitation.

4. The PPO algorithm tends to be more reliable and elastic than the DQN algorithm in terms of adaptability to new environments or levels of the game.

   Due to its policy optimization technique, which immediately adjusts the policy depending on the interactions with the present environment, PPO demonstrates greater generalization capabilities. As a result, PPO is better able to adapt to and absorb information from different situations. Additionally, PPO can efficiently handle variable degrees of difficulty and unseen game levels owing to its usage of a value function and policy modifications in a single phase.

   PPO performs better overall in terms of robustness and adaptability when it comes to generalizing to fresh surroundings or undiscovered game levels. It is a better option for applications where adaptability and generalization are essential because it can directly optimize the strategy and manage various difficulty levels. This is proved and observed as the best model (PPO_3 utilizing MLPPolicy) runs better through random stages via gym_super_mario_bros.make('SuperMarioBrosRandomStages-v0'), than the best DQN model.

5. The step size during policy updates is determined by the learning rate. In order to balance the trade-off between stability and convergence speed, an appropriate learning rate must be set. While a lower learning rate could result in slower learning, a higher learning rate could cause unstable updates. Only difference between the models PPO_1 and PPO_2 is the learning rate value. While both models have the same preprocessing method, PPO_2 has a learning rate ten times greater the learning rate of PPO_1. As it can be seen from *Figure 1,* Episode Mean Reward value is dramatically decreased, and the Entropy Loss increased. In this case, increasing the learning rate has a harmful impact on the performance of the model.

   Investigating the plots from *Figures 3 and 4,* the episode mean reward and train loss in the DQN method are significantly influenced by the exploration parameters and buffer size. When compared to DQN_3, DQN_1 outperforms it in the comparison. The lower exploration fraction (0.1) in DQN_1 denotes a greater emphasis on exploitation, hastening the learning of the best strategies. DQN_1 performs better because the exploration initial and final epsilon (1.0 and 0.1, respectively) strike a balance between exploration and exploitation.

   DQN_1 has a buffer size of 10,000, while DQN_3 has a buffer size of 50,000. A reduced buffer size in DQN_1 would exp
   ose the agent to a lesser variety of experiences, which might make learning more difficult. In contrast, DQN_3's higher buffer size enables a wider range of experiences, which can improve performance.

   The performance of DQN_1 is improved by its reduced buffer size, balanced exploration epsilon values, and lower exploration fraction. For the DQN algorithm to perform optimally, it is essential to maintain a balance between exploration and

exploitation, as well as to use a buffer that is the right size. The effects of these settings can vary based on the environment and work at hand, so it's vital to keep this in mind. Additional testing and parameter adjustment may be necessary to determine the ideal setup.

6. When compared to certain other policy gradient approaches, PPO is computationally efficient. It performs numerous epochs of updates on the acquired data and repeatedly modifies the policy settings. The number of policy updates, the size of the neural network, and the quantity of samples taken all affect how complex PPO is.

   DQN, on the other hand, might require greater computational power. To approximate action-value functions, deep neural networks and Q-learning are combined. DQN uses forward-pass computations, backward-pass computations for gradient updates, and replay memory management as its computational stages. The quantity of actions, the size of the network, the size of the replay memory, and the number of iterations all have an impact on how complex DQN is.

   Regarding the training processes in the experimental work, iterations took longer to complete for the models utilized the DQN Algorithm. Moreover, Google Colab collapsed many times during the training of these models, due to the GPU limitations of Colab and sudden internet disconnections. Greater Episode Mean Reward values and shorter training sessions, make the PPO algorithm more suitable for the task given in this homework.

7. In general, reinforcement learning tasks that require managing high-dimensional input states like images are thought to be more suitable for CNNPolicy. It is ideally suited for visual applications due to its capacity to collect spatial information, parameter efficiency, and translation invariance. It's crucial to remember that the final decision on the policy is determined by the particular problem, dataset, and other elements. Observing the results obtained from the experimental work of this homework, PPO algorithm utilizing MLPPolicy yields a slightly better performance than the other models. Moreover, changing the policy from CNNPolicy to MLPPolicy has a minor impact on the models utilizing DQN algorithm.

Conclusion

In conclusion, training an agent to play the Super Mario Bros. video game has shown potential when using reinforcement learning algorithms, particularly PPO and DQN. We observed that the agents' performance increased over time through the iterative training process, enabling them to get past challenges and gain better scores.

The episode mean reward and training loss were significantly influenced by the hyperparameter choices made, including the exploration parameters and buffer size. It is possible to improve agent performance and learning effectiveness by modifying these settings.

PPO algorithms demonstrated superior exploration and exploitation balance as compared to DQN algorithms. DQN, on the other hand, demonstrated quicker learning in some circumstances.

The algorithms' capacity to generalize to fresh settings or previously undiscovered game levels also differed. Both PPO and DQN revealed the ability to adapt to new phases, although PPO was more robust and adaptable while exploring new surroundings.

