Flappy Bird with Reinforcement Learning

Teng Li
lite@oregonstate.edu

Kang Li
likang@oregonstate.edu

I-Shen Liao
liaoi@oregonstate.edu

1 Introduction

Flappy bird is a 2D game. The object is controlling the flappy bird to pass through the gap between each pair of pipes. The player has two actions to control the flappy bird. Pressing "up" makes the bird flap up. Not pressing any key makes the bird fall down. In AI area, creating an agent to learn how to play a game is a popular topic. There are many different theories and algorithms are used to solve such problems or improving the performance. In this project, we are going to use various model-free learning algorithms to train an agent playing flappy bird and compare the performance of those learning algorithms including SRARA, Q-Learning, DQN and policy gradient. The goal of flappy bird is directing the flappy bird to pass through any pair of pipes and move as far as possible. We will use nave reinforcement learning to solve this problem. Also, we will use CNN to learn features from each frame of the game and train the agent to control the flappy bird correctly.

2 Model and Simulator

Markov decision-making process provides a mathematical framework to solve the problem that partly random and partly under the control of a decision maker. Markov decision-making process is a useful tool in the research field of solving optimization problems through dynamic planning and reinforcement learning. In this project, we use Markov decision-making process to train an agent to learn how to play the flappy bird. Instead of capturing image from game screen, we defined state, action, transition and reward.

The state contains distances between the pair of pipes and the flappy bird, and the vertical speed of the bird which has acceleration of gravity and maximum speed.

There are only two actions that pressing "up" makes the flappy bird jump up and pressing nothing makes the flappy bird fall down.

The transitions are the probability of the current state with the flappy bird's action lead to next state. The probabilities are always 1 since when the flappy bird takes an action, it always leads the flappy bird to a certain state.

The rewards are getting 0.1 point when the flappy bird move forward each frame, getting 1 point when the flappy bird passes through each pair of pipes and getting -1 when the flappy bird crash a pipe and die.

The vertical and horizontal distances between the flappy bird and pipes would change in each frame and the distance is based on the flappy birds action and the pipes location. When the flappy bird flaps up, the vertical distance is increased as the Figure 1.

We defined the downward direction is positive. The horizontal speed of the flappy bird is a constant 4, but the vertical speed of the flappy bird is not a constant. When the agent or player presses nothing, the flappy bird is going to fall down and the falling speed is increasing based on the gravity until reaching the max speed. We defined the acceleration of gravity is a constant 1. When the flappy bird fly up, the vertical speed of the flappy bird plus -9. We also defined the max speed of the flappy in up or downward directions are -9 and 10.

Every time when the flappy bird reaches a new state have not reached before, the new state with a reward would be updated to the saved network. After training the agent for a while, the agent is able to predict what the best action is in certain state.

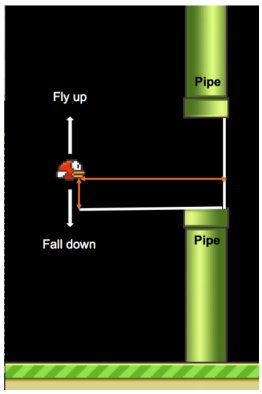

Figure 1: The Status of flappy bird

In deep neural network, each layer of nodes is trained based on the previous layer. The depth your neural net is, the more complex the features your nodes can recognize because each layer is trained based on the previous layer. Deep neural network allows machine learning models to directly extract raw sensory data from image. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. Each convolutional layer is optimized by the back-propagation. The goal of convolutional operation is to extract different features from each different layer. The first convolutional layer extracts the lower level feature, such as edges, lines and angles. More convolutional layers can extract more complex features from lower level convolutional layer. The goal of in this project is to develop a deep neural network from the raw pixel to learn the features and decide on what actions to take in the next state. Reinforcement learning is able to teach the agent to make right decisions under uncertain and high dimensions conditions by experiencing the scenarios.

