

SHANGHAI JIAO TONG UNIVERSITY



**Project Title**:  Playing the Game of Flappy Bird with Deep Reinforcement Learning

**Group Number**: G-07

**Group** **Members**: Wang Wenqing 116032910080

Gao Xiaoning 116032910032

Qian Chen 116032910073

Contents

[1 Introduction 1](#_Toc484635508)

[2 Deep Q-learning Network 2](#_Toc484635509)

[2.1 Q-learning 2](#_Toc484635510)

[2.1.1 Reinforcement Learning Problem 2](#_Toc484635511)

[2.1.2 Q-learning Formulation [6] 3](#_Toc484635512)

[2.2 Deep Q-learning Network 4](#_Toc484635513)

[2.3 Input Pre-processing 5](#_Toc484635514)

[2.4 Experience Replay and Stability 5](#_Toc484635515)

[2.5 DQN Architecture and Algorithm 6](#_Toc484635516)

[3 Experiments 7](#_Toc484635517)

[3.1 Parameters Settings 7](#_Toc484635518)

[3.2 Results Analysis 9](#_Toc484635519)

[4 Conclusion 11](#_Toc484635520)

[5 References 12](#_Toc484635521)

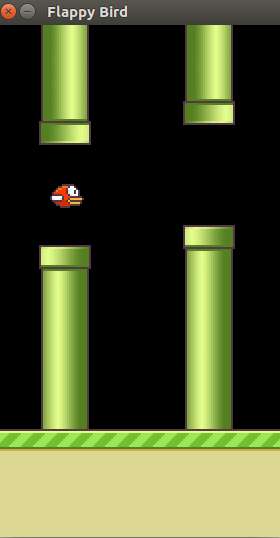
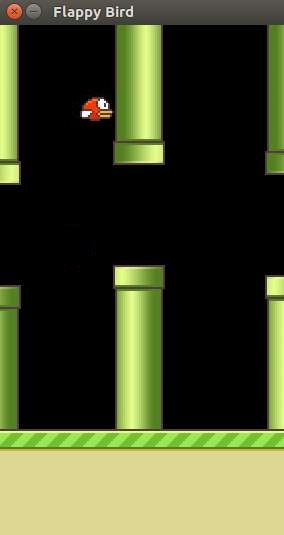
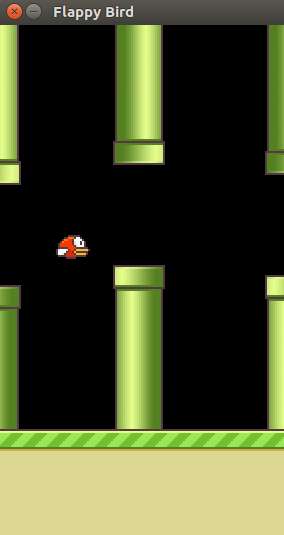
**Playing the Game of Flappy Bird with Deep Reinforcement Learning**

**Abstract**

Letting machine play games has been one of the popular topics in AI today. Using game theory and search algorithms to play games requires specific domain knowledge, lacking scalability. In this project, we utilize a convolutional neural network to represent the environment of games, updating its parameters with Q-learning, a reinforcement learning algorithm. We call this overall algorithm as deep reinforcement learning or Deep Q-learning Network(DQN). Moreover, we only use the raw images of the game of flappy bird as the input of DQN, which guarantees the scalability for other games. After training with some tricks, DQN can greatly outperform human beings.

# Introduction

Flappy bird is a popular game in the world recent years. The goal of players is guiding the bird on screen to pass the gap constructed by two pipes by tapping screen. If the player tap the screen, the bird will jump up, and if the player do nothing, the bird will fall down at a constant rate. The game will be over when the bird crash on pipes or ground, while the scores will be added one when the bird pass through the gap. In Figure1, there are three different state of bird. Figure 1 (a) represents the normal flight state, (b) represents the crash state, (c) represents the passing state.



(a) (b) (c)

**Figure 1**: (a) normal flight state (b) crash state (c) passing state

Our goal in this paper is to design an agent to play Flappy bird automatically with the same input comparing to human player, which means that we use raw images and rewards to teach our agent to learn how to play this game. Inspired by [1], we propose a deep reinforcement learning architecture to learn and play this game.