Overall, this study underscores the potential of reinforcement learning in training intelligent agents for complex tasks like game playing. Further exploration of advanced algorithms and hyperparameter tuning could lead to even more impressive results and performance improvements in the future.

APPENDIX

YouTube link for the best episode (episode at which the highest score is obtained), using PPO_3 model, utilizing MLPPolicy: https://www.youtube.com/watch?v=eub-oTVlWP8

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# Install the necessary libraries
!pip install wheel==0.38.4
!pip install stable-baselines3[extra]
!pip install torch==1.10.1+cu113 torchvision==0.11.2+cu113 torchaudio===0.10.1+cu113 -
f https://download.pytorch.org/whl/cu113/torch_stable.html
!pip install gym_super_mario_bros==7.3.0 nes_py
!pip install tensorboard
```

```
# Import the game
import gym_super_mario_bros
# Import the Joypad wrapper
from nes_py.wrappers import JoypadSpace
# Import the SIMPLIFIED controls
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT

# Import preprocessing wrappers
# Framestack allows us to stack multiple frames together
# GrayScaleObservation converts the frames to grayscale
# Performance is improved by reducing the size of the frames
from gym.wrappers import GrayScaleObservation, FrameStack
# Import the vectorized environment
from stable_baselines3.common.vec_env import VecFrameStack, DummyVecEnv, VecMonitor
from matplotlib import pyplot as plt
# Import PPO for training
```

```
from stable_baselines3 import PPO
# Import PPO for training
from stable_baselines3 import DQN
```

Start the environment

```
# Generate the game environment
env = gym_super_mario_bros.make('SuperMarioBros-v0')

# Limit the action space to the SIMPLE_MOVEMENT
env = JoypadSpace(env, SIMPLE_MOVEMENT)

# Note: If some error occurs in training, run this cell again.
# Error is caused by dimensions of the observation space.
```

```
# Actions Mario can do
SIMPLE_MOVEMENT
# Number of button combinations
```

```
# Shape of the observation space
env.observation_space.shape
```

```
# Number of button combinations
env.action_space
```

```
# Do a random action
SIMPLE_MOVEMENT[env.action_space.sample()]
```

```
# Properties of the frame that the game will be displayed on
env.observation_space
```

Preprocessing

```
# Grayscale reduces dimensionality
# and improve performance (fewer data to process)
env = GrayScaleObservation(env, keep_dim=True)
# Note: If 'MLPPolicy' is used, then keep_dim=False

# Vectorize the environment
env = DummyVecEnv([lambda: env])

# Stack frames
env = VecFrameStack(env, 4)
# Monitor your progress
env = VecMonitor(env, "/content/drive/MyDrive/Colab
Notebooks/MarioData/PPO_altered_env/train/")

# Set the directories
```

```
CHECKPOINT_DIR = "/content/drive/MyDrive/Colab
Notebooks/MarioData/PPO_altered_env/train/"
LOG_DIR = "/content/drive/MyDrive/Colab Notebooks/MarioData/PPO_altered_env/logs/"

# Create the callback: check every 1000 steps
callback = SaveOnBestTrainingRewardCallback(save_freq=100000, check_freq=1000,
chk_dir=CHECKPOINT_DIR)
```

Training

```
# PPO Training

# Create the model
model = PPO('CnnPolicy', env, verbose=1, tensorboard_log=LOG_DIR,
learning_rate=0.0001, n_steps=512, ent_coef=0.01, vf_coef=0.5)
# Start the training
model.learn(total_timesteps=1000000, log_interval=1, callback=callback)
```

```
# DQN Training

# Create the model
model = DQN('MlpPolicy', 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 )
# Start the training
model.learn(total_timesteps=1000000, log_interval=1, callback=callback)
```

```
# Load the model
model = PPO.load(CHECKPOINT_DIR + "best_model", env=env)
```

Test the model

```
# Test the model
state = env.reset()

while True:
    action, _states = model.predict(state)
    state, reward, done, info = env.step(action)
    env.render()
```

Resume training

```
# Resume a training
```

```
# Load the most recent zip file
model_dqn = DQN.load("/content/drive/.shortcut-targets-by-id/1-3c5hKVTtHh3lW01y3paib-wraxEX6qO/DQN_default/train/models/iter_700000", tensorboard_log=LOG_DIR)
# Set the environment
model_dqn.set_env(env)
# Resume the training
model_dqn.learn(total_timesteps=1000001, callback=callback, log_interval=1, tb_log_name="DQN_1", reset_num_timesteps=False)

# Remark:
# As the training continues, naming of the zip files may cause error, overwriting the previous written files
```

Plot using TensorBoard

```
%cd drive/MyDrive/DQN_default
```

```
%load_ext tensorboard
```

```
%ls
```

```
# There must be a file named "logs" within the current directory
# that has the log files in it
%tensorboard --logdir logs

# Another method:
# 1. Download the log files to your local device
# 2. Open terminal
# 3. Activate an environment
# 4. Type " %tensorboard --logdir logs "
# 5. Copy the link to your browser to see the plots
```