EE 449 Homework 3

Reinforcement Learning

Mehmet Arslan

2374411

1.1) Basic Questions

Agent:
Reinforcement Learning: An agent is an automated system which tries to learn best behaviour by interacting with environment. Its decision affected by policy it uses and its current state to reach goal.

Supervised Learning: Similar to a model that undergoes training using labeled data, aiming to establish a mapping between unseen inputs and their corresponding outputs.

Environment:
Reinforcement Learning: Everything that the agent interacts with. Provides feedback to the agent in terms of reward according to its taken actions.

Supervised Learning: In supervised learning, the model learns from labeled datasets rather than directly from its surroundings. However, if we explore the concept of "learning from somewhere," we can draw a parallel between that datasets in SL can be liken to environment in RL.

Reward:
Reinforcement Learning: Feedback signal which is provided to the agent by the environment after taking an action. Agent tries to maximize rewards sum by learning policy.

Supervised Learning: In contrast to the concept of reward in reinforcement learning (RL), the equivalent signal in supervised learning is the loss or error signal. This signal is computed as the difference between the model's prediction and the correct value. Unlike RL, where the goal is to maximize the reward, in supervised learning, the objective is to minimize the loss function by adjusting the model's parameters.

Policy:
Reinforcement Learning: A policy is a set of rules followed by the agent to choose next action.

Supervised Learning: There's no exact equivalent. You may liken it to supervised learning algorithms.

Exploration:
Reinforcement Learning: The process involves the agent actively searches for new states and actions in order to expand its knowledge and gather additional information about the environment.

Supervised Learning: There's no exact equivalent. Cross validation or utilization of distinct data to train model may be represent exploration.

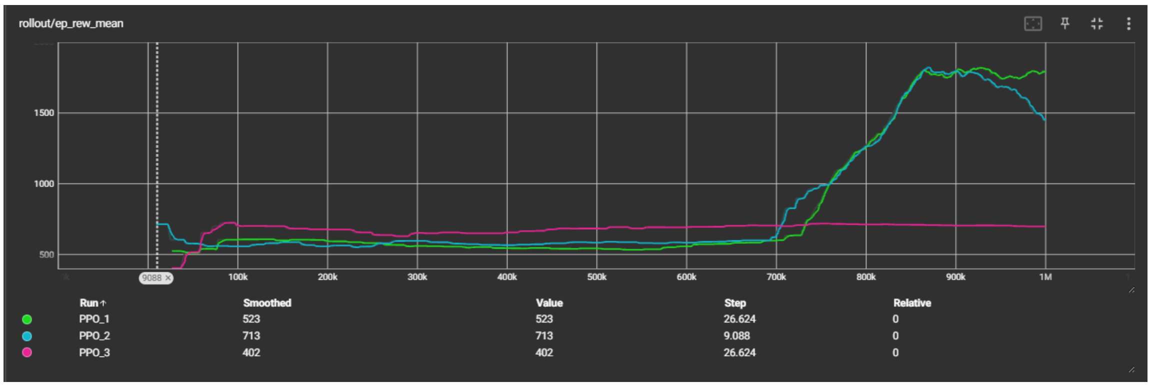
Exploitation:
Reinforcement Learning: The process includes the agent selecting the best action from its existing knowledge in order to achieve the highest possible reward.

Supervised Learning: There's no exact equivalent. A trained model can be viewed in the context of supervised learning, where it takes the form of mapping functions. When a model learns these mappings, it leverages its most up-to-date knowledge to handle unseen data, which may be represent form of exploitation in the RL.

3.1) Benchmarking

In PPO1, the default parameters remained unchanged.
In PPO2, the "**n_steps=512**" parameter was modified to "**n_steps=128**".
In PPO3, the "**CnnPolicy**" parameter was modified to "**MlpPolicy**".