Recent years, a huge amount of work has been done on deep learning in computer vision [6]. Deep learning extracts high dimension features from raw images. Therefore, it is nature to ask whether the deep learning can be used in reinforcement learning. However, there are four challenges in using deep learning. Firstly, most successful deep learning applications to date have required large amounts of hand-labelled training data. RL algorithms, on the other hand, must be able to learn from a scalar reward signal that is frequently sparse, noisy and delayed. Secondly, the delay between actions and resulting rewards, which can be thousands of time steps long, seems particularly daunting when compared to the direct association between inputs and targets found in supervised learning. The third issue is that most deep learning algorithms assume the data samples to be independent, while in reinforcement learning one typically encounters sequences of highly correlated states. Furthermore, in RL the data distribution changes as the algorithm learns new behaviors, which can be problematic for deep learning methods that assume a fixed underlying distribution.

This paper will demonstrate that using Convolutional Neural Network (CNN) can overcome those challenge mentioned above and learn successful control polices from raw images data in the game Flappy bird. This network is trained with a variant of the Q-learning algorithm [6]. By using Deep Q-learning Network (DQN), we construct the agent to make right decisions on the game flappy bird barely according to consequent raw images.

# Deep Q-learning Network

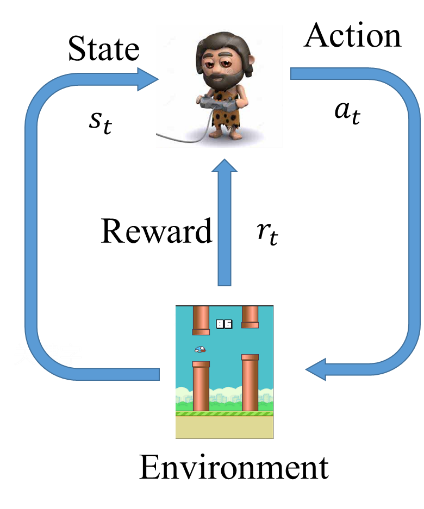
Recent breakthroughs in computer vision have relied on efficiently training deep neural networks on very large training sets. By feeding sufficient data into deep neural networks, it is often possible to learn better representations than handcrafted features [2][3]. These successes motivate us to connect a reinforcement learning algorithm to a deep neural network, which operates directly on raw images and efficiently update parameters by using stochastic gradient descent.

In the following section, we describe the Deep Q-learning Network algorithm (DQN) and how its model is parameterized.

## Q-learning

### Reinforcement Learning Problem

Q-learning is a specific algorithm of reinforcement learning (RL). As **Figure 2** show, an agent interacts with its environment in discrete time steps. At each time t, the agent receives an state  and a reward . It then chooses an action  from the set of actions available, which is subsequently sent to the environment. The environment moves to a new state and the reward  associated with the transition is determined [4].



**Figure 2**: Traditional Reinforcement Learning scenario

The goal of an agent is to collect as much reward as possible. The agent can choose any action as a function of the history and it can even randomize its action selection. Note that in order to act near optimally, the agent must reason about the long term consequences of its actions (i.e., maximize the future income), although the immediate reward associated with this might be negative [5].

### Q-learning Formulation [6]

In Q-learning problem, the set of states and actions, together with rules for transitioning from one state to another, make up a Markov decision process. One episode of this process (e.g. one game) forms a finite sequence of states, actions and rewards:



Hererepresents the state, is the action and is the reward after performing the action. The episode ends with terminal state. To perform well in the long-term, we need to take into account not only the immediate rewards, but also the future rewards we are going to get. Define the total future reward from time point t onward as:



In order to ensure the divergence and balance the immediate reward and future reward, total reward must use discounted future reward:



Here is the discount factor between 0 and 1, the more into the future the reward is, the less we take it into consideration. Transforming equation can get:



In Q-learning, define a function representing the maximum discounted future reward when we perform action in state:



It is called Q-function, because it represents the “quality” of a certain action in a given state. A good strategy for an agent would be to always choose an action that maximizes the discounted future reward:



