华东理工大学20_20_—20_21_学年第_春_学期

《 Introduction to Engineering 》课程论文

2021.6

班级（Class） 2nd-year Computer Science of Technology Major

学号(ID)20010026 姓名(Name)阿吉兹

开课学院(School)_信息学院 任课教师(Teacher)应方立成绩(Mark)____

论文题目（Application of Reinforcement Learning in Video games）:
论文要求(Requirements)： This essay in computer science provides students with an opportunity to investigate a particular aspect of engineering issues in Computer Science. Within this context, we can research the latest developments and future possibilities in a rapidly changing subject in Artificial Intelligence and its rea-life engineering applications. There are many possible areas to be explored, each with a wealth of topics such as advances in deep learning development, comparison of the efficiency of algorithms for machine learning, automatic gradient descent learning, recurrent neural network and so on. You can choose any related topics in the AI or ML areas, and it can be either from the contents in classes or is from the third-party academic resources like international conference or journal papers. In the essay, the details of implementation on the topic should be demonstrated as the main contents. There are a few requirements for writing this essay as below: 1. You can follow the format template in the second page(MS word template only) to write the essay and please delete the template page after finishing the writing of this essay. 2. An abstract and some keywords are necessary for highlighting your work in the beginning 3. Section 1 Introduction should include the subject of background, main contributions and the implications for society. 4. Section 2 Related works should show the literature reviews or the contents you reviewed from the third-party resources and how it inspires your work 5. Section 3 Framework or Method demonstrates your development of the application or the framework of neural network or ML methods 6. Section 4 Evaluations should include how your application can run the script and trained to solve the real-world problems based on Machine Learning or Deep learning process 7. Section 5 Conclusion explains what you have learned in the class and how those methods can apply to your current work
教师评语： 教师签字： 2021 年 6 月

Acknowledgments

First and foremost, I would like to express my heartfelt gratitude to my professor, Mr. 应方立, for his unwavering support and guidance during this project, this course, as well as his patience and extraordinary compassion. I truly appreciate his guidance which assisted me in completing this essay.

Abstract

Reinforcement learning (RL) in video games has reached a level of accuracy that yields dramatic results. Videos games are a massive field for testing the capabilities of RL and the developers are using it to its full extent. In this essay, I will be discussing the implementation of RL in video games with the example of the Pong game.

Contents

Chapter 1 Introduction

1.1 History of Machine Learning

Machine learning's origin in its contemporary sense is associated with the name of the American psychologist notable in the field of artificial intelligence Frank Rosenblatt (July 11, 1928 – July 11, 1971) also called the father of deep learning. He invented a group that set up the machine in order to identify the alphabetical order of code letters Rosenblatt(1957,1959,1960) which is based on ideas about the work of the human nervous system.

The first boom of machine learning was in the 1960s, many groups were involved in the design and testing of learning recognition systems such as Widrow(1961), Bongard(1960), and Braverman (1962). In 1967, Marcello Pelilo invented the "nearest neighbor rule" which is an algorithm used to find the most efficient route for a traveling salesperson.

1.2 History of Reinforcement Learning

The history of RL is divided into two sub-field which were merged afterward the first one is trial and error which started with the psychology of animal learning. The second one is the problem with optimal control.

RL is one of the most promising directions to actually get to very intelligent robotic behavior so in the most common machine learning applications people use what we call supervised learning and this means that you give input to your neural network model but you know the output that your model should produce and therefore you can compute gradients using back propagation algorithm to train that network to produce your outputs.

Chapter 2 Related Works

2.1 Introducing OpenAI gym library

OpenAI was founded in late 2015 as a non-profit with a mission to “build safe artificial general intelligence (AGI) and ensure AGI’s benefits are as widely and evenly distributed as possible.” In addition to exploring many issues regarding AGI, one major contribution that OpenAI made to the machine learning world was developing both the Gym and Universe software platforms.

Gym is a collection of environments/problems designed for testing and developing reinforcement learning algorithms—it saves the user from having to create complicated environments. Gym is written in Python, and there are multiple environments such as robot simulations or Atari games. There is also an online leaderboard for people to compare results and code.

