Training Super Mario Bros agents with Double Deep Q-Network and Rainbow DQN

1st Tonson Praphabkul
School of Engineering and Technology
Asian Institute of Technology
Pathum Thani, Thailand
Email : st123010@ait.asia

2nd Ati Tesakulsiri
School of Engineering and Technology
Asian Institute of Technology
Pathum Thani, Thailand
Email : st123009@ait.asia

***Abstract*—In this project, the Rainbow algorithm was used to create an agent that could play Super Mario Bros. in the OpenAI Gym environment. Double Deep Q-Network (DDQN) and two additional models, namely Rainbow with LSTM and Rainbow with Transformer, were compared to the Rainbow algorithm. The experiment's findings demonstrated that the DDQN model performed best because to its shorter training period and hardware constraints. Additionally, the study discovered that the Rainbow algorithm was trained more effectively by applying permanent experience replay memory. Overall, the research summarized in this study showed how reinforcement learning algorithms may be used to create agents that are capable of playing challenging video games like Super Mario Bros.**

***Index Terms*—Deep Reinforcement Learning, Double Deep Q-Network, Rainbow DQN, Transformers, Long Short Term Memory (LSTM)**

I. Introduction

Double Deep Q-Network (DDQN) is a popular reinforcement learning algorithm that has demonstrated remarkable success in mastering a wide range of games. However, DDQN struggles to master games that require a lot of experience to store in experience replay. These games are usually characterized by a large state space and require the agent to learn complex strategies over a long period of time. Storing a sufficient number of experiences in experience replay becomes a challenging task due to the large amount of memory required, which limits the agent's ability to learn effectively from past experiences.

The size of experience replay is related to the problem of catastrophic forgetting [6]. In reinforcement learning, catastrophic forgetting occurs when the agent forgets previously learned information when it learns new information. In the context of experience replay, catastrophic forgetting can occur when the agent updates its neural network based on a new set of experiences that are sampled from the replay buffer, and in the process, overwrites previously learned information.

Prioritized Experience Replay (PER) addresses the problem of catastrophic forgetting by selectively prioritizing experiences that the agent has not encountered frequently or experiences that the agent struggled with during training. By focusing on these experiences, the agent is more likely to retain important knowledge and avoid overfitting to a limited set of experiences. In addition, by prioritizing more informative experiences, the agent can learn more efficiently and make progress in its training faster [8]. But even with PER, the size of the replay buffer remains a limiting factor in reinforcement learning. The agent can only store a finite number of experiences, and if the game environment requires a large number of experiences to master, the replay buffer may become saturated and limit the agent's ability to learn effectively.

Thus DDQN will be unable to master games that require the agents to remember the large state because of limited number of experience. Increasing the size of experience replay can be very difficult due to hardware limitation. In this works we focusing on reducing the effects of catastrophic forgetting by increasing sample efficiency through Rainbow DQN, Rainbow DQN with LSTM and Rainbow DQN with Transformers.

II. Related Work

DeepMind utilizes a deep learning model that only accepts raw pixel input to perform Reinforcement Learning and demonstrate it using Atari 2600 video games. To train the model, stochastic minibatch updating is combined with experience replay memory. [7]

A. Super Mario Bros.

Super Mario Bros. is a platform game developed and published by Nintendo for the Nintendo Entertainment System (NES), and it is one of the most popular games of all time with over 58 million copies sold worldwide. The Mario movement is to walk and run to the right, left, and jump. Reaching the flag post at the end of each stage without Mario losing all of his lives is the game's main goal. The second goal is to reach the best score, which is achieved by gathering coins. Eliminate the monsters and finish a level quickly. Super Mario Bros. is suitable for testing basic RL models as this game has a high-dimensional state and observation state and also has relatively high-dimensional action spaces, which still require the execution of different skills in sequence [9] but do not require much computational power.

B. Double Deep Q-Networks (DDQN)

A multi-layered neural network that generates a vector of action values Q for a given state s. Two important ingredients

of the DDQN algorithm are the use of the use of experience replay, and a target network. Experience replay observed transitions are stored for some time and sampled uniformly from this memory bank to update the network. Making fixing the parameters θ_t^- in the target network, and only update them every τ time-steps so that $\theta_t^- \leftarrow \theta_t$ at designated intervals. Thus its objective function is [7]

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(s_{t+1}, a; \theta_t')$$

C. Rainbow DQN

Rainbow DQN is a reinforcement learning algorithm that builds upon the DQN algorithm by incorporating several additional techniques to improve learning and performance. The name "Rainbow" refers to the combination of several different methods used in the algorithm. [4]

1) Double Deep Q-learning (DDQN): Rainbow DQN uses a variation of the DQN algorithm called Double Q-Learning to address the issue of overestimation bias. Double Q-Learning uses two separate value functions to estimate the value of each action, which reduces the likelihood of overestimating the value of any given action. $Y_t^{DoubleDQN}$ with [10]

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(s_{t+1}, \underset{a}{argmax} Q(s_{t+1}, a; \theta_t); \theta_t')$$

2) Dueling networks: The models uses uses a dueling network architecture to separate the estimation of state value and action advantage. This allows the agent to learn which states are valuable and which actions to take within those states more effectively. [12]

$$q_\theta(s,a) = v_\eta(f_\xi(s)) + a_\psi(f_\xi(s),a) - \frac{\sum_{a'} a_\psi(f_\xi(s),a')}{N_{actions}}$$

where ξ, η, and ψ are, respectively, the parameters of the shared encoder f_ξ, of the value stream v_η, and of the advantage stream a_ψ; and $\theta = \xi, \eta, \psi$ is their concatenation.

3) Prioritized experience replay: DQN randomly selects samples from the replay buffer. Prioritized experience replay [8] captures transitions with probability p_t in relation to the most recent absolute TD error as a measure of learning potential. A bias toward recent transitions is created by the maximum priority with which new transitions are stored into the replay buffer.

$$p_t \propto \mid R_{t+1} + \gamma_{t+1} \underset{a'}{max} q_{\bar{\theta}}(s_{t+1}, a') - q_\theta(s_t, a_t) \mid^w$$

4) Distributional RL: Rainbow DQN uses distributional reinforcement learning, which represents the action-value function as a probability distribution over possible values rather than a single value. This allows the agent to learn more about the distribution of possible rewards, which can lead to more robust learning and better performance. Distributional variation of Q-learning is subsequently created by minimizing the Kullbeck-Leibler divergence between the distribution dt and the target distribution dt' . [2]

5) Noisy Nets: In games like Montezuma's Revenge, the limits of exploring utilizing Epsilon greedy rules are evident. A deterministic and noisy stream are combined in the noisy linear layer that Noisy Nets propose [3].The network can eventually learn to ignore the noisy stream, but it will do so at varying speeds depending on where in the state space it is.

D. LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that can be combined with reinforcement learning (RL) to improve learning and performance in sequential decision-making tasks. LSTM is particularly useful in RL settings where the agent must make decisions based on long-term dependencies and history. By using Advantage learning and directed exploration it can solve non-Markovian tasks with long-term dependencies between relevant events [1], [5].

E. Transformer

Transformer is a type of neural network architecture that has been widely used in natural language processing tasks [11], such as machine translation and text generation. It uses self-attention mechanism which allowed the models to selectively focus on different parts of input sequence.

In this work, we only used Transformer and LSTM as the new representation to learn the probability distribution for each action at each state.

III. Methodologies

In the practice section, we use the first level from the Super Mario Bros. game as an environment. By default, the single observation is a 256 x 240 pixels RGB image. We transforms input images into gray-scale image with a resolution of 84 x 84 pixels. Then we stacked 4 consecutive frame together as one single-observation point.

A. Model architecture

- Input: 4 gray-scaled consecutive frames with the shape of 4x84x84 (Number of frame, height, width)
- Hidden Layer: Convolves 32 8×8 filters of stride 4 with the input image and applies a rectifier non-linearity
- Hidden Layer: Convolves 512 4×4 filters of stride 2 and applies a rectifier non-linearity
- Hidden Layer: Convolves 64 3×3 filters of stride 1 and applies a rectifier non-linearity
- Hidden Layer: Fully connected layer with the size of 3136 $\rightarrow$ 128 and applies a rectifier non-linearity
- Advantage layer / Value layer
- Hidden Layer: Noisy linear layer with the size of 128 $\rightarrow$ 128
- Output Layer: Noisy linear layer with the size of 128 $\rightarrow$ Number of possible action