Here π represents the policy, the rule how we choose an action in each state. Given a transition, equation can get following bellman equation - maximum future reward for this state and action is the immediate reward plus maximum future reward for the next state:



The only way to collect information about the environment is by interacting with it. Q-learning is the process of learning the optimal function, which is a table in. Here is the overall algorithm 1:

|  |
| --- |
| **Algorithm 1** Q-learning |
| Initialize Q[num\_states, num\_actions] arbitrarily  Observe initial state *s0*  **Repeat**  Select and carry out an action *a*  Observe reward r and new state *s’*    *s = s’*  **Until** terminated |

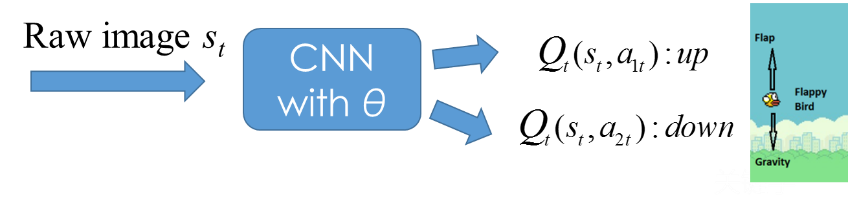
## Deep Q-learning Network

In Q-learning, the state space often is too big to be put into main memory. A game frame of  binary images has states, which is impossible to be represented by Q-table. What’s more, during training, encountering a known state, Q-learning just perform a random action, meaning that it’s not heuristic. In order overcome these two problems, just approximate the Q-table with a convolutional neural networks (CNN) [7][8]. This variation of Q-learning is called Deep Q-learning Network (DQN) [9][10].

After training the DQN, a multilayer neural networks can approach the traditional optimal Q-table as followed:



As for playing flappy bird, the screenshot *st* is inputted into the CNN, and the outputs are the Q-value of actions, as shown in **Figure 3**:



**Figure 3:** In DQN, CNN’s input is raw game image while its outputs are Q-values Q(s, a), one neuron corresponding to one action’s Q-value.

In order to update CNN’s weight, defining the cost function and gradient update function as [9][10]:





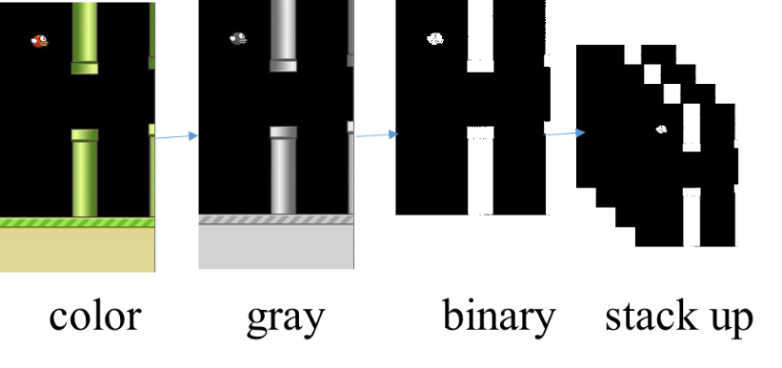


Here, are the DQN parameters that get trained and are non-updated parameters for the Q-value function. During training, use equation to update the weights of CNN.

Meanwhile, obtaining optimal reward in every episode requires the balance between exploring the environment and exploiting experience.-greedy approach can achieve this target. When training, select a random action with probability or otherwise choose the optimal action . Theanneals linearly to zero with increase in number of updates.

## Input Pre-processing

Working directly with raw game frames, which are pixel RGB images, can be computationally demanding, so we apply a basic preprocessing step aimed at reducing the input dimensionality.



**Figure 4:** Pre-process game frames. First convert frames to gray images, then down-sample them to specific size. Afterwards, convert them to binary images, finally stack up last 4 frames as a state.