2.2 Some examples from gym library

Gym library provides us with a lot of retro games from Atari (is a third-party company for Nintendo that was the leader of the video game market from 1975 to the early 1980s) and in this subsection, we are going to look briefly through some of the games that we can use to implement RL and afterward we will explain in detail the game of Pong.

2.2.3 Implementing RL in solving Taxi game

Taxi is one of many environments available on OpenAI Gym. The goal of a Taxi is to pick up passengers and drop them off at the destination in the least amount of moves.
To implement RL in this game we need to make an agent that takes actions randomly and then train it to be a better taxi driver.

The first step is to give the agent an initial state to know its current position in the environment and then we let it randomly go through it since it doesn’t know which is the best route we will assign a reward system for. The agent gets positive feedback when it successfully drops a passenger. After a few iterations, the agent will notice that there are some routes that have better rewards. So it will keep track of which actions led to those rewards and keep repeating them.

2.2.4 Implementing RL in solving Montezuma's Revenge

The goal of the agent in this game is to navigate a bunch of ladders jump over a skull grab a key and then actually navigate to the door in order to get to the next level.

The problem with this game is the actions for the agent to get a reward are very complicated it's never going to get there with random actions, as a result, the policy gradient is never going to see a single positive reward so it has no idea of what to do.

For us to solve this issue of sparse rewards is the use of rewards shaping. It is the process of manually designing a reward function that needs to guide our policy to some desired behavior in the case of Montezuma's revenge, for example, we could give our agent a reward every single time it manages to avoid the skull or reach the key and these extra rewards will guide it to some being better at the game.

2.2.5 Pong Game

The reason behind choosing this game is that I used to play this game with my sister when we were young and seeing how I can apply the methods I learned in machine learning is so fascinating. The main difference between this game and the others in the gym library or other games is the process is easier because the concept of the game is not as complicated. Pong is one of the first computer games ever created, this simple "tennis-like" game features two paddles and a ball, the goal is to defeat your opponent by being the first one to gain 20 points, a player gets a point once the opponent misses a ball. With this example, we can do the same as the other games of the reward system and we don't have to worry if the task is too complicated for the agent. We can use random actions like going up and down and then get the output and of course, we are going into much details about this later on in the essay.

Chapter 3 Reinforcement Learning

3.1 Introduction

All along with this chapter I will present a Reinforcement Learning by citing its definition and its implementation.

3.2 Reinforcement Learning

3.2.1 Definition

The reinforcement learning technique involves learning from the interaction of the model and its environment. This interaction does not involve a supervisor; rather, it is a trade-off between exploitation and exploration. Clearly defined, this interaction is analogous to children performing actions and being rewarded if the action is "good," and experienced without a teacher's assistance.
There are two kinds of reinforcement: positive reinforcement and negative reinforcement:

• **Positive Reinforcement:** Occurs when the strength and frequency of an event increase as a result of a specific behavior. It signifies that there is a positive influence on the behavior.

• **Negative Reinforcement:** This is a process in which the system avoids unfavorable conditions, and the specific behavior becomes stronger.

3.2.2 Implementation of RL in Pong game

We want to train a neural network to play the game of Pong what we would do in a supervised setting is we would have a good human gamer play the game of pong for a couple of hours and we would create a data set where we log all of the frames that that human is seeing on the screen as well as the actions that he takes in response to those frames so whatever is pushing the up or down arrow and then we can feed those input frames through a very simple neural network that at the output can produce two simple actions it's either going to select the up action or the down action and by simply training on the data set of the human gameplay using back propagation we can actually train that neural network to replicate the actions of the human gamer but there are two significant downsides to this approach. On one hand, in supervised learning, we have to create a data set to train on which is not always a very easy thing to do, and on the other hand, if you train your neural network model to simply imitate the actions of the human player well then by definition the agent can never surpass the human. So if we want to train the agent to be better at the game by itself we should use RL.

The framework of reinforcement learning is very similar to the normal framework in supervised learning so we still have an input frame we run it through some neural network model and the network produces an output action we either up or down but the only difference here is that now we don't actually know the target label so we don't know in any situation whether we should have gone up or down because we don't have a data set to train on.