B. Model Hyper-parameters

- Observation space: 4 gray-scaled consecutive frames with the shape of 4x84x84
- Action space: 2 (['right'], ['right', 'A'])
- Loss function: KL divergence (cross entropy loss)

$$D_{KL}(\Phi\hat{\mathcal{T}}Z_{\tilde{\theta}(x,a)||Z_\theta(x,a)})$$

- Optimizer: Adaptive Moment Estimation (ADAM) with learning rate of 0.00025 and $6.25e^{-5}$ with ϵ of 0.00015
- Batch size: 32
- Gamma: 0.99
- Noisy Linear σ: 0.5
- Prioritization exponent α: 0.6
- Prioritization importance sampling β: 0.4
- Multi-step returns η: 3
- Distribution atoms: 51
- Distribution min: 0.001

C. Traditional Rainbow Method

All of the aforementioned components have been combined into one agent called Rainbow. First, a n-step return is used. in place of the 1-step distributional loss. By constricting the value distribution in S_{t+n} in accordance with the cumulative discount and shifting it by the shortened n-step discounted return, creating the target distribution. Standard proportional replay with priority. The transitions are ranked using the absolute TD error. This can be calculated using the mean action values in a distributional context. However, all distributional Rainbow variants in our experiments prioritize transitions by the KL loss. Dueling network design that has been modified for usage with return distributions makes up the network architecture. Then, It swap out all linear layers for their noisy counterparts. Applying factorized Gaussian noise within these noisy linear layers to lower the number of independent noise variables.

D. LSTM Rainbow method

Using a recurrent neural network to approximate the Q-value function. Sequence of states is given as the input and the network consists of RNN unit (LSTM/GRU), the output of the RNN unit at the final time-step is used to predict the Q-value of all possible actions is generated. In our experiments, the RNN unit (LSTM) is fed the sequence of partially observed states, the output from the final time-step is used to predict the Q-value function.

E. Transformer Rainbow Method

Replacing a transformer instead of RNN to extract the meaningful feature out of the input sequence. However, transformers were designed to work for sequence to sequence (seq2seq) tasks such as automatic speech recognition or language translation. But in this work we do not have sequence to sequence task, rather we need to predict the Q value function given the input sequence. In our experiments we only use the encoder module to extract the features from input sequence which are further used to predict the Q-values.

TABLE I
TRAINING TIME FOR 300K FRAMES (SECONDS)

Model	DDQN	Rainbow	Rainbow TF	Rainbow LSTM
Time(s)	**8821.51**	12818.10	29341.70	15951.73
R time(%)	100	145.31	332.62	180.83

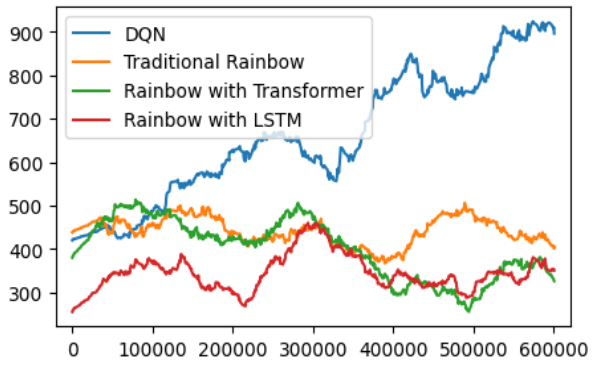

Fig. 1. Comparison between model

IV. EXPERIMENTAL RESULTS

A. Model Performance

In table I, our experiments compared the performance of three different reinforcement learning models in term of training time. The results showed that DDQN was the fastest model to train. followed by Rainbow, Rainbow LSTM and Rainbow TF.

Based on the experimental results on figure1, we found that the DDQN model significantly outperformed the other three models in terms of performance. Specifically, the DDQN model achieved significantly better results compared to the Rainbow with LSTM, Rainbow with Transformer, and Rainbow DQN models. However, among the Rainbow variants, the Rainbow DQN model performed slightly better than the Rainbow with LSTM and Rainbow with Transformer models.

We think the reason for poor performance of Rainbow DQN is because the models is more complex and it require more training time to achieve optimal results. The other reasons is that Rainbow DQN requires extensive tuning for each specific task, we think the current hyper-parameters might not be appropriate for this tasks.

B. Ablation Study

Based on the ablation study results, In the figure 2 we found that the model using 5 action spaces outperformed the model using only 2 action spaces. Although the model with 5 action spaces initially had lower performance, it eventually surpassed the model with 2 action spaces. These results suggest that having a larger action space can be beneficial for the model's overall performance, even if it takes longer for the model to learn how to use the additional actions effectively. These findings may have important implications for improving the performance of reinforcement learning models in tasks that require more complex decision-making.