3 Methods

This section describes how the models are parameterized and the Q-learning, SARSA, DQN, and Policy Gradient algorithm. We used reinforcement learning setting tries to evaluate the actions in a given state based on the reward it observes by executing it. For Q-learning and SARSA algorithms, we calculate the state and action then update on the extract images of game instances. For DQN and Policy Gradient we get the states from each frame.

3.1 SARSA

A SARSA agent interacts with the environment and updates the policy based on actions taken, hence this is known as an on-policy learning algorithm. The Q value for a state-action is updated by an error, adjusted by the learning rate alpha. Q values represent the possible reward received in the next time step for taking action a in the state s, plus the discounted future reward received from the next state-action observation [Wiering and Schmidhuber 1998].

SARSA learns the Q values associated with taking the policy it follows itself following an exploration/exploitation policy. SARSA is an on-policy learning algorithm which learns by trying to evaluate the policy being followed during learning. The main update is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \beta Q(s',a') - Q(s,a)) \quad (1)$$

3.2 Q-learning

In Q-learning, we define a function Q(s, a) representing the discounted future reward when we perform an action a in state s, and continue optimally from that point on. The way to think about Q(s, a) is that it is the best possible score at the end of the game after performing an action a in state s. Q function represents the quality of a certain action in given state [Matiisen 2015]. The main idea in Q-learning is that we can iteratively approximate the Q function using the Bellman equation. The main update is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \beta \max_{a' \in A} Q(s',a') - Q(s,a)) \quad (2)$$

Once we get the Q function, we pick each action with the highest Q value:

$$\pi(s) = \arg\min_{a}(s,a) \quad (3)$$

3.2.1 Q-learning 5 * 5 pixels block

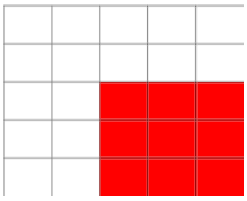

Figure 2: 5 * 5 pixels block

After comparison, we found Q-learning has better performance than SARSA from the result. Base on this result, we considered to improve Q-learning to find a better agent. We found that using Q-learning to train the model in each pixel is a little slow because there are too many states in the MDP. So, we considered reducing the quantity of the states to increase training efficiency. We calculated the distance between the bird and the pipe and mapped the states in a 5 * 5 pixels bulk. No matter how many pixels in this bulk, we considered this bulk is current state. In this algorithm, the quantity of states decreases and the training efficiency increase.

3.2.2 Q-learning 10 * 10 pixels block

Similarly, we calculated the distance between the bird and the pipe and mapped the states in a 10 * 10 pixels block. No matter how many pixels in this bulk, we considered this block is current state. In this situation, the quantity of states decreases even more than 5 * 5 pixels block, so the training efficiency increased.

3.2.3 Q-learning Double Pipes

To try to train better agent, we considered calculating the distance between the bird and two pairs of pipes. So, the agent may choose a better action when the bird is passing the first pair of pipes. It can help the bird pass the second pair of pipes easier. Although the quantity of the states is very large in this situation, the model may

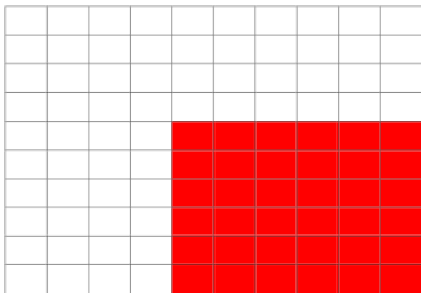

Figure 3: 10 * 10 pixels block

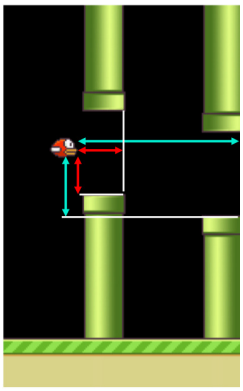

Figure 4: Double Pipes

choose a better action to easy to pass the second pair of the pipe when it is passing the first pair of pipes.