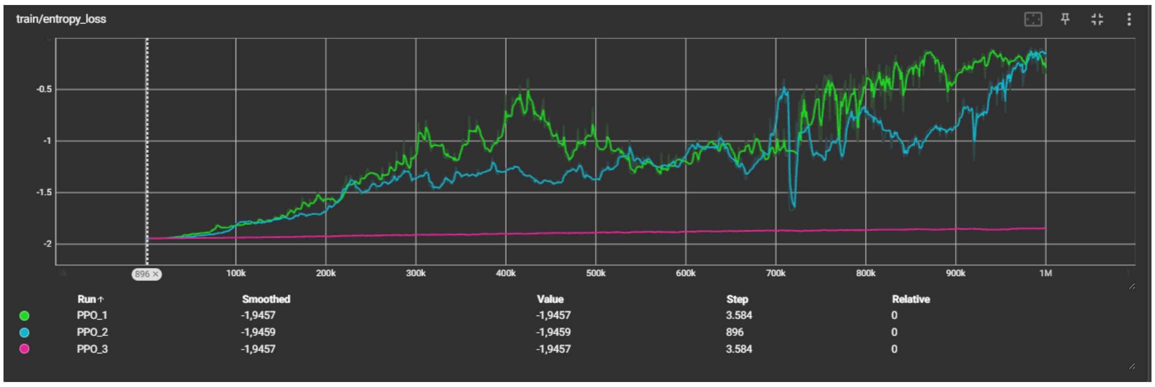
3.1.a) The evaluation of the "**ep_rew_mean**" value was performed in three distinct PPO scenarios.



	Run ↑	Smoothed	Value	Step	Relative
●	PPO_1	523	523	26.624	0
●	PPO_2	713	713	9.088	0
●	PPO_3	402	402	26.624	0

Figure 1: ep_rew_mean curves in three distinct PPO scenarios

3.1.b) The evaluation of the "**entropy_loss**" value was performed in three distinct PPO scenarios.

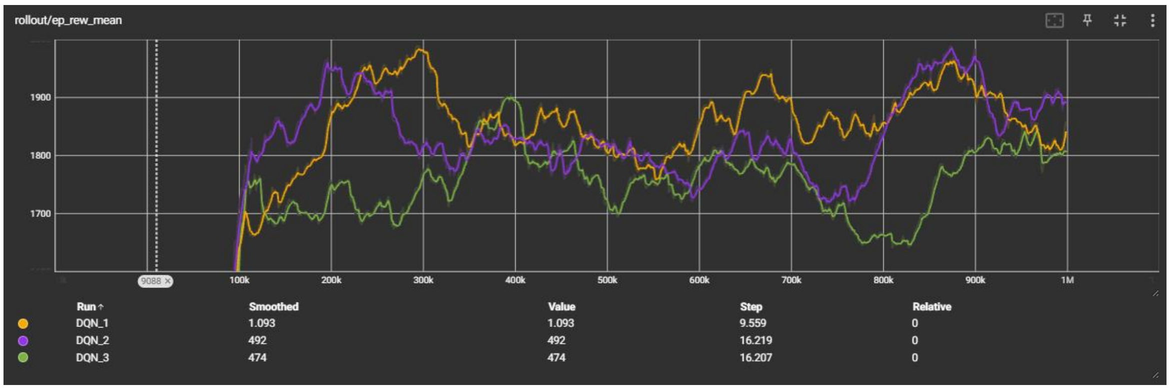


	Run ↑	Smoothed	Value	Step	Relative
●	PPO_1	-1,9457	-1,9457	3.584	0
●	PPO_2	-1,9459	-1,9459	896	0
●	PPO_3	-1,9457	-1,9457	3.584	0

Figure 2: entropy_loss curves in three distinct PPO scenarios

In DQN1, the default parameters remained unchanged.
In DQN2, the "**train_freq=8**" parameter was modified to "**train_freq=32**".
In DQN3, the "**learning_rate=5e-3**" parameter was modified to "**learning_rate=2e-1**".