In order to improve the accuracy of the convolutional network, the background of game was removed and substituted with a pure black image to remove noise. As **Figure 4** shows, the raw game frames are preprocessed by first converting their RGB representation to gray-scale and down-sampling it to an image. Then convert the gray image to binary image. In addition, stack up last 4 game frames as a state for CNN. The current frame is overlapped with the previous frames with slightly reduced intensities and the intensity reduces as we move farther away from the most recent frame. Thus, the input image will give good information on the trajectory on which the bird is currently in.

## Experience Replay and Stability

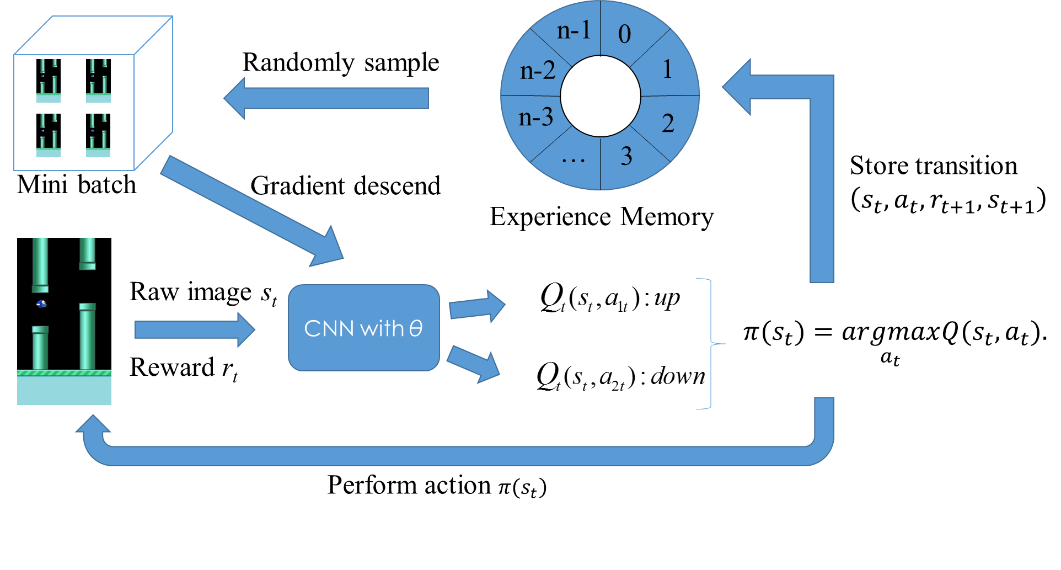
By now we can estimate the future reward in each state using Q-learning and approximate the Q-function using a convolutional neural network. But the approximation of Q-values using non-linear functions is not very stable. In Q-learning, the experiences recorded in a sequential manner are highly correlated. If sequentially use them to update the DQN parameters, the training process might stuck in a local minimal solution or diverge.

To ensure the stability of training of DQN, we use a technical trick called experience replay. During game playing, particular number of experience are stored in a replay memory. When training the network, random mini-batches from the replay memory are used instead of the most recent transition. This breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum. As a result of this randomness in the choice of the mini-batch, the data that goes in to update the DQN parameters are likely to be de-correlated.

Furthermore, to better the stability of the convergence of the loss functions, we use a clone of the DQN model with parameters. The parametersare updated to after every C updates to the DQN.

## DQN Architecture and Algorithm

As shown in **Figure 5**, firstly, get the flappy bird game frame, and after pre-processing described in section 2.3, stack up last 4 frames as a state. Input this state as raw images into the CNN whose output is the quality of specific action in given state., the agent performs an action According to policy , with probability , otherwise perform a random action. The current experience is stored in a replay memory, a random mini-batch of experiences are sampled from the memory and used to perform a gradient descent on the CNN’s parameters. This is an interactive process until some criteria are being satisfied.



**Figure 5:** DQN’s training architecture: upper data flow show the training process, while the lower data flow display the interactive process between the agent and environment.

The complete DQN training process is shown in Algorithm 2. We should note that the factor is set to zero during test, and while training we use a decaying value, balancing the exploration and exploitation.

|  |
| --- |
| **Algorithm 2** Deep Q-learning Network |
| Initialize replay memory D to certain capacity N  Initialize the CNN with random weights  Initialize =:  **for** games = 1: maxGames do  **for** snapShots = 1: T do  With probability select a random action *at*  otherwise select  Execute *at* and observe *rt+1* and next sate *st+1*  Store transition (*st* *,at* , *rt+1* , *st+1*) in replay memory D  Sample mini-batch of transitions from D  **for** j = 1: batchSize do  **if** game terminates at next state then  Q\_pred =: *rj*  **else**  Q\_pred =: *rj* +  **end** **if**  Perform gradient descent on  according to equation  **end** **for**  Every C steps reset =:  **end** **for**  **end** **for** |

# Experiments

This section will describe our algorithm’s parameters setting and the analysis of experiment results.

## Parameters Settings

**Figure 6** illustrates our CNN’s layers setting. The neural networks has 3 CNN hidden layers followed by 2 fully connected hidden layers. Table 1 show the detailed parameters of every layer. Here we just use a max pooling in the first CNN hidden layer. Also, we use the ReLU activation function to produce the neural output.

图片包含 文字

已生成高可信度的说明

**Figure 6:** The layer setting of CNN: this CNN has 3 convolutional layers followed by 2 fully connected layers. As for training, we use Adam optimizer to update the CNN’s parameters.

**Table 1:** The detailed layers setting of CNN

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Filter size** | **Stride** | **Num filters** | **Activation** | **Output** |
| conv1 | 80×80×4 | 8×8 | 4 | 32 | ReLU | 20×20×32 |
| max\_pool | 20×20×32 | 2×2 | 2 |  |  | 10×10×32 |
| conv2 | 10×10×32 | 4×4 | 2 | 64 | ReLU | 5×5×64 |
| conv3 | 5×5×64 | 3×3 | 1 | 64 | ReLU | 5×5×64 |
| fc4 | 5×5×64 |  |  | 512 | ReLU | 512 |
| fc5 | 512 |  |  | 2 | Linear | 2 |

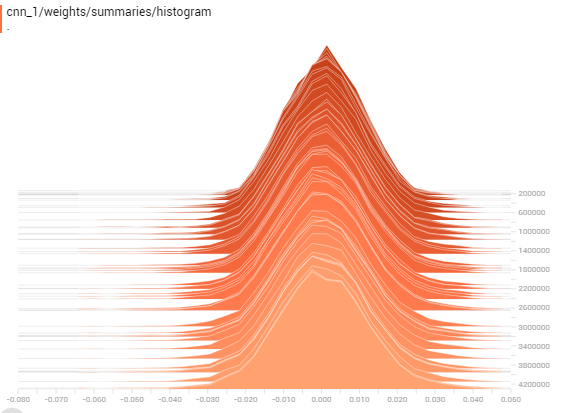
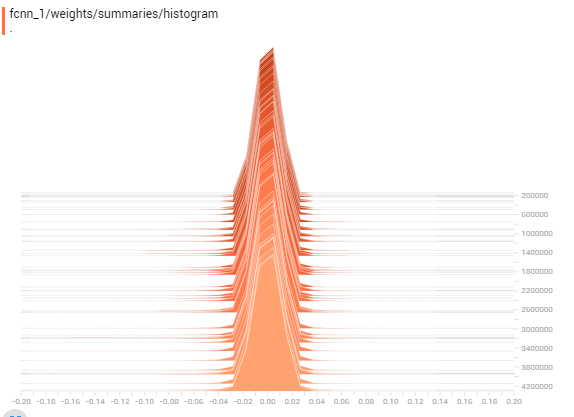
**Table 1** lists all the parameter setting of DQN. We use a decayed  ranging from 0.1 to 0.001 to balance exploration and exploitation. What’s more, **Table 2** shows that the batch stochastic gradient descent optimizer is Adam with batch size of 32. Finally, we also allocate a large replay memory.

**Table 2:** The training parameters of DQN

|  |  |
| --- | --- |
| **Parameters** | **value** |
| Observe steps | 100000 |
| Explore steps | 3000000 |
| Initial\_epsilon | 0.1 |
| Final\_epsilon | 0.001 |
| Replay\_memory | 50000 |
| batch size | 32 |
| learning rate | 0.000001 |
| FPS | 30 |
| optimization algorithm | Adam |

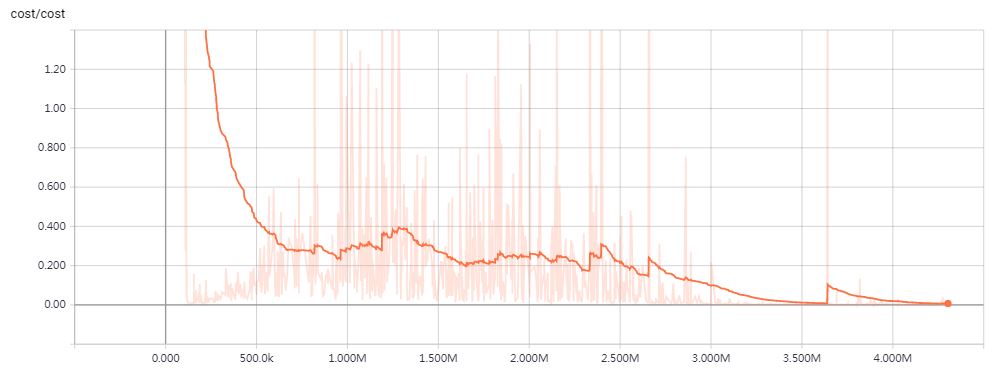
## Results Analysis

We train our model about 4 million epochs. **Figure 7** shows the weights and biases of CNN’s first hidden layer. The weights and biases finally centralize around 0, with low variance, which directly stabilize CNN’s output Q-value and reduce probability of random action. The stability of CNN’s parameters leads to obtaining optimal policy.

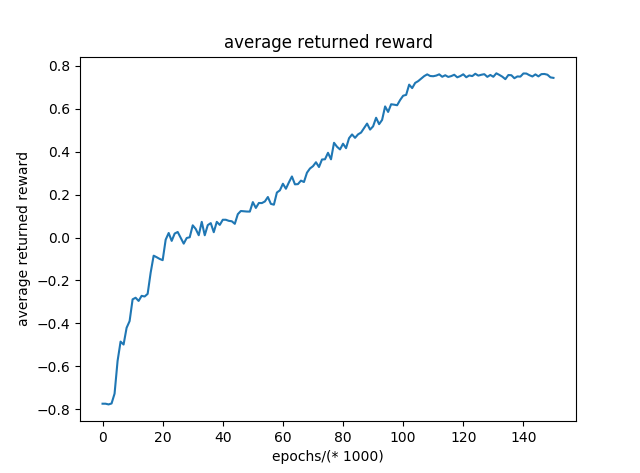
**Figure 7:** Left (right) figure is the histogram of weights (biases) of CNN’s first hidden layer

**Figure 8** is the cost value of DQN during training. The cost function has a slow downtrend, close to 0 after 3.5 million epochs. It means that DQN has learned the most common state subspace and will perform optimal action when coming across known state.In a word, DQN has obtained its best action policy.



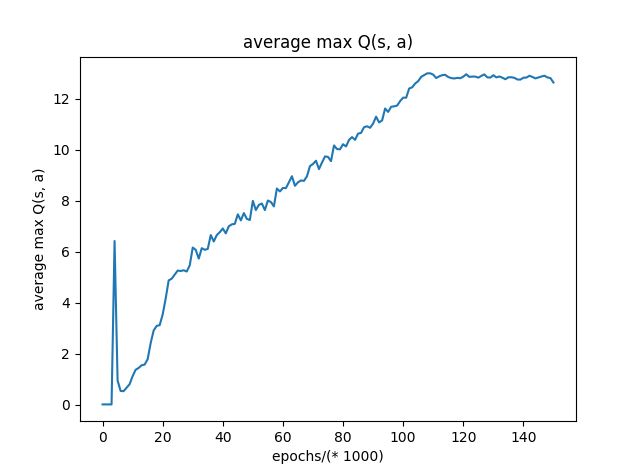
**Figure 8:** DQN’s cost function: the plot shows the training progress of DQN. We trained our model about 4 million epochs.

When playing flappy bird, if the bird gets through the pipe , we give a reward 1, if dead, give -1, otherwise 0.1. **Figure 9** is the average returned reward from environment. The stabiltiy in final training state means that the agent can automatically choose the best action, and the environment gives the best reward in turns. We know that the agent and environment has enter into a friendly interaction, guaranteeing the maximal total reward.



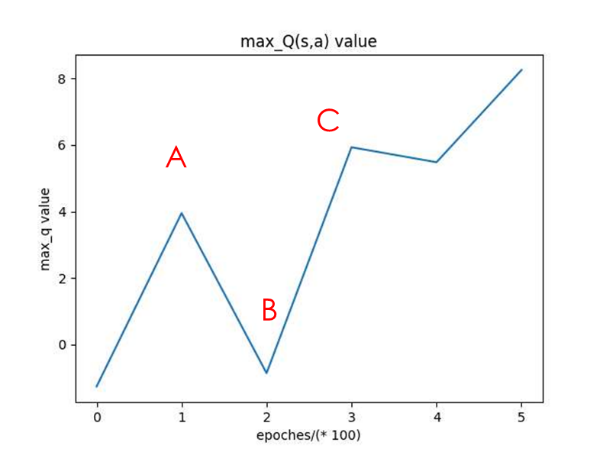
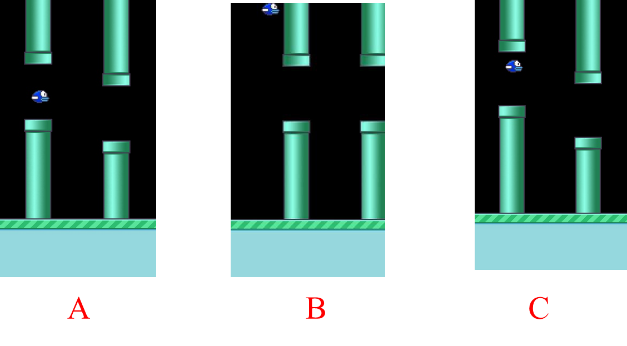
**Figure 9:**The average returned reward from environment. We average the returned reward every 1000 epochs.

From this **Figure 10**, the predicted max Q-value from CNN converges and stabilizes in a value after about 100 000. It means that CNN can accurately predict the quality of actions in specific state, and we can steadily perform actions with max Q-value. The convergence of max Q-values states that CNN has explored the state space widely and greatly approximated the environment well.



**Figure 10:** The average max Q-value obtained from CNN’s output. We average the max Q-value every 1000 epochs.

**Figure 11** illustrates the DQN’s action strategy. If the predicted max Q-value is so high, we are confident that we will get through the gap when perform the action with max Q-value like A, C. If the max Q-value is relatively low, and we perform the action, we might hit the pipe, like B. In the final state of training, the max Q-value is dramatically high, meaning that we are confident to get through the gaps if performing the actions with max Q-value.

**Figure 11:** The leftmost plot shows the CNN’s predicted max Q-value for a 100 frames segment of the game flappy bird. The three screenshots correspond to the frames labeled by A, B, and C respectively.

# Conclusion

We successfully use DQN to play flappy bird, which can outperform human beings.

DQN can automatically learn knowledge from environment just using raw image to play games without prior knowledge. This feature give DQN the power to play almost simple games. Moreover, the use of CNN as a function approximation allow DQN to deal with large environment which has almost infinite state space. Last but not least, CNN can also greatly represent feature space without handcrafted feature extraction reducing the massive manual work.

# References

1. C. Clark and A. Storkey. Teaching deep convolutional neural networks to play go. arXiv preprint arXiv:1412.3409, 2014. 1.
2. Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. *In Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.
3. George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30 –42, 2012, 1.
4. Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. *MIT Press*, 1998.
5. Brian Sallans and Geoffrey E. Hinton. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5:1063–1088, 2004.
6. Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
7. Hamid Maei, Csaba Szepesv´ari, Shalabh Bhatnagar, and Richard S. Sutton. Toward off-policy learning control with function approximation. *In Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pages 719–726, 2010.
8. Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. *In Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.
9. V.Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D.Wierstra, andM. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv*:1312.5602, 2013. 1.
10. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 3, 5.