# Playing Aircraft Warfare Game with Reinforcement Learning

**Yan Zeng**
ShanghaiTech University
zengyan@shanghaitech.edu.cn

**Yijie Fan**
ShanghaiTech University
fanyj@shanghaitech.edu.cn

**Luojia Hu**
ShanghaiTech University
hulj@shanghaitech.edu.cn

**Ziang Li**
ShanghaiTech University
liza1@shanghaitech.edu.cn

**Chongyu Wang**
ShanghaiTech University
wangchy5@shanghaitech.edu.cn

## Abstract

## 1 Introduction

Aircraft Warfare is a classic game we all enjoyed very much, which is also perfect for fully practicing what we have learned in class, as shown in Figure 1. The rule of the game is rather simple. The **goal** of the player – a upward facing aircraft plane – is to make the score as high as possible. The player can get **reward** by managing to hit enemies – downward facing aircraft planes – with five **actions**, namely, up, down, left, right, and using the bomb. The **state** includes the life value and positions of player and enemies and so on. Game overs when life value decreases to 0.

However, playing the game well is quit tough when it comes to difficult mode. Hence we turn to AI for help. To the best of our knowledge, reinforcement learning has shown to be very successful in mastering control policies in lots of tasks such as object recognition and solving physics-based control problems[]. Specially, Deep Q-Networks (DQN) are proved to be effective in playing games and even could defeat top human Go players[]. The reason they can work well is that games can quickly generate large amounts of naturally self-annotated (state-action-reward) data, which are high-quality training material for reinforcement learning. That is, this property of games allows deep reinforcement learning to obtain a large number of samples for training to enhance the effect almost costlessly. For example, DeepMind's achievements such as playing Atari with DQN, AlphaGo defeating Go masters, AlphaZero becoming a self-taught master of Go and Chess. And OpenAI's

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

main research is based on games, such as the progress made in Dota. For these reasons, we believe that training agents based on deep reinforcement learning techniques is a promising solution

In this paper, we implement an AI-agent for playing the Aircraft Warfare with Approximate Q-learning method and Deep Q-learning method. For Approximate Q-learning, we extracted four most useful features, i.e., interactions with the closest aircraft, bomb supply, double bullet, and the trade-off between movement and explosion. We trained our agent with an online learning method. For Deep Q-learning, we utilize a convolutional neural network which takes the screen patch as input and outputs the expected converged sum of the discounted reward of taking each action given the current state of the 6 legal actions. We also trained two DQN in case overfitting.

result...


Figure 1: The Aircraft Warfare Game

## 2 Methodology

### 2.1 Approximate Q-learning

Since the number of states in the Aircraft Warfare Game is extremely large, we choose Approximate Q-learning for reinforcement learning.

#### 2.1.1 Feature Extraction

In total, we extracted four features, which are the interaction with the closest aircraft, the interaction with bomb supply, the interaction with double bullet , and the trade-off between movement and explosion. The following will explain in detail how the features are extracted.

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + w_3 f_3(s,a) + w_4 f_4(s,a)$$

In this game, it is very necessary to control the distance between our aircraft and enemy aircraft and props.Thus,we use the Manhattan distance to measure the interaction of the aircraft with the external environment. For the current state $s$, we observe the location of the nearest enemies, game props to the aircraft. Next, we analyze the effect of action $a$ on the Manhattan distance between the aircraft and the aforementioned target. The weights corresponding to each of these features reflect the choices and trade-offs made by the aircraft in various situations.The purpose of the plane's actions can be highly summarized as attacking, dodging, and picking up props. Both affect the value of $Q(s,a)$ and the training of Approximate Q-learning.

#### 2.1.2 Training

In the training phase we train the learning with multiple hyperparameters and update the weights according to the formula below.

$$\text{difference} = r + \gamma \max_{a'} Q(s',a') - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\text{difference}]$$

$$w_i \leftarrow w_i + \alpha[\text{difference}] f_i(s,a)$$

Our training is an online learning approach. After getting a new state $s$ in each round, we choose the action $a$ with the highest $Q(s,a)$. After taking the action in the current round, we get state $s$. Then,

we use these two states with association to obtain the difference and update the parameters of the model. $\gamma$ is the discount parameter.$\alpha$ is the learning rate. The adjustment of these two parameters can better help our model to converge.

### 2.2 Deep Q Network

Deep Q Network is a kind of Deep Reinforcement Learning, which is a combination of Deep Learning and Q-learning. Due to the limitations of Q-learning that it is impossible to choose the best action when the number of combinations of states and actions is infinite, using a deep neural network to help determine the action is reasonable.

#### 2.2.1 DQN algorithm

In the Aircraft Warfare Game the environment is deterministic, so all the equations listed below are formulated deterministically for simplicity. Our aim will be to train a policy that can maximize the cumulative, discounted reward $R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$,here the $\gamma$ is the discount factor,which should be a constant between 0 and 1 to make sure the sum can converge.In the Q-learning algorithm, we get a table of the $Q$ values of the combinations of states and actions, then we construct a policy that maximizes the rewards:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s, a)$$

However, since the number of the combinations of states and actions is infinite in this scene, so we use a neural network to resemble $Q^*$. And by Bellman equation,we get:

$$Q^\pi(s, a) = r + \gamma Q^\pi(s', \pi(s'))$$

The difference between the two sides of the equation is known as the difference discussed in the lecture:

$$\delta = Q(s, a) - (r + \gamma \max_{a'} Q(s', a'))$$

To minimize this difference, we use the Huber loss which acts like the mean squared error when the error is small, but like the mean absolute error when the error is large - this makes it more robust to outliers when the estimates of $Q$ are very noisy. We calculate this over a batch of transitions, $B$, sampled from the replay memory:

$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r)\ \in\ B} \mathcal{L}(\delta)$$

$$\text{where} \quad \mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

#### 2.2.2 DQN Structure and Training

Our network is a convolutional neural network which takes the screen patch as input and outputs the expected converged sum of the discounted reward of taking each action given the current state of the 6 legal actions. To prevent overfitting, we trained two Deep Q Networks: policy network and target network. They have the same structure but the parameters are different. We get the best action from the policy network and computes the $\max_{a'} Q(s', a')$ from the target network for added stability. How can this method help prevent overfitting? Suppose there is only one network, when the parameters are updated, not only does the $Q(s, a)$ change, but the $\max_{a'} Q(s', a')$ also changes then the loss we want to minimize is always changing. By introducing the target network we can temporarily fix $\max_{a'} Q(s', a')$ which makes the loss a fixed value to help prevent overfitting. To explore the environment, we use the $\epsilon$ greedy method when choosing the action and the value of $\epsilon$ is decayed with time to lower the regret.

## 3 Result