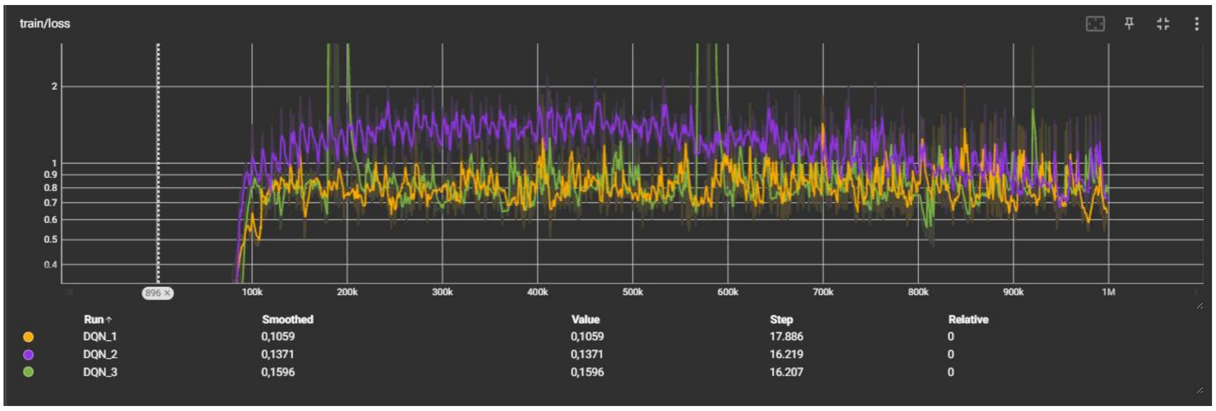
3.1.c) The evaluation of the "**ep _rew_mean**" value was performed in three distinct DQN scenarios.



	Run ↑	Smoothed	Value	Step	Relative
●	DQN_1	1.093	1.093	9.559	0
●	DQN_2	492	492	16.219	0
●	DQN_3	474	474	16.207	0

Figure 3: ep_rew_mean curves in three distinct DQN scenarios

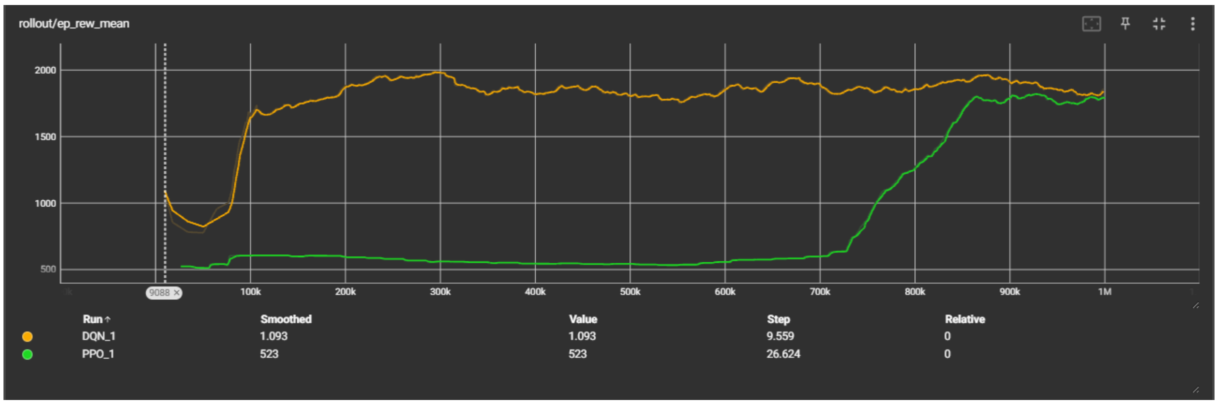
3.1.d) The evaluation of the "**loss**" value was performed in three distinct DQN scenarios.



	Run ↑	Smoothed	Value	Step	Relative
●	DQN_1	0,1059	0,1059	17.886	0
●	DQN_2	0,1371	0,1371	16.219	0
●	DQN_3	0,1596	0,1596	16.207	0