3.3 DQN

The DQN algorithm takes the whole image picture as input to predict the Q value for each action. To extract information from an image, we used the convolutional neural network which is similar to the architecture mentioned in Googles paper, two convolutional layers followed by two fully connected layers. The input is the image of the current frame, and the outputs are the Q values for all the actions. The agent will choose an action based on Q values calculated by the network. The algorithm will use the following loss function to update θ_i [Mnih et al. 2016].

$$L_i(\theta_i) = (r + \gamma \max_{a' \in A} Q(s',a' : (\theta_i^-)) - Q(s,a : \theta_i))^2 \quad (4)$$

3.3.1 Preprocess

Before the images get processed, they will be scaled down from 228 * 512 to 84 * 84 and converted to grayscale, which can speed up the computation and save memory space. In other words, the image of a frame will be preprocessed to the same size and with only one channel, 84 * 84 * 1. To get temporal information, the algorithm will stack the most recent 4 frames as the input. Therefore, after the preprocessing stage, the input of the network is a tensor with the size of 84 * 84 * 4.

3.3.2 Network architecture

The first hidden layer convolves 32 filters of 8*8 with stride 4 with the input image and applies a rectifier nonlinearity. The second

hidden layer convolves 64 filters of 4 * 4 with stride 2, again followed by a rectifier nonlinearity. The final hidden layer is fully-connected and consists of 256 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action.

3.3.3 Other parameters

To enable the agent to explore, in the project we use ε-greedy with fixed to 0.0001. The γ is set to 0.99, and the batch size is set to 32. The reply memory size is set to 50,000 which could store the most recent 50,000 frames state information. The model parameter of the target network will be synchronized every 3,000 steps.

3.4 Policy Gradient

Instead of learning the Q value, the policy gradient algorithm learns the policy directly. We also applied the policy gradient algorithm in our project. We use the estimated value of each state as the baseline to implement an actor-critic algorithm to reduce the variance of gradient algorithms. Thus it can speed up the training process. The policy is updated according to the policy gradient theorem.

$$\nabla_\theta \rho(\theta) = E_{\pi\theta}[Q_{\pi\theta}(S,A)\nabla_\theta log\pi_\theta(S,A)] \tag{5}$$

The architecture of the network used in the actor-critic algorithm is similar as the neural network architecture used in DQN, two convolutional layers followed by two fully connected layers, and the parameters of these layers are identical. Unlike DQN, the actor-critic algorithm has two independent networks; one is used to predict the action, and the other is used to estimate the value of the state. Another difference is that the output layer of the policy network is a softmax layer which generates probabilities over all actions. The valid value of each possibility is from 1e-20 to 1.0. Therefore, the network does not need the -greedy function to explore states.

4 Experiments and Results

4.1 SARSA

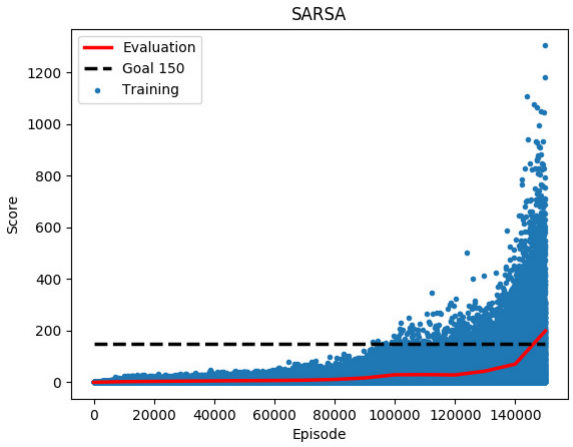

Figure 5: Results of SARSA

The episode is the bird crash base, ceiling or the pipe from the beginning. Such as illustrated in Figure 5, the blue dot is the score for each episode in training the model. We recorded the q values in every 10,000 episodes. The red curve is the all average of 100 times evaluation for all recorded q values. We found the model only can pass few pipes at the beginning 80,000 episodes. Moreover, the score increased rapidly after training 140,000 episodes. And the model can achieve the goal, passing 150 pipes in 150,000 episodes.