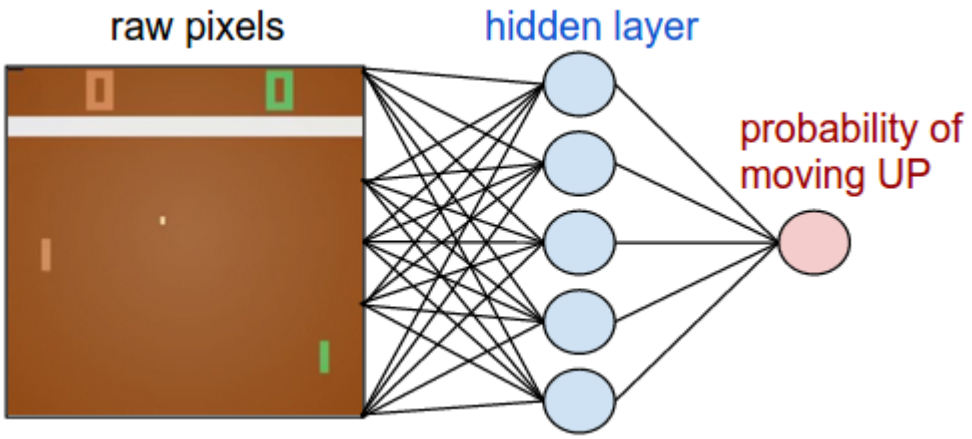

Figure 3-2-2-1 Neural Network

In reinforcement learning, the network that transforms input frames into output actions is called the policy Network. The simplest way to train a policy network is a method called policy gradients. The approach in policy gradients(as shown in the graph down below) is that we start out with a completely random network we feed that network a frame from the game engine, and it produces a random up with action we know either up or down we send that action back to the game engine and the game engine produces the next frame and this is how the loop continues and the network, in this case, it could be a fully connected network. To further explain we take the state of the game and decide what we should do. We will use a 2-layer neural network that takes the raw image pixels and produces a single number indicating the probability of going up or down.