Figure 4: entropy_loss curves in three distinct DQN scenarios

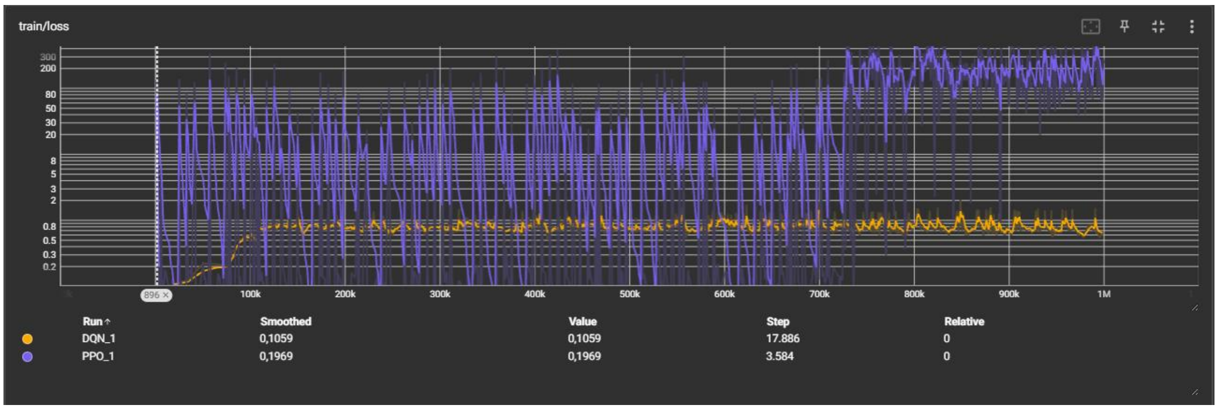
3.1.e) When comparing **PPO** and **DQN** in terms of their "**ep_rew_mean**", assuming the **usage** of **identical preprocessing methods**, the following insights can be observed:



	Run ↑	Smoothed	Value	Step	Relative
●	DQN_1	1.093	1.093	9.559	0
●	PPO_1	523	523	26.624	0

Figure 5: PPO & DQN comparison in terms of ep_rew_mean curves for identical preprocessing methods

3.1.f) When comparing **PPO** and **DQN** in terms of their "**loss**", assuming the **usage** of **identical preprocessing methods**, the following insights can be observed:



	Run ↑	Smoothed	Value	Step	Relative
●	DQN_1	0,1059	0,1059	17.886	0
●	PPO_1	0,1969	0,1969	3.584	0

Figure 6: PPO & DQN comparison in terms of loss curves for identical preprocessing methods

Note: However, due to the extended processing time and timeout limitations of Google Colab, I was unable to train the best model for the 5 million timesteps.

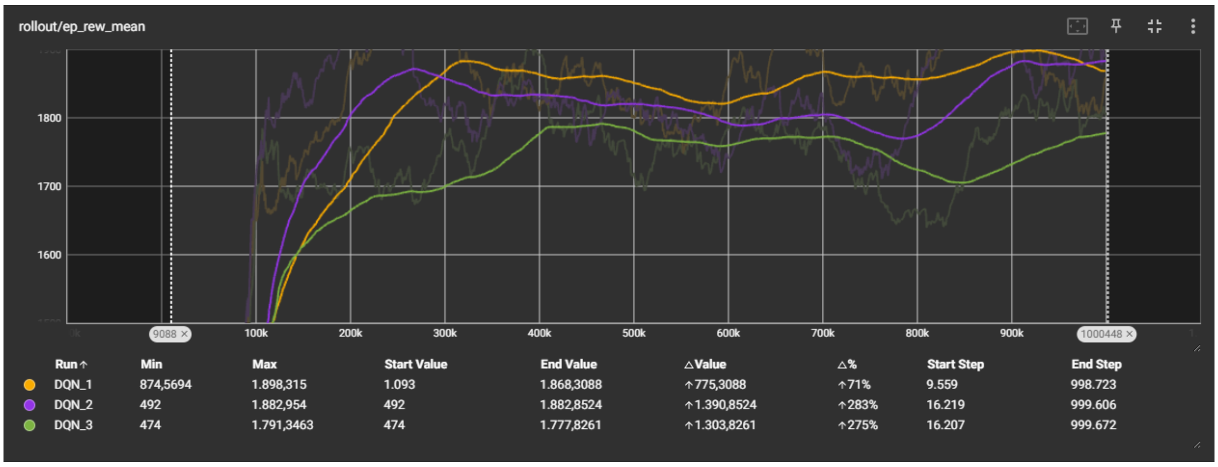
In order to identify the optimal algorithm, I saved and analyzed various videos. Consequently, I have selected the following model with its hyperparameters provided below.