4.2 Q-learning

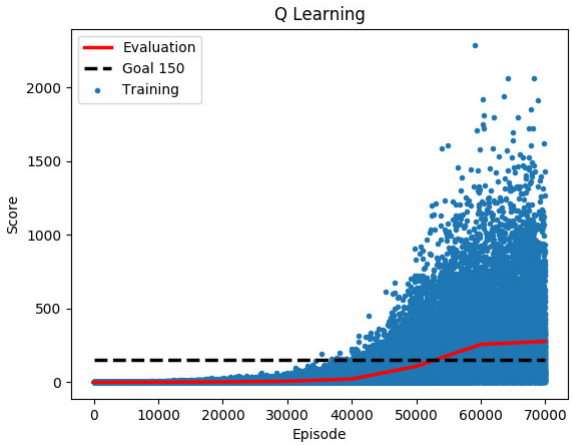

Figure 6: Results of Q-learning

Such as illustrated in Figure 6, similarly the blue dot is the score for each episode in training the model. We recorded the q values in every 10000 episodes. The red curve is the all average of 100 times evaluation for all recorded q values. We found the model only can pass few pipes at the beginning 40,000 episodes. Moreover, the score increased rapidly after training 40,000 episodes. And the model can achieve the goal, passing 150 pipes in 50,000 episodes.

Comparing the results of SARSA and Q-learning, we found Q-learning has better training efficiency. We decided to adjust parameter in Q-learning.

4.2.1 Q-learning 5 * 5 pixels block

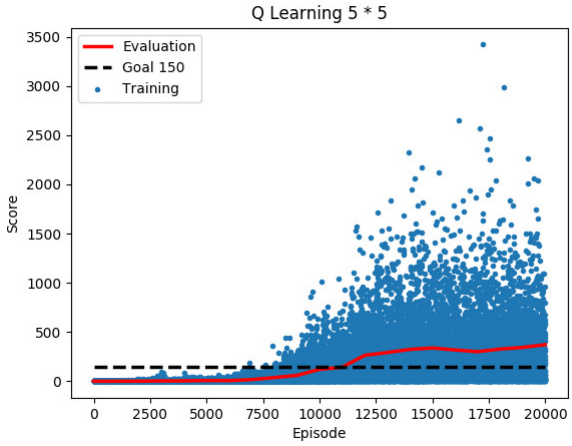

Figure 7: Results of Q-learning 5 * 5 pixels block

Such as illustrated in Figure 7, similarly the blue dot is the score

for each episode in training the model. We recorded the q values in every 1,000 episodes. The red curve is the all average of 100 times evaluation for all recorded q values. We found the model only can pass few pipes at the beginning 5,000 episodes. Moreover, the score increased rapidly after training 7,000 episodes. And the model can achieve the goal, passing 150 pipes in 10,000 episodes.

4.2.2 Q-learning 10 * 10 pixels block

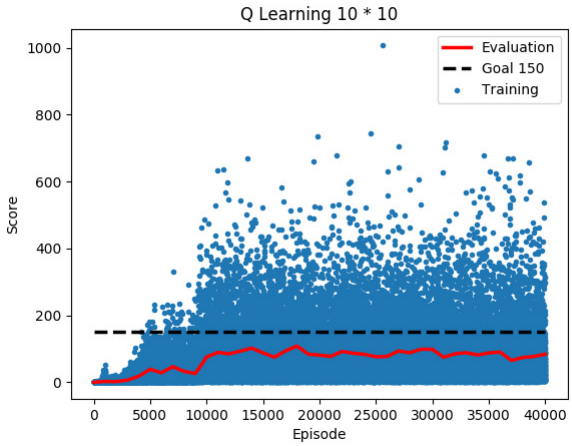

Figure 8: Results of Q-learning 10 * 10 pixels block

Such as illustrated in Figure 8, similarly the blue dot is the score for each episode in training the model. We recorded the q values in every 1,000 episodes. The red curve is the all average of 100 times evaluation for all recorded q values. We found the model can get a better score very quickly. However, the average score is stable at 80 for the rest of episodes.

4.2.3 Q-learning Double Pipes

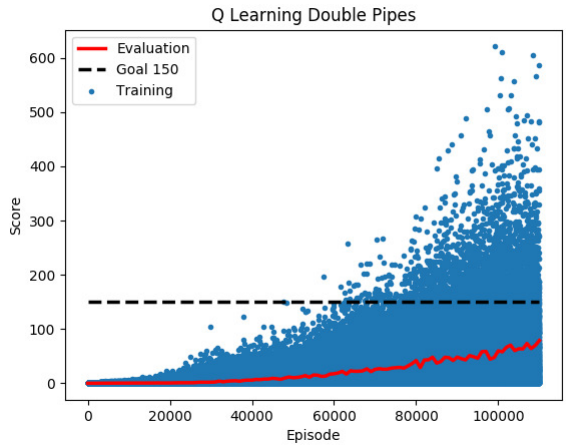

Figure 9: Results of Double Pipes

Such as illustrated in Figure 9, similarly the blue dot is the score for each episode in training the model. We recorded the q values in every 1,000 episodes. The red curve is the all average of 100 times evaluation for all recorded q values. We found the score increased very slowly and smoothly. However, to train this model need a lot of time, we cannot train too many episodes in limited time.

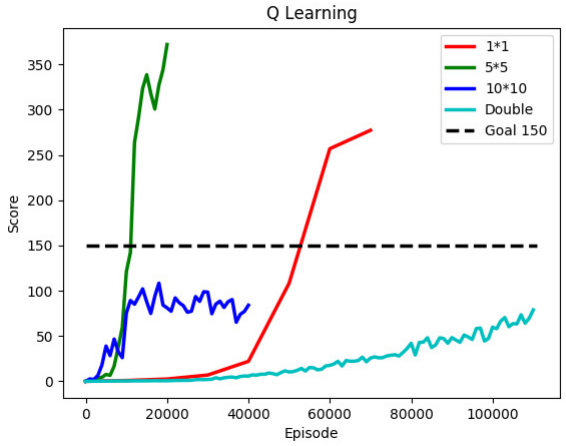

Figure 10: Results of improve Q-learning

Comparing the results of these training model, we found Q-learning 10 * 10 can get a better score at the beginning, such as illustrated in Figure 10. However, Q-learning cannot get a good score in the rest training. The score stays at 80 more or less.

Comparing the results of these training model, we found Q-learning 5 * 5 can get a best score, although it cannot get better score than Q-learning 10 * 10 at the beginning, such as illustrated in Figure 10.

Comparing the results of these training model, we found Q-learning considered double pair of pipes can get a good score with the training episode increasing, such as illustrated in Figure 10.

4.3 DQN

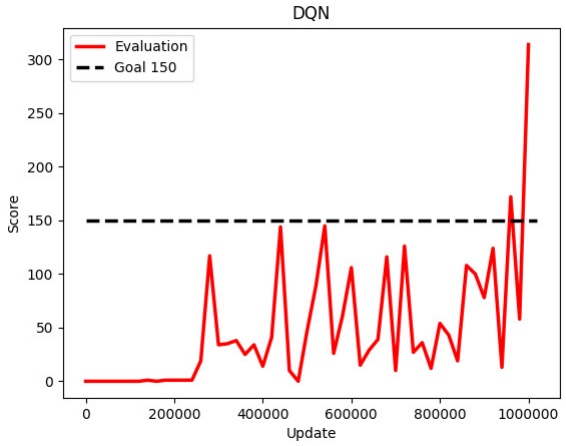

Figure 11: Results of DQN

The agent can obtain a reasonable performance after 500,000 frames training, like passing around 100 pipes which is better than human performance. After 1 million update steps, it could pass more than 300 pipes after, which is way better than the goal we set

in the proposal. The performance is measured by using each model saved per 20,000 frames to run 5 episodes and average the result.

The result is not very stable due to the randomness of the game that the position and shape of the pipe are generated randomly. Another reason is the DQN model does not get well trained. In our case, the agent was trained by 1 million frames which are significantly less than 50 million frames mentioned in Googles paper. The 1 million update steps are equivalent to around 4,100 episodes.

We also tried to remove the target network to reduce the model complexity of the algorithm. However, even it still works, the training process is slow down significantly. After 1 million frames, the agent can only pass no more than 10 pipes.

The highest score for DQN algorithm is around 700 which is obtained after 1 million frames. The score is not an average score shown in Figure 11.

4.4 Policy Gradient

The actor-critic algorithm did not work in the beginning. The agent seems to converge to a suboptimal point after a period of training, that the bird will fly directly into the ceiling until it collides with the first pipe. We tried to use the k-step algorithm rather than the 1-step algorithm to compute the temporal difference of the estimated value, but it did not make much difference. We also used two independent neural networks to do policy and value computation individually and introduced the entropy loss term [Mnih et al. 2015] into the loss function. However, they did not work either.

Finally, we had to modify the game rule a little bit, such as adding the negative reward for touching the ceiling. Ultimately, the agent can get out of the suboptimal point after the modification. However, the training process is way slower than the training process of DQN. The agent can pass only 8 pipes after 1 million frames. We also could manage to speed up the training process by removing the randomness of the game in which the pipe will appear in a fixed position with a fixed shape.

5 Conclusion and Future Work

In conclusion, all algorithms are able to train an agent to play the game with different performance and training speed. The DQN algorithm has the best performance among all the algorithms and fastest training process. However, since SARSA and Q-learning do not need to spend any time on processing image during the training process, these two algorithms might get trained faster than DQN.

According to Googles paper, the model could get well trained after 50 million frames with a replay memory of 1 million most recent frames, which is around 38 days of game experience [Mnih et al. 2016]. However, due to the limitation of the computational resource and training time, our model can not get trained well to get the best performance. Therefore, we will try to train our model further in the future to try to figure out how well it can be.

Moreover, we will try to use more parameter combinations to figure out the best training parameters for the game. We also will try to modify the neural network architecture, such as adding max-pooling layers between convolutional layers, to find out whether the architecture complexity could affect the performance of the network.

Finally, to speed up the actor-critic algorithm training process, we could implement the A3C algorithm. The algorithm utilizes the multi-thread method, which enables the agent to get trained simultaneously. Therefore, it will significantly reduce the training time of the policy gradient algorithm. We also could try to run the algorithm on the GPU instead of the CPU to improve the training efficiency to some extent.

Contribution of the Authors

- Teng Li implements SARSA, Q-learning and improved Q-learning algorithms.
- Kang Li implements DQN and policy gradient algorithms.
- I-Shen Liao assists in implementing DQN and policy gradient algorithms.

References

MATIISEN, T. 2015. Demystifying deep reinforcement learning.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., PETERSEN, S., BEATTIE, C., SADIK, A., ANTONOGLOU, I., KING, H., KUMARAN, D., WIERSTRA, D., LEGG, S., AND HASSABIS, D. 2015. Human-level control through deep reinforcement learning.

MNIH, V., BADIA, A. P., MIRZA, M., GRAVES, A., HARLEY, T., LILLICRAP, T. P., SILVER, D., AND KAVUKCUOGLU, K. 2016. Asynchronous methods for deep reinforcement learning.

WIERING, M., AND SCHMIDHUBER, J. 1998. Fast online q. *Machine Learning 33*, 1, 105–115.