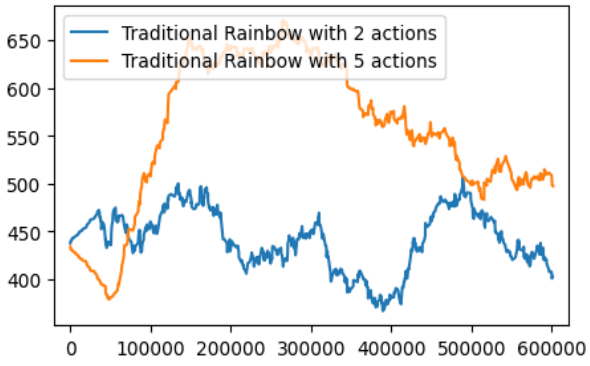

Fig. 2. Comparison between Action Space (Dist/No of Frame)

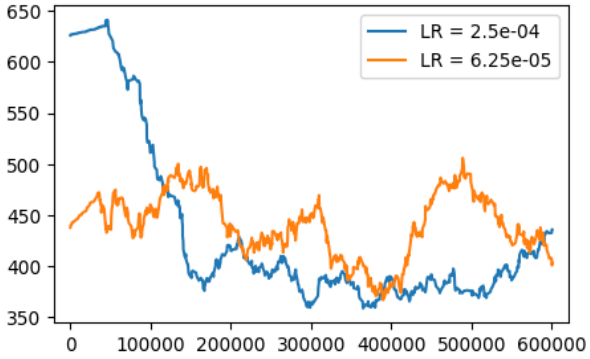

Fig. 3. Comparison between Learning Rate (Dist/No of Frame)

In the figure 3, we investigated the impact of different learning rates on the performance of our agent. We trained two models using learning rates of 2.5e-04 and 6.25e-5 respectively. Our results show that the model trained with a learning rate of 6.25e-5 outperforms the model trained with a learning rate of 2.5e-04. Specifically, the model trained with a higher learning rate achieved a lower average reward. We suspect that the learning rate of 2.5e04 is too high for this environment. It leads to unstable and unpredictable training behavior, which causes the model to converge to a sub-optimal solution.

In the figure 4. The results showed that the model with permanent experience replay significantly outperformed the model without it. The permanent experience replay allowed the model to retain important experiences and learn from them over a longer period of time. This resulted in better exploration and exploitation of the state-action space and a more stable learning process. On the other hand, the model without permanent experience replay struggled to learn from past experiences, leading to slower convergence and a lower overall performance. Therefore, it can be concluded that permanent experience replay is an important component of the reinforcement learning algorithm and can significantly improve the performance of the model. For the permanent experience replay, we store first 25,000 memory states as the permanents memory.

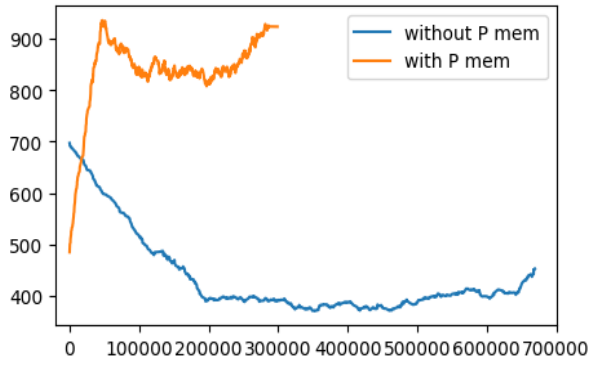

Fig. 4. Comparison between Permanent Memory and Non-Permanent Memory (Dist/No of Frame)

V. Conclusion and Future Work

To conclude, this research was successful in creating an agent that can play Super Mario Bros. using the Rainbow algorithm. Other reinforcement learning algorithms, including DDQN, Rainbow with LSTM, and Rainbow with Transformer, were used to compare the agent's performance against. The DDQN algorithm beat the other algorithms in terms of hardware needs and training time, according to the results. The study also discovered that applying persistent memory aided in improving model performance.

References

[1] Bram Bakker. Reinforcement learning with long short-term memory. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[2] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017.

[3] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration, 2019.

[4] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[6] Prakhar Kaushik, Adam Kortylewski, Alex Gain, and Alan Yuille. Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[8] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[9] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[10] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[12] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.