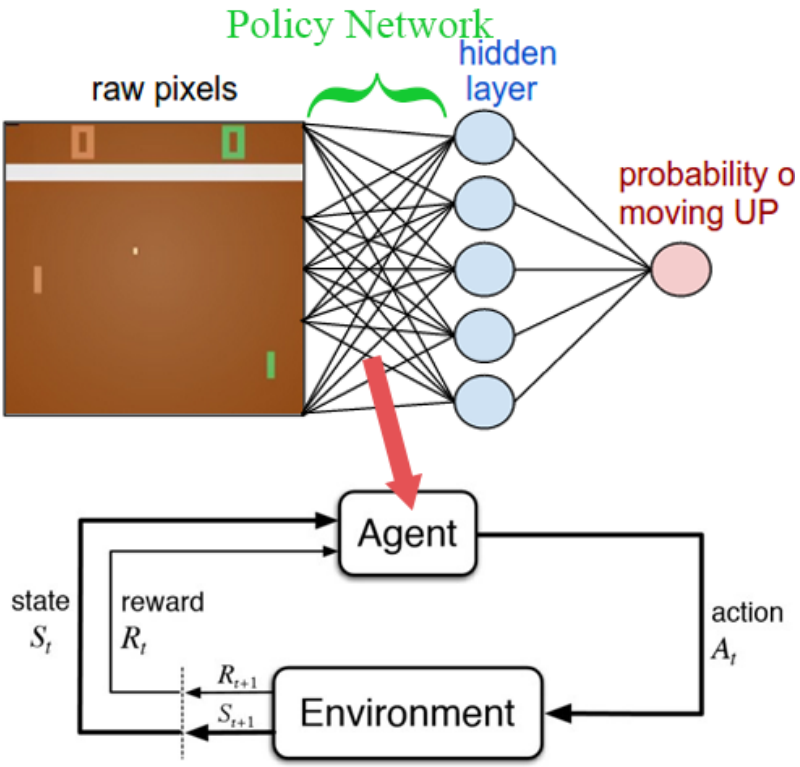

Figure 3-2-2-2 Policy gradient flow chart

In reality, the output of the network is going to consist of two numbers the probability of going up and the probability of going down, and what we will do while training is actually a sample from the distribution so that we are not always going to repeat the same exact actions and this will allow the agent to explore the environment a bit randomly(exploration) and hopefully discover better rewards and better behavior. Since we want to enable our agent to learn entirely by itself the only feedback that we're going to give it is the scoreboard in the game so whenever our agent manages to score a goal it will receive a reward of +1 and if the opponent scored a goal then our agent will receive a penalty of -1 and the entire goal of the agent is to optimize its policy to receive as much reward as possible.

In order to train our policy network, the first thing we're going to do is collect a bunch of experiences so you're just going to run a whole bunch of those game frames through your network select random actions feed them back into the engine, and just create a whole bunch of random pong games and now obviously since our agent hasn't learned anything useful yet it's going to lose most of those games.

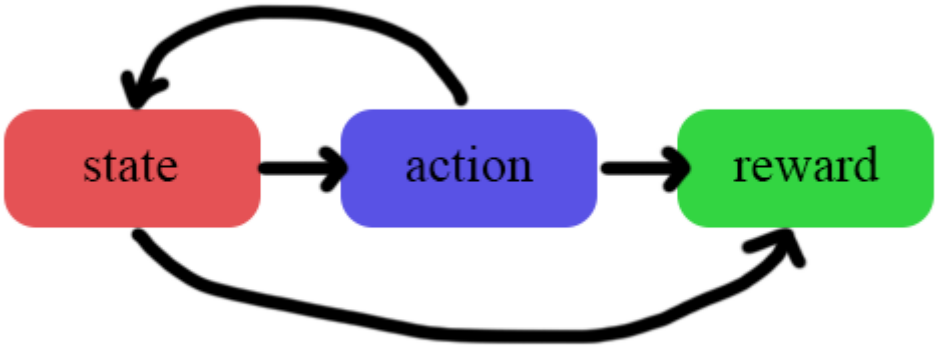

Figure 3-2-2-3 Flow Chartof relationship between state,action and rewrad

Sometimes our agent might get lucky and it's going to randomly select a whole sequence of actions that actually lead to scoring a goal and in this case, our agent is going to receive a reward. The key thing to understand is that for every episode regardless of whether we want a positive or a negative reward we can already compute the gradients that would make the actions that our agents have chosen more likely in the future and this is very crucial

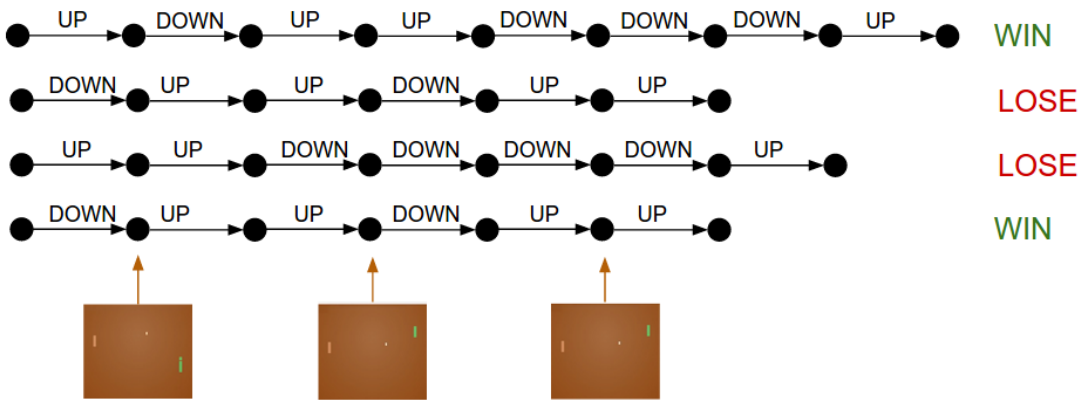

Figure 3-2-2-4 reward system

The main task for policy gradients is that for every episode where we've got a positive reward we're going to use the normal gradients to increase the probability of those actions in the future but whenever we got a negative we're gonna apply the same gradient but we're gonna multiply it by -1 and this minus sign will make sure that in the future all the actions that we took in a very bad episode and are going to be less likely in the future and so the result is that while training our policy network the actions that lead to negative rewards are slowly going to be filtered out and the actions that lead to positive rewards are going to become more and more likely so in a sense, our agent is learning how to play the game of Pong.