DQN0 Hyperparameters:

```
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
)
```

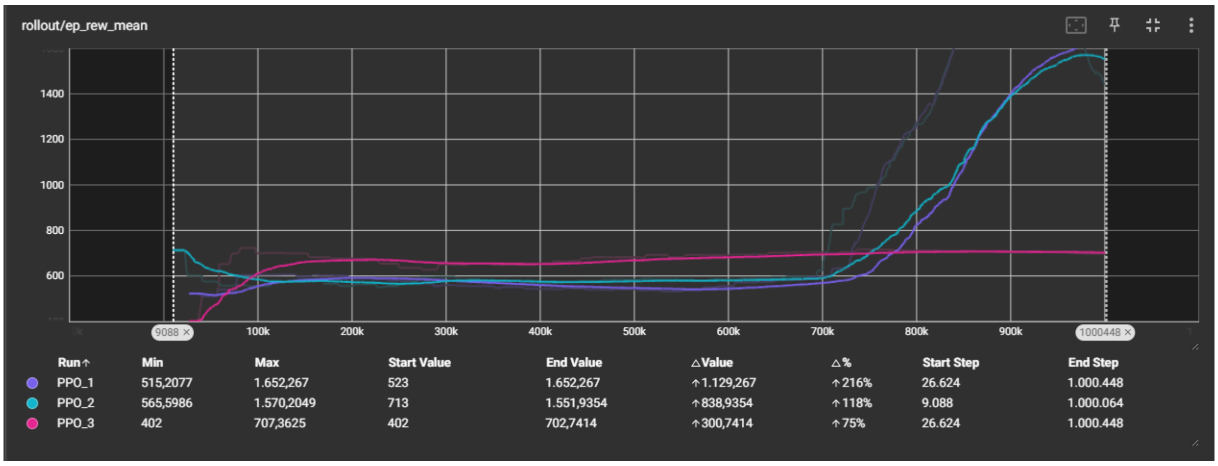
Video Link: https://youtu.be/aCvFwydV7iM

Best algorithm between DQN's adjust smoothing coefficient to 1.0 value. After that, to determine best one between PPO's. Finally, combine both graphs into a single graph.



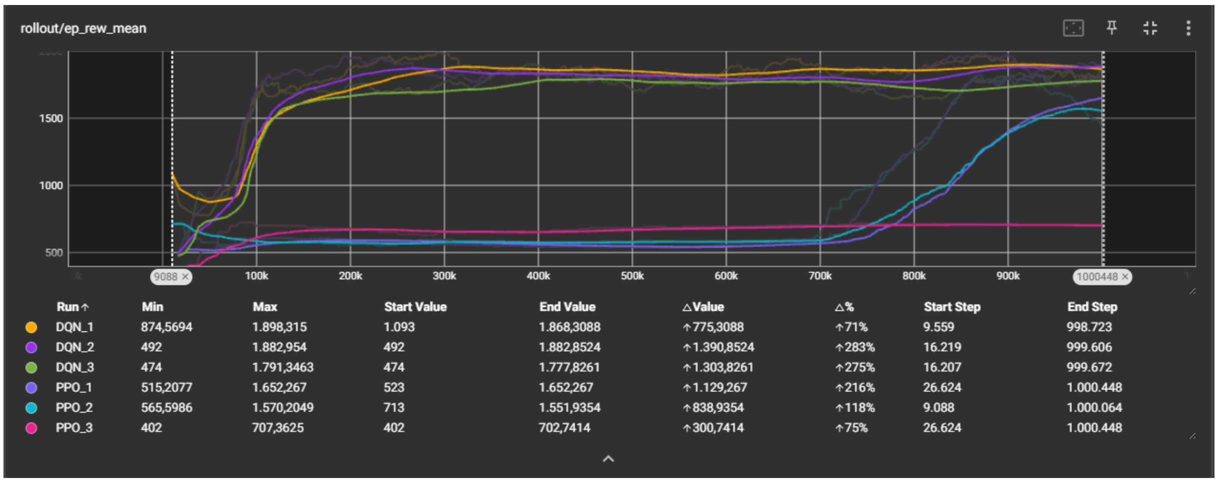
	Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
●	DQN_1	874,5694	1.898,315	1.093	1.868,3088	↑775,3088	↑71%	9.559	998.723
●	DQN_2	492	1.882,954	492	1.882,8524	↑1.390,8524	↑283%	16.219	999.606
●	DQN_3	474	1.791,3463	474	1.777,8261	↑1.303,8261	↑275%	16.207	999.672

Figure 7: ep_rew_mean curves in three distinct DQN scenarios with smoothing coefficient = 0.99



	Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
●	PPO_1	515,2077	1.652,267	523	1.652,267	↑1.129,267	↑216%	26.624	1.000.448
●	PPO_2	565,5986	1.570,2049	713	1.551,9354	↑838,9354	↑118%	9.088	1.000.064
●	PPO_3	402	707,3625	402	702,7414	↑300,7414	↑75%	26.624	1.000.448

Figure 8: ep_rew_mean curves in three distinct PPO scenarios with smoothing coefficient = 0.99



	Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
●	DQN_1	874,5694	1.898,315	1.093	1.868,3088	↑775,3088	↑71%	9.559	998.723
●	DQN_2	492	1.882,954	492	1.882,8524	↑1.390,8524	↑283%	16.219	999.606
●	DQN_3	474	1.791,3463	474	1.777,8261	↑1.303,8261	↑275%	16.207	999.672
●	PPO_1	515,2077	1.652,267	523	1.652,267	↑1.129,267	↑216%	26.624	1.000.448
●	PPO_2	565,5986	1.570,2049	713	1.551,9354	↑838,9354	↑118%	9.088	1.000.064
●	PPO_3	402	707,3625	402	702,7414	↑300,7414	↑75%	26.624	1.000.448

Figure 9: ep_rew_mean curves in three distinct DQN & PPO scenarios together with smoothing coefficient = 0.9

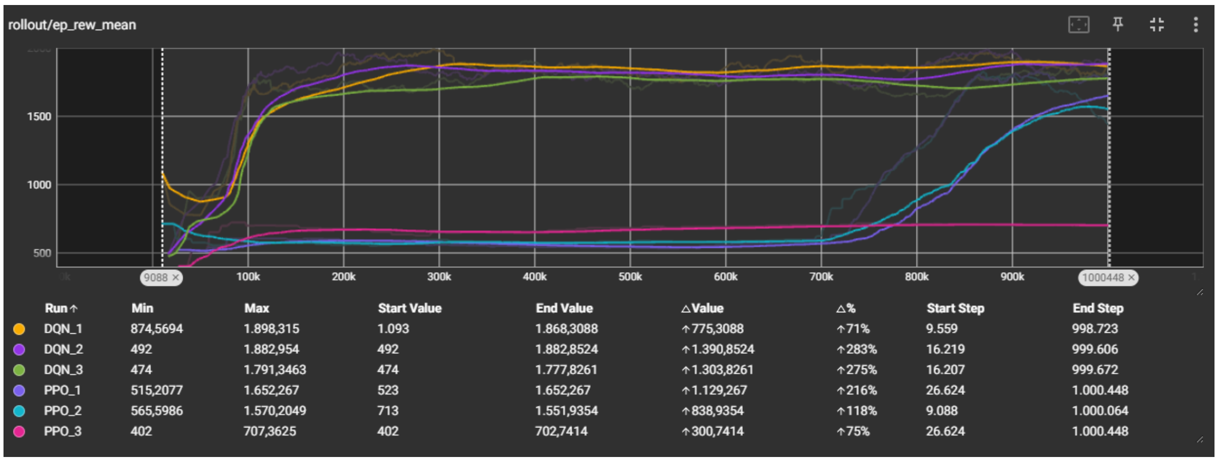
3.2) Discussions

3.2.1)

Indeed, I have watched the videos. Initially, Mario struggled to pass the first pipe. However, with each iteration, Mario gradually learned and became proficient at passing pipes. Among the videos, the highest score achieved by my agent was 800, which I included in the video. I visually observed the learning process of PPO and DQN and highlighted the initial stages as the worst performance, while showcasing the end of the learning process as the best evidence of improvement. Mario jumped over pipes after 500k iterations.

3.2.2)

To determine the learning speed and efficiency of two algorithms, we must examine their Learning Curves which is depicted as the average reward per episode (ep_rew_mean). By analyzing the six graphs of PPO1, PPO2, PPO3, DQN1, DQN2, and DQN3, our goal is to identify the algorithm that exhibits the fastest ascent and achieves higher rewards in earlier episodes. Upon careful examination of the graph, it becomes evident that both DQN1 and DQN2 exhibit a rapid ascent. However, DQN1 not only reaches the maximum reward but also demonstrates less fluctuation compared to DQN2. Based on these observations, we can conclude that the DQN1 algorithm is the most efficient and fastest learner among other algorithms.



	Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
●	DQN_1	874,5694	1.898,315	1.093	1.868,3088	↑775,3088	↑71%	9.559	998.723
●	DQN_2	492	1.882,954	492	1.882,8524	↑1.390,8524	↑283%	16.219	999.606
●	DQN_3	474	1.791,3463	474	1.777,8261	↑1.303,8261	↑275%	16.207	999.672
●	PPO_1	515,2077	1.652,267	523	1.652,267	↑1.129,267	↑216%	26.624	1.000.448
●	PPO_2	565,5986	1.570,2049	713	1.551,9354	↑838,9354	↑118%	9.088	1.000.064
●	PPO_3	402	707,3625	402	702,7414	↑300,7414	↑75%	26.624	1.000.448

Figure 10: Learning Curves for 3 distinct PPO & DQN scenarios

3.2.3)

Both algorithms PPO and DQN approaches exploration and exploitation problem in a distinct ways due to their different working principles.

DQN (Deep Q-Learning): DQN is a value-based algorithm that derives a policy by learning a value function. It utilizes a method known as the "greedy strategy" to investigate the environment. In this approach, the algorithm decides to explore a random action with probability ε while additionally selecting to exploit the best known action using probability (1-ε.). The balance progressively switches from exploration to exploitation as a result of steadily decreasing ε.

The PPO (Proximal Policy Optimization): approach simply seeks to improve the policy. A method for balancing exploration and exploitation is also included, although unlike in DQN, this equilibrium is implicitly maintained by policy optimization. In other words, PPO urges the new policy to not stray too far from the previous policy in an effort to avoid making radical changes that would stimulate damaging types of inquiry. Additionally, this guarantees that the policy makes use of prior information.

The primary challenge in reinforcement learning lies in finding the right balance between exploration and exploitation. In complex environments, the PPO algorithm tends to yield superior performance compared to the DQN. Conversely, in simpler environments, DQN often outperforms PPO.

3.2.4)

When it comes to the ability to generalize to new environments or unseen levels of a game, DQN has shown a tendency to outperform PPO at stable environments. However, the challenge arises when we consider generalization to new environments, where DQN may struggle to adapt and perform effectively. From an alternative standpoint, PPO demonstrates superior abilities in effectively handling and adapting to environmental changes, as compared to DQN. Unlike DQN, which relies heavily on the reward structure. Overfitting property of DQN prevents its generalization abilities. PPO's policy optimization approach allows it to adapt and become more readily in the case of unseen environments.

3.2.5)

Learning Rate: A high learning rate can lead to faster learning but might result in divergence. A low learning rate can slow down the learning but might result in more stable training and better convergence.

n_steps (PPO): A larger number of steps may cause more stable learning, but it may slow down the learning process.

batch_size (DQN): Larger batch sizes may stabilize the learning but require more memory and computational resources.

train_freq (DQN): A lower value provides the update of the model more frequently, leading to faster learning but also more computational expense.

buffer_size (DQN): Larger buffer can store more diverse experiences, possibly leading to better learning.

When considering the crucial parameters for PPO and DQN, we can provide the following answers. In both algorithms, the learning rate, policy choice, and exploration parameters play a significant role. Additionally, for PPO, the parameter n_steps can be considered important, whereas for DQN, learning_starts, batch_size, and buffer_size are vital factors.

3.2.6)

PPO is a policy optimization method that focuses on optimizing the policy directly. Its main computational cost arises from computing the policy gradient. When using a large batch size or a large number of steps, this can require a significant amount of memory and computation. Additionally, PPO requires fresh samples from the environment for each update, which can be time-consuming if the environment is complex. However, PPO is usually more efficient in terms of sample usage compared to DQN, meaning it needs fewer interactions with the environment to learn a good policy.

On the other hand, DQN is a value-based method that involves learning an action-value function. The computational cost in DQN comes from managing and sampling from a replay buffer, which requires substantial memory when using large buffer sizes. Another cost comes from computing the target Q-values and updating the Q-function, which can be computationally demanding, especially with large batch sizes. Unlike PPO, DQN can reuse past samples stored in the replay buffer, reducing the need for environment interactions. However, DQN is generally less sample-efficient than PPO, meaning it requires more interactions with the environment to learn a good policy.

Both PPO and DQN have been successfully applied to a wide range of tasks and environments, from simple control tasks to complex game environments. DQN is more suitable for tasks where there is abundant memory available and the environment is inexpensive to sample from. On the other hand, PPO is better suited for tasks with limited memory, expensive environment sampling, or rapidly changing tasks.

3.2.7)

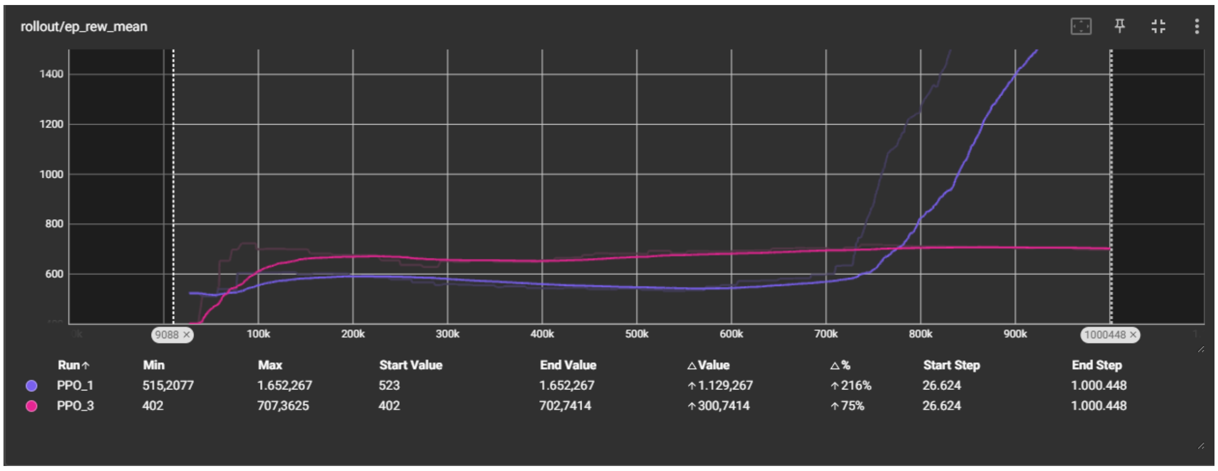
We can compare the MlpPolicy and CnnPolicy as follows:

The MlpPolicy consists of fully connected layers and is suitable for low-dimensional inputs. On the other hand, the CnnPolicy is composed of convolutional layers and is more appropriate for higher-dimensional inputs.

CnnPolicy is particularly well-suited for handling large-dimensional inputs like images because it is specifically designed to learn specific patterns from raw input images. This approach reduces the number of parameters required for the learning process compared to MlpPolicy, making it more computationally efficient and desirable.

In contrast, when utilizing the MlpPolicy for inputs with large dimensions, neural networks tend to lose the unique patterns present in image data. This occurs because fully connected layers do not take into account specific patterns that exist in the inputs. Furthermore, the MlpPolicy requires a greater number of parameters compared to the Cnn policy, resulting in a slower model and potentially leading to overfitting.

As a result, for the problems that requires handling with large dimensional input datas such as images, CnnPolicy would provide better resutls. Algorithms such as PPO and DQN are often used in conjunction with the CnnPolicy when dealing with visual inputs, such as playing video games like Mario using raw image data.



	Run↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
●	PPO_1	515,2077	1.652,267	523	1.652,267	↑1.129,267	↑216%	26.624	1.000.448
●	PPO_3	402	707,3625	402	702,7414	↑300,7414	↑75%	26.624	1.000.448

Figure 11: Learning Curves for CnnPolicy and MlpPolicy

After analyzing the provided curves, it is evident that PPO3 with MlpPolicy exhibits signs of overfitting at lower reward values. In contrast, the curve generated by CnnPolicy shows a steeper slope and reaches the maximum reward value by the end of its iteration process around 900k to 1M.