Combining Q-Learning with Artificial Neural Networks in Flappy Bird
Project Report of COMP 599

Jianan Yue
McGill University
Montreal, Quebec H3A0C3
Email: jianan.yue@mail.mcgill.ca

Yulin Shi
McGill University
Montreal, Quebec H3A0C3
Email: yulin.shi@mail.mcgill.ca

***Abstract*—For the flappy bird game, a typical delay control problem, we propose using the reinforcement learning algorithms to learn the control policy. Specifically, we use the Bellman equation to propagate the end-point reward to the entire state space; use model free Q-function to search optimal policy; and, to deal with the curse of dimension, use the neural network to parametrise the approximate function of the optimal controller. The Flappy Bird environment is developed in the JavaScript. The open source machine learning library Convnetjs is included to train the neural network. It is validated that the neural network Q-learning is proper for the learning problem. It is also observed that some strategies emerged after training the simple neural network.**

1. Introduction

1.1. Flappy Bird Problem

The well-known game Flappy bird, in a functional perspective of view, is a simulation world consists with a part of deterministic dynamic: the acceleration and a part of stochastic dynamic: the position of the gap. By observing the location of the path and estimating the trajectory of the bird in a continuous time, the players selects among two input commands. The object is to go through as many gaps as possible while avoiding colliding with the pipes. The game can be described in discrete time as a Markov process.

$$s' \sim \varepsilon(s, a) \tag{1}$$

where s is the current state, a is the current action, s' is the next state. Symbol $\sim$ mean the value of s' is subject to a distribution.

In this view, it is feasible to use an artificial agent following a policy (2) such as the ϵ-greedy policy to replace human as the player to play the Flappy Bird game. [**?**]

$$a = \pi(s) \tag{2}$$

1.2. Reinforcement Learning

One difficulty of learning the Flappy Bird problem is that the leaner agent does not perceive a prompt reward after taking an action. Rather, they get the reward at the end of an episode which is defined as the entire state-action trajectory of a round of play. At the end of an episode, the reward is proportional to the number of pipes the bird goes through. The agent is trained to learn a policy that maximize the overall reward. However, the strong delayed reward in the Flappy Bird problem invalids the classical machine learning algorithms such as the regression and classification which assumes that the samples are independent. For example, the agent does not perceive any reward or penalty when it does not choose the jump action and stays far away from the gap, even if these actions and states have a strong negative influence on the subsequent rewards. Therefore, we need an index that reflect the expected utility of a state, other than the instantaneous reward. We call the index as the value function

$$V^{\pi}(s) = E_{\pi}\left\{R_t | s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{3}$$

where $0 < \gamma \leq 1$ is the discount factor.

For the Flappy Bird problem as a Markov process, the definition of the (3) can be reformulated as the Bellman equation (4) and solved using episode samples iteratively.

$$V^{\pi}(s) = E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\right\} \tag{4}$$

For a time-invariant system with smooth value function, the iterative equation (3) is a contraction map. The value function will converge to its fixed point.

1.3. Q-learning

Another issue arising is that while the value function alone does not tell the agent how to act without a model (5) which can be very complex in real environments and usually suffers greatly from poor accuracy. [**?**]

$$\hat{s}' = \hat{\varepsilon}(s, a) \tag{5}$$

Since the Flappy Bird problem is basically a control problem, it is worthy of considering a model-free approach. Here we first define a Q function with respect to state and action

$$Q^{\pi}(s, a) = E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\right\} \tag{6}$$

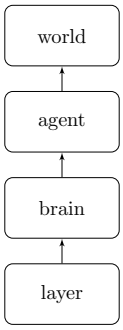

Figure 1. Software Architecture

Given the value of a Q function, it is possible to generate an optimal policy without a model

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s, a) \tag{7}$$

There are basically two approaches of learning the Q function given episode samples: off-policy (8) where actions are chosen greedily and on-policy (SARSA) (9) where actions are chosen based on the policy.

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \tag{8}$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma Q(\pi(s, a), a') - Q(s, a) \right) \tag{9}$$

1.4. Artificial Neural Network

For low dimensional problem, is it feasible to use tables to describe the value functions of discrete states. However, for large dimensional problems like the Flappy Bird problem, due to the curse of dimension, the table of the value function can be very large, which makes it impractical to maintain and retrieve. In this case, a good alternative might be to use parametrized function to approximate the policy function.

Function approximation falls into two categories: linear function approximation and nonlinear function approximation. In a linear function approximation, the value of function is the linear combination of a set of nonlinear feature functions like the polynomial function, Gaussian function and other customized functions. Good understanding of the model is needed for choosing good feature functions. Nonlinear function approximation includes search tree, neural network etc. They are found to be able to learn complex problems adaptively. Therefore, in this project, we choose the neural network to parameterize the policy function.

2. Software Architecture

For the simulation and learning task of the project, a three level object-oriented programming architecture is designed in JavaScript: the "World", the "Agent", the "Brain (neural network)" and the "Layer". A world supports stochastic environment simulations. It calls an agent object and sends it the partially observed states. The relationship is in Figure 1.

In each time step, the world and the agent is updated according to the action and the previous state. the agent calls a brain, passes the state and reward to the brain and obtain the action command from the brain. The brain calls the layers for Q-function values.

2.1. Layer

There are different types of layers. Each uses a different excitation function like sigmoid functions, tanh functions, relu functions and maxout functions. The number of neurons can be customized by users during layer construction.