To further explain how policy Gradients work (again refer to the diagram below). Our policy network will calculate the probability of going UP as 30% (logprob -1.2) and DOWN as 70% (logprob -0.36) and then sample action from this distribution. For example, let's suppose we sample DOWN, and we will execute it in the game. At this point, we can notice one interesting fact: We could immediately fill in a gradient of 1.0 for DOWN, and find the gradient vector that would encourage the network to be slightly more likely to do the DOWN action in the future. So we can immediately evaluate this gradient. In the example below, going DOWN ended up with us losing the game (-1 reward). So if we fill in -1 for the log probability of DOWN and do backdrop we will find a gradient that *discourages* the network to take the DOWN action for that input in the future.

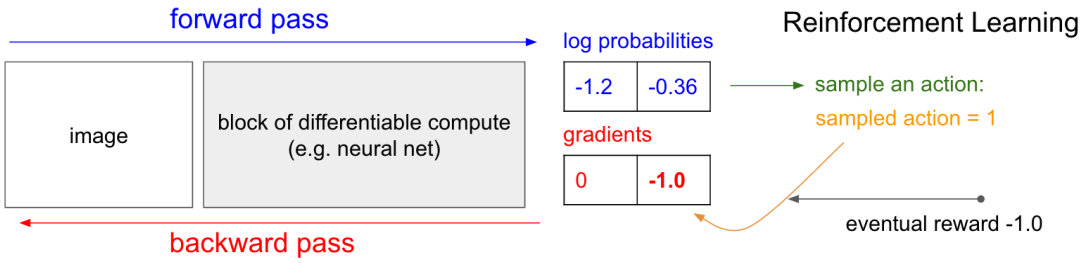

Figure 3-2-2-5 how policy gradient works

Chapter 4 Evaluation

4.1 Introduction

In this chapter we are going to build an AI agent that can win against a computer in the game pong I choose the game pong because it's a very simple game that does not have many rules. The AI agents will be using a neural network that we will be building from scratch. we will implement reinforcement learning once we get it running we can then implement more Atari games.

4.2 General Approach

We are going to use a new neural network because our AI agent does not have any pieces of information about how is the game built and how does it work so it will just be looking and taking images from the game and processing them basically removing the color background and then downsampling it to reduce the resolution. Afterward, it will be using a neural network to compute the probability of moving the paddle up or down. When the round is over we need to know whether the agent won or lost after each episode (when 20 points are reached) this will restart again and it would pass results through the backpropagation algorithm to compute the gradient for our weights. If the agent wins it will get a reward and if it loses it will not get any reward. This is going to push our agent to learn. This process is going to be repeated until our weights are tuned to the point where the AI will win against the computer.

4.3 Getting Things Ready

The first thing we need is open AI; is a research organization founded in 2015 by Elon Musk that specializes in digital intelligence, they launched a toolkit called Gym and this toolkit is for developing and comparing reinforcement learning algorithms by teaching agent anything from walking to playing games. I checked out the website and went through the documentation and I installed the library that I need for the game of pong (pip install gym[Atari]). After installing we took a test code from the website to make sure everything is working properly.

4.4 Explanation of the code

For this project I followed the tutorial on a website called the robot camp and they have some really nice tutorials about reinforcement learning applications.

4.4.1 Imports

Here I am going to explain the libraries that I used for this project.

Gym Library: as mentioned before it's a library with a lot of applications like Atari games

Numpy Library: used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices.

Gym Wrappers: override how the environment processes observations, rewards, and action.

Time: provides many ways of representing time in code, such as objects, numbers, and strings.

