Training Flappy Bird with Deep Q Network and SARSA

Choo Huan Long
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp060974@mail.apu.edu.my

Liew Wen Heng
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp075352@mail.apu.edu.my

Wong Ying Cheng
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp062394@mail.apu.edu.my

Yong Zhen Xing
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp068733@mail.apu.edu.my

Zailan Arabee Abdul Salam
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
zailan@apu.edu.my

***Abstract*— Deep Q-Network (DQN) is implemented in this Flappy Bird Game and the purpose of project is to tweak and change the parameters to meet the desire outcome, which is passing pipes if the agent can. DQN are employed into the project to maximize cumulative reward while make decisions in real-time gameplay. In this project, the Flappy Bird game environment is set up along with the state and action spaces, the Q-network, an experience replay buffer, and a training loop. Using ε-greedy policies, the agent learns to make the best decisions by balancing exploration and exploitation. The Deepmind's experience replay was employed to enhance the stability of learning. Adam optimizer and SDG optimizer is monitoring the agent`s train/loss, and the α and ε will be modified to make a comparison between different parameters.**

Keywords—Deep Q-Network, Epsilon-Greedy Exploration, Adam Optimizer, SGD Optimizer, Flappy Bird

I. INTRODUCTION

In this Flappy Bird Project, Deep Q-Network has been utilised and it is a convolutional neural network, trained with a variant of Q-learning. By training the agent to play the flappy bird game, the agent needs to know the input, which is raw pixels to make output, value function that estimates future rewards. The Deep Q-Network (DQN) algorithm was developed by DeepMind in 2015. It was able to solve Atari games by combining the deep neural networks and reinforcement learning. The algorithm was developed by enhancing a reinforcement learning (RL) algorithm called Q-Learning with deep neural networks and a technique called experience replay.

II. LITERATURE REVIEW

A. Similar Project

There is a similar project that use deep reinforcement learning in their project, which is Recommender system enhancement using deep reinforcement learning. (Shuhrat, et al., 2021) In this project, they use Deep reinforcement learning in their recommendation system. Their objective is to provide an efficient service, as include the user preference and recommendation, to the user. The algorithm needs to learn the user preferences and provide suggestions to the user.

Another similar project is using convolutional neural network for fashion images classification (Tan et al., 2023). In this project, they are using Convolutional Neural Network to recognizing images. There are aiming to identify the parameters will affect the accuracy of the trained model while the model is doing the image classification.

B. Methodology/ Approach

In the deep reinforcement learning, non-RL based recommendations and RL based recommendations are implemented in their project. Non-RL based recommendations is a recommendation that didn't use reinforcement learning to develop the suggestion approach, included content-based recommendation. It uses static or time as a feature. RL based recommendations refer to recommendation system that use reinforcement learning to develop the suggestion approach. It explores the probabilistic and lead to decision-making, such as Bayesian strategies. The agent needs to take decisions based their training, and maximize for the rewards, to improve the performance, balance exploration and exploitation. Artificial Neural Network (Anns) serves as the basic for deep learning approaches, and it is used to classify the images. There are three layers, which is input, one hidden layer and output layer, and the threshold are connected. The result should be able to show the accuracy, loss, and training time for each batch of images. Multiple of operations will be collected by Convolutional Neural Network (CNN) and known as convolution layers. The weight and biases will be implemented to the CNN model and the input data will need to be train. The feature maps will be the output of CNN model will show in arrays.

C. Conclusion/ Recommendation

In deep reinforcement learning, they came out with the conclusion that to create a hybrid engine it is important to take into consideration that different approaches to develop the recommendation system. In the literature, they recommend using RL-based recommendation systems that use reinforcement learning techniques to develop suggestion approaches. Additionally, they recommended to make the use of multiple exploration techniques in combination with the Deep Q-Network. In CNN project, different table and data were recorded and the table contain of training loss, accuracy,

testing loss and accuracy. The SGD optimizer were better than Adam optimizer, and different batch size will produce different of loss and accuracy. In conclusion, a perfect parameter will significantly affect the model to produce the outcomes.

III. Materials and Methods

We utilized Python 3.10 as a tool to run the Flappy Bird game that was packaged with Q-learning algorithm and Deep Neural Network. We use the source code developed by **uvipen** found in GitHub. We are able to modify the parameters and train the agent to play Flappy Bird game thanks to the source code. The link to the source code: https://github.com/uvipen/Flappy-bird-deep-Q-learning-pytorch

A. Deep Q-Network

Deep Q-Network is a type of reinforcement learning algorithm, made up of Q-learning and Convolutional Neural Networks with the goal of maximizing the reward. In the off-policy Q-learning, Bellman Equation is utilized, and the Q value is updated iteratively. Here is the Bellman Equation:

$$Q_{i+1}(s,a) = r + \gamma \max_{a'} Qi(s',a')$$

s': state of next frame

a': action of next frame

r: reward

γ: discount factor

$Q_i(s,a)$: the state and action at the i^{th} iteration.

It can be seen that by updating it iteratively, it will eventually get to the optimal Q-function. However, this can lead to rote-learning, where the model memorized the states instead of generalizing it. To prevent this, deep neural network is used to create a model that can approximate the optimal Q-function of unseen states as well (Appiah & Vare (2018). This update ignores the unpredictability of state transitions in the game.

Since the states (sequence of pipe) for every game are different, generalizing the knowledge of every previous game to future ones is crucial. In order to approximate the Q-values for generally (for unseen states as well), Deep Q-Network is built which takes a model, namely Q_θ and slowly discovers a θ through iterations such that $Q_\theta(s,a)$ can approximate the Q value for every state and action.

Since the input for the algorithm will be images, convolutional neural network (CNN) is the most effective way to represent the Q-function as it is very good at extracting significant features from images (Pilcer, et.al. 2015). The CNN is made up of three convolutional layers, two dense layers, with linear functions and Rectified Linear Units (ReLU). It takes 4 consecutive game screens as input to estimate the Q-value for actions a=0 (do nothing) and a=1 (jump).

By utilizing gradient descent on loss function during training, the weights of the neural network can be updated. The loss function is defined as:

$$loss(i) = 1/2\,[r_i + \max_{a'} (Q(s', a')) - Q(s_i, a_i)]^2$$

The goal is to minimize the loss function by modifying the parameters of the network iteratively. A good exploration technique, expressed by the parameter ε (epsilon), is crucial for effective training. During training, the agent will choose an action based on probability ε or perform the action with the greatest Q-value.

B. Experience Replay

We employed Deepmind's experience replay method to enhance the stability and convergence of Q-value by the approximation of Q-function over time. The last experiences, which are consist of

$$s_i, a_i, r_i, s_{i+1}$$

will be stored in a memory buffer that has a fixed capacity. The oldest experiences will be discarded when the storage is full to make space for new experiences. A batch experiences in replay memory are sampled randomly when training the neural network through gradient descent instead of using them in the order that they are collected (Pilcer, et.al. 2015). This can stabilize training as it breaks the temporal correlation between successive experiences. The loss is determined by comparing the predicted Q-values for the sampled experiences with the target Q-values, which encourages the neural network to approximate an optimal Q-function. The code for experience replay memory buffer is shown in Fig. 1.

C. Epsilon-Greedy Exploration

We started off by implementing an epsilon greedy

```
parser.add_argument("--num_iters", type=int, default=100000)
parser.add_argument("--replay_memory_size", type=int, default=50000,
                    help="Number of epoches between testing phases")
```

Fig. 1: Source code of experience replay

strategy as we required some way to initially explore some state space, that is, choosing a random action with probability ε (initial epsilon), otherwise perform the action that maximizes the Q-function (Tran, 2020).

The ε is initialized by --initial_epsilon as 0.99, indicating that there is a 99% chance of exploration and 1% chance of exploitation at the beginning of the training as seen in Fig. 2.

```
parser.add_argument("--initial_epsilon", type=float, default=0.99)
parser.add_argument("--final_epsilon", type=float, default=1e-4)
```

Fig. 2: Source code of epsilon

As training progresses, the epsilon is annealed linearly over time, which means it decreases linearly over time from initial epsilon to final epsilon. This is done to shift the agent's focus to exploitation from exploration over the training period. The code where this is implemented is shown in Fig. 3:

```
# Exploration or exploitation
epsilon = opt.final_epsilon + (
        (opt.num_iters - iter) * (opt.initial_epsilon - opt.final_epsilon) / opt.num_iters)
```

Fig. 3: Source code for exploration or exploitation

D. Adam Optimizer (Adaptive Moment Optimizer)

Adam Optimizer algorithm automatically adjusts the learning rates for each parameter individually based on past gradient information, which regularizes and optimizes the parameters and enables the model to learn more effectively and converge faster. It helps to prevent the model from getting

stuck in local minima during the training due to complicated and uneven structure of the loss landscape. The inclusion of momentum term speeds up the process of gradient descent, allowing a faster convergence (Jiang, et al. 2020).

IV. RESULT

A. Optimizer

By keeping the α value at 0.1, γ value of 0.99, initial ε greedy at 0.9, the loss graphs of the results are shown below. Figure 4 indicates the loss graph of using Adam optimizer while Figure 5 represents the loss graph of using SGD optimizer.

As shown in Fig. 4, we can observe that the loss value is gradually decreasing, the big fall is clearly started at around 3500 iterations and decreased steadily until it reached convergence at around 5000 iterations. This means that the parameters provided are effective for training the agent.

In Fig. 5, the SGD optimizer costs more iterations to observe a convergence trend, which is started at around 4800 iterations and reaches convergence at around 5500 iterations. However, we observe that there are multiple spikes after the model reached convergence, which are at around 6100, 9500, 13000 iterations. The reason for sudden spike to occur may be the ε greedy approach that the agent switches from exploitation to exploration which will increase the loss value. Besides, the model of SGD optimizer is much closer to the convergence compared to the ADAM optimizer.

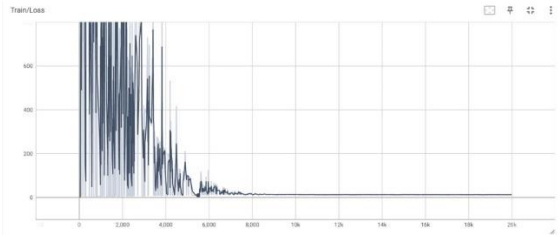

Fig. 4: Adam Optimizer Loss Graph

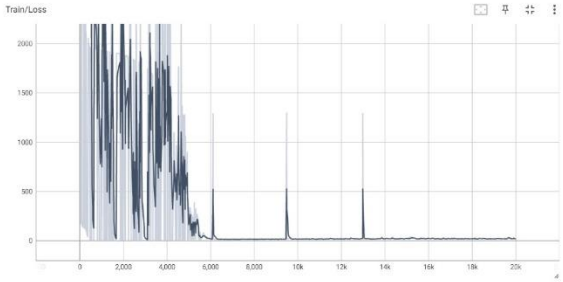

Fig. 5: SGD Optimizer Loss Graph

B. Learning rate (α)

In Fig. 6, the graph is scores versus iterations with two different α values of model, blue line indicates 0.1 α value, while the orange line indicates 0.01 α value. We can observe that both models have similar trends which is plateau occurred. This may be because other parameters are affecting the stability for the model to obtain scores. Besides, the α value of 0.1 increases faster than the 0.01 α value which means that the higher α value allows the agents to gain more scores in shorter iterations.

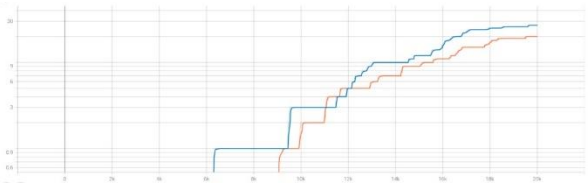

Fig. 6: Learning rate Graph

C. Epsilon (ε) Value

In Fig. 7, the graph is score versus iteration with two different ε values which are 0.5, indicated by the blue line and the grey line representing 0.9 ε value. The starting point for model to obtain score is at around 1700 iterations for 0.5 ε value while the 0.9 ε value started at around 7200 iterations. This is because the higher ε value tend to explore more state in the environment before change to exploitation phase to maximize the reward. Besides, we can notice that both models have a much bigger curve began from 14000 iterations, before that the curve of models are much slower.

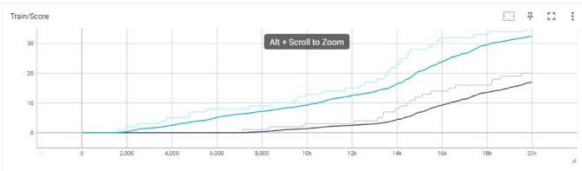

Fig. 7: Epsilon (ε) Graph

V. CONCLUSIONS

However, the data given to the algorithm highly influences the run-time, but we are still able to receive a meaningful result in a limited timeframe even though it will take long training model time. While we were tweaking with the algorithm, the appropriate parameters were implemented. The expectation and result with the training model was met within a short learning time, which approximately 5 hours for each training. Besides that, the comparison of Adam optimizer provides a smoother and lower loss than SGD optimizer, and higher learning rate make agent to achieve higher score in a short iteration. As a result, the Q-Learning algorithm can be continuously improved by tweaking a more perfect parameter.

ACKNOWLEDGMENT

We would like to express our gratitude to uvipen for providing DQN in Flappy Bird source code for us to test the model.

REFERENCES

Shuhrat, B., Ramachandran, C. R., Abdul Salam, Z.A. (2021). Recommender systems enhancement using deep reinforcement learning. Journal of Applied Technology and Innovation. vol. 5, no. 4. e -ISSN: 2600-7304

Tang, J.S., Tey, J.Y., Pu, J.Y., Por, J.X., Voon, P.Y., Abdul Salam, Z.A. (2023) Convolutional Neural Network for Fashion Images Classification (Fashion-MNIST). Journal of Applied Technology and Innovation vol. 7, no. 4. e -ISSN: 2600-7304

Pilcer, L.S., Hoorelbeke, A., D'andigne, A. (2018). Playing Flappy Bird with Deep Reinforcement Learning. IEEE Transactions on Neural Networks, 6.

Appiah, N., & Vare, S. (2018). Playing FlappyBird with Deep Reinforcement Learning., 6.

Tran, T. V. (2020). FlapAI Bird: Training an Agent to Play Flappy Bird Using Reinforcement Learning Techniques. ArXiv, 13.

Jiang, X., Hu, B., Satapathy, S.C., Wang, S.H., Zhang, Y.D. (2020). Fingerspelling Identification for Chinese Sign Language via AlexNet-Based Transfer Learning and Adam Optimizer. Scientific Programming. https://doi.org/10.1155/2020/3291426