The connected layers are not expected to be able to select from a set of admissible actions for the agent directly. In stead, they together estimate the value of the Q-function whose value will be passed to the brain. Therefore, the output layer becomes special because it produces the value function while the output value of intermediate layers are meaningless. The common feature of all layers is that they all define "forward" functions which make prediction and "backward" function which updates layer parameters.

2.2. Brain

A "Brain" class has layer attributes. The users define the layers depth, layer types and the initial parameters of the neuron layers. Actions of an agent is selected in the "forward" function of a brain. It either chooses among all actions the one corresponds to the best value, or in a small probability, chooses an action in a random distribution. The later is called the ϵ-greedy policy. The ϵ-greedy policy allows the agent to explore the environment.

2.3. Agent

An agent object (which is a bird in the game) stores its current state. It update its height, distance and velocity states according to the Newton's law and the action. It serves as the deterministic part of the environment. Each agent has a brain. We define distance to the gap as heuristic rewards in addition to the final rewards, for helping boosting the training speed.

2.4. World

The stochastic environment is created in the "World" class. It creates a new pipe when the "bird" flies through a pair of pipe. The heigh of the gap of the pipe is chosen in random, which serves as the unknown part of the environment. Collision is detected in the world and passed to the agent.

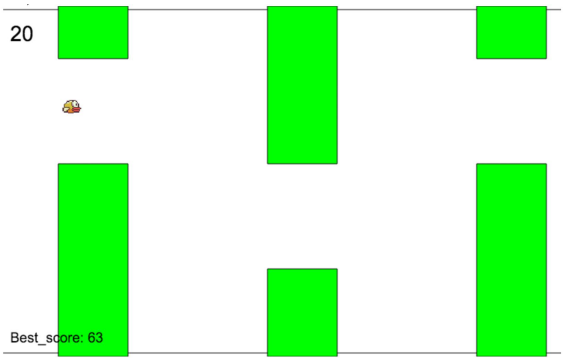

Figure 2. Flappy Bird interface

3. Experiments

3.1. Settings

All of the functions are written in JavaScript which runs in browsers. They are called by an html file.

There are three special design characteristics which make the web page interface interactive, which is similar to the "rldemo" project of "convnetjs". First, there is a text area where users edit the default settings of neural network layers and training parameters. Second, there is a canvas like Figure 2 where the the Flappy bird animation and the estimated performance are displayed and updated in real time. Herein, the users are allowed to control the speed of the game. Third, using JSON to create and an I/O windows allowing users to save the current neural network to a file or to load the well-trained network. [add the URL of the file].

A four-layer neural network is chosen in this experiment for training the agent. The two inner layers are fully connected layers, each has 30 neurons. It is also found that 5 neurons is an enough choice of the resolution of the neural network in terms of a policy function.

To avoid stalling in a local optimal policy, the training process starts from an random exploration process caused burning steps. The length of the burning steps is chosen as 3,000 time steps. In the burning steps, the action is chosen according to a distribution of: 5% in choosing tapping action, 95% in choosing resting. This probability is chosen carefully as to keep the "bird" flying admits the middle way of the passage, rather than hitting the wall easily before exploring the potential "successful" states. After burning steps, it uses the ϵ-greedy policy. The value of ϵ is decreasing with respect of time. The learning rate γ in (3) is chosen as 0.001.

3.2. Results and Discussions

As mentioned in the Software Architecture section, an layer object of an agent produces the performance index which can be used to choose greedy actions. Here, we use the performance index to denote the training performance. We plot the scalar performance index as a function of

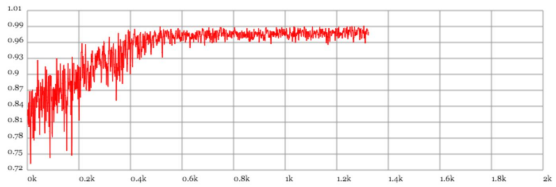

Figure 3. performance Curve

training steps in Figure 3. It shows that the value of the performance index keeps on oscillating. But the trend is that the value of performance index is increasing from the initial value of 0.8. After 500 round of training episodes, the value stabilizes at 0.97.

What we observe in the animation canvas is that the "bird" seldom go through the first pipe pair at the very beginning. But after 2 hours of training, it gets able to go through multiple pipe pairs. In fact, if put the ϵ value of the ϵ-greedy to be zero, the bird is expected to go through infinite pipe pairs.

Another observation is that, without teaching the agent explicitly about operation strategy. Some strategies which elongate life time arise from the simple Q-learning mechanics. For example, after getting through the gap a pipe pair, the bird move quickly to the same height of the gap of the next pipe pair. We do not yet know why the Q-learning resulted in this strategy, but it does increase the expected "success" rate.

4. Conclusion

Without designing an explicit automatic controller for playing the "Flappy Bird" game, we chose using the Q-learning mechanism to train an neural network based agent. The agent observe the real-time physical variables of the uncertain game world. After a finite time of training using Bellman iteration, the agent is able to generate controller commands which increases the expected overall rewards. What is more, a game strategy emerged as a result of the simple learning mechanism. The game simulation and the training process is written in JavaScript. It runs on browser without calling for GPU. The software is developed based on API provided by the "convnetjs" website. The result shows that, after a finite time of training, the performance indexed by the neural network output is improved. Also, the "bird" is able to go through more and more pipe pairs which is the aim of this project.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.