Shape

Description automatically generated with medium confidence**Deep Q-Network for Stochastic Process Environments**

Kuangheng Gerald He

**Abstract:** Reinforcement learning is a powerful approach for training an optimal policy to solve complex problems in a given system. This project aims to demonstrate the application of reinforcement learning in stochastic process environments with missing information, using Flappy Bird and a newly developed stock trading environment as case studies. We evaluate various structures of Deep Q-learning networks and identify the most suitable variant for the stochastic process environment. Additionally, we discuss the current challenges and propose potential improvements for further work in environment building and reinforcement learning techniques.

**Keywords: Reinforcement Learning, Stochastic Process, Flappy Bird, Stock Trading Simulation.**

1. **INTRODUCTION**

The primary objective of our research is to develop an optimal policy for predicting the behavior of a stochastic process environment. To achieve this, we will be using the game Flappy Bird as our starting point, as it has been previously demonstrated as a suitable environment for reinforcement learning.

In the first part of our project, our goal is to train a DQN (Deep Q-Network) agent to successfully play the game of Flappy Bird. The game involves navigating a bird through pipes while avoiding obstacles to earn points. The agent will be provided with position information and the current score, and it must learn to recognize the bird and pipes and locate them on its own. The game's state space is challenging, requiring the agent to generalize its learning to successfully play the game better than even human players.

Fig 1: The game image of Flappy Bird env

In the second part of our research, we will train a DQN agent with previous experience gained from the Flappy Bird environment. To accomplish this, we modified the Flappy Bird environment to a stochastic process environment and developed a stock trading simulation environment with essential trading metrics. The DQN agent will be trained by imitating the network design developed in the first part of our project. The agent will receive a larger observation space beyond the single position value provided in the Flappy Bird environment. Finally, the agent will be able to trade using the trained policy and earn profit in a backtest class.

Overall, our research aims to develop a robust and generalized DQN agent that can predict and adapt to the behavior of stochastic process environments such as stock trading simulations. This will contribute to the ongoing effort to develop more efficient and effective AI algorithms that can adapt to complex and dynamic environments.

1. **RELATED WORK**

Previous work in this area has primarily been conducted by Google DeepMind. Mnih et al. successfully trained agents to play Atari 2600 games using deep reinforcement learning, achieving performance exceeding that of human experts on multiple games. In the domain of game environments, many successful attempts have been made to train agents using gym. For instance, Kevin Chen (2017) employed Deep Q-Networks (DQN) to train an agent to play Flappy Bird, achieving an average score of 215. In the realm of stock trading, Lin Willam Cong et al. demonstrated the flexibility of using deep reinforcement learning to manage portfolios without tagging information.

1. **METHOD**
2. Deep Q-network

The Deep Q-Network (DQN) algorithm used in our research for the Flappy Bird environment is based on a PyTorch implementation of a neural network with three linear layers and two dropout layers. The neural network is defined as a class called enhancedDQN, which takes in the number of input features, the number of hidden units, and the number of output actions as inputs. The forward function of the network applies ReLU activation to the output of each linear layer and applies dropout with a probability of 0.1 to the output of the first two linear layers. The output of the final linear layer represents the Q-values for each possible action. The DQN algorithm learns the optimal Q-values by minimizing the mean squared error loss between the predicted Q-values and the target Q-values, calculated using the Bellman equation:

\begin{equation}

Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max\_{a'} Q(s', a') - Q(s, a)]

\end{equation}

where $Q(s,a)$ is the Q-value for a state-action pair, $s$ is the current state, $a$ is the current action, $r$ is the reward for taking action $a$ in state $s$, $\gamma$ is the discount factor, and $\alpha$ is the learning rate. The agent selects actions according to an $\epsilon$-greedy policy, where with probability $\epsilon$ it selects a random action and with probability $1-\epsilon$ it selects the action with the highest Q-value.

1. Kaiming initialization

In our research, we utilized the Kaiming initialization method for the weights of our DQN network. The Kaiming initialization is a method of weight initialization that is designed to improve the convergence speed and performance of deep neural networks. This method initializes the weights using a Gaussian distribution with mean 0 and variance $\frac{2}{n}$, where $n$ is the number of input features. The Kaiming initialization is applied to the weights of the linear layers using the PyTorch with a nonlinearity of ReLU. In addition, a constant bias of 0.1 is added to the linear layers. These weight and bias initialization methods help to improve the performance of our DQN agent in learning the optimal Q-values and navigating the Flappy Bird environment.

1. Replay memory method

Our implementation of the DQN algorithm in the Flappy Bird game involved the use of a replay memory method. The ReplayMemory class stored recent transitions consisting of the current state, action taken, next state observed, and corresponding reward, in a deque data structure with a maximum capacity. The push method added a new transition to the memory buffer, and the sample method randomly selected a batch of transitions for training. This method helped decorrelate transitions, improve sample efficiency and provide a more stable training process for the DQN agent.

1. Model Optimization

The optimize model function performs one optimization step in the DQN algorithm by computing the expected state-action values using the Bellman equation:

$Y\_{t} = R\_{t+1} + \gamma \max\_{a} Q(S\_{t+1},a)$

It defining the Huber loss function:

$L = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & \text{if } |y - \hat{y}| \leqslant \delta \ \delta(|y - \hat{y}| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$

The function calculating the loss between expected and predicted state-action values, backpropagating the loss, clipping gradients, and performing an optimization step. It utilizes the policy and target networks to update the Q-values of the agent's policy.

1. Training Pipeline
2. 6
3. 7
4. **RESULTS AND DISCUSSION**
5. **CONCLUSIONS**
6. **BIBLIOGRAPHY**

Chen, Kevin. Deep Reinforcement Learning for Flappy Bird - Stanford University. 2017, <https://cs229.stanford.edu/proj2015/362_report.pdf>.

Chuchro, Robert, and Deepak Gupta. Game Playing with Deep Q-Learning Using Openai Gym. 2017, <http://cs231n.stanford.edu/reports/2017/pdfs/616.pdf>.

Cong, Lin and Tang, Ke and Wang, Jingyuan and Zhang, Yang, AlphaPortfolio: Direct Construction Through Deep Reinforcement Learning and Interpretable AI (August 1, 2021).

He, Kaiming, et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.” 2015 IEEE International Conference on Computer Vision (ICCV), 2015, https://doi.org/10.1109/iccv.2015.123.