Random: generate random numbers

```
import gym
import numpy as np
import gym.wrappers
import random
import time
```

4.4.2 Image Processing

In the preprocessing part of the program there are 4 main functions:

1. Crop the image and keep only the parts that we need and convert from 80x80 matrix to 1600x1 matrix(flatten the matrix)

```
processed_observation = processed_observation.astype(np.float).ravel()
```

2. Removing the background

```
def remove_background(image):
```

3. downsample the image resolution because we only need half of it

```
def downsample(image):
```

4. Removing color from image

```
def remove_color(image):
```

4.4.3 Activation Function

The sigmoid(x) and relu(x) refer to the functions performing sigmoid and RELU activation calculations respectively

```
def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))

def relu(vector):
    vector[vector < 0] = 0
    return vector
```

4.4.4 Neural Network

In this subsection I will go over the code of Neural Network and explain it.

```
def neural_net(observation_matrix, weights):
```

will calculate the output of each layer

```
hidden_layer_values = np.dot(weights['1'], observation_matrix)
```

apply an activation function

```
hidden_layer_values = relu(hidden_layer_values)
```

using the activation function hidden layers to calculate the final output value
and apply a sigmoid function to get a better result between 0 and 1 and therefore a valid probability

```
output_layer_values = sigmoid(output_layer_values)
```

Take the probability and convert it into an action return 2 mean the stick will go up and return 3 will go down

```
def Move_up_or_down(probability):
    random_value = np.random.uniform()
    if random_value < probability:
        # up in openai gym
        return 2
    else:
         # down in openai gym
        return 3
```

Compute gradients

```
def compute_gradient(gradient_log_p, hidden_layer_values,
observation_values, weights):
    delta_L = gradient_log_p
    dC_dw2 = np.dot(hidden_layer_values.T, delta_L).ravel()
    delta_l2 = np.outer(delta_L, weights['2'])
    delta_l2 = relu(delta_l2)
    dC_dw1 = np.dot(delta_l2.T, observation_values)
    return {
        '1': dC_dw1,
        '2': dC_dw2
    }
```

update the weights

```
def weights_update(weights, expectation_g_squared, g_dict, decay_rate,
learning_rate):
    epsilon = 1e-5
    for layer_name in weights.keys():
        g = g_dict[layer_name]
        expectation_g_squared[layer_name] = decay_rate *
expectation_g_squared[layer_name] + (1 - decay_rate) * g**2
        weights[layer_name] += (learning_rate *
g)/(np.sqrt(expectation_g_squared[layer_name] + epsilon))
        g_dict[layer_name] = np.zeros_like(weights[layer_name])
```

4.4.5 Reinforcement Learning

Now we are gonna use RL the process is like so; the actions taken towards the end of an episode are influencing our learning more than actions taken at the beginning and this is called discount rewards(takes a list of reward corresponding to different time-steps as input and outputs a list of discounted rewards corresponding to different time-steps). Also we going to use back propagation to compute the gradient for each episode and this is the most important part of RL is how our agent will figure how to improve over time.

```
def discount_rewards(rewards, gamma):
    discounted_rewards = np.zeros_like(rewards)
    running_add = 0
    for t in reversed(range(0, rewards.size)):
        if rewards[t] != 0:
            running_add = 0
            running_add = running_add * gamma + rewards[t]
            discounted_rewards[t] = running_add
    return discounted_rewards

def discount_plus_rewards(gradient_log_p, episode_rewards, gamma):
        discounted_episode_rewards = discount_rewards(episode_rewards,
gamma)
        discounted_episode_rewards -=
np.mean(discounted_episode_rewards)
    discounted_episode_rewards /= np.std(discounted_episode_rewards)
    return gradient_log_p * discounted_episode_rewards
```

4.4.6 Main Function

Finally all we need to do is to create the environment and incorporat the previous functions to make the agent learn step by step over multiple episodes

```
def main():
    env = gym.make('Pong-v4', render_mode='human')
    height, width, channels = env.observation_space.shape
    actions = env.action_space.n
    env.unwrapped.get_action_meanings()
```

4.4.7 Images from the Pong game

For this agent to be fully functional the tutorial states that it needs 1 day to train but the convenience of this report I ran the test over 3 hours to see how well it did the following images show the results:

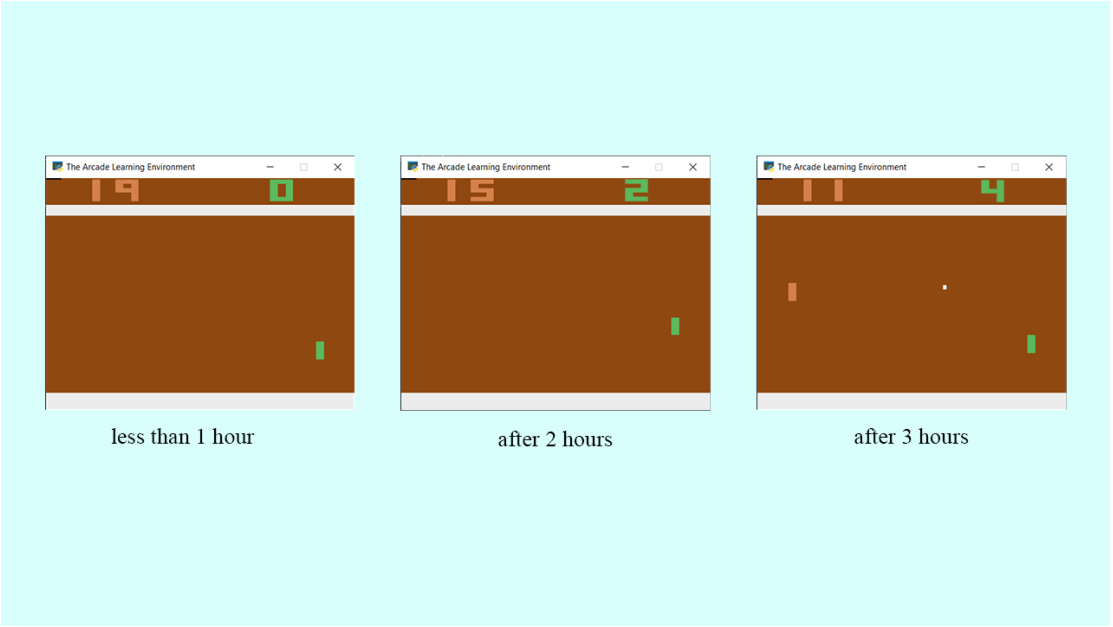

Figure 4-4-7 Pong Agent after training

Chapter 5 Conclusion

In this essay, I learned a lot about machine learning and the implementation of RL in video games which is a very interesting topic. I found one of the most fascinating toolkits for RL which is gym and I followed tutorials on how to make it solve the problems I had with the game. I gained a good understanding of RL as it addresses the problem of learning control strategies for autonomous agents with less to no data. It learns itself continuously so it gets better and better at doing the task. I learned that RL uses the trial and error method to improve the agent. Also, I acquired knowledge of policy gradients and how they work in RL, and how it increases the probability of an action that gives positive rewards. Finally, I covered how we reward and punish algorithms based on the predictions and actions they make with the example of the Pong game.

References:

https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2
https://www.gymlibrary.ml/environments/atari/adventure/
https://www.gymlibrary.ml/content/tutorials/
https://www.oreilly.com/radar/introduction-to-reinforcement-learning-and-openai-gym/
https://towardsdatascience.com/intro-to-reinforcement-learning-pong-92a94aa0f84d
https://medium.com/gradientcrescent/fundamentals-of-reinforcement-learning-automating-pong-in-using-a-policy-model-an-implementation-b71f64c158ff
https://www.youtube.com/watch?v=JgvyzIkgxF0
https://www.youtube.com/watch?v=CBb0Q3GCHh0
https://therobotcamp.com/2020/04/21/tutorial-teach-ai-to-play-pong-from-scratch-with-reinforcement-learning/
https://towardsdatascience.com/reinforcement-learning-fda8ff535bb6
https://en.wikipedia.org/wiki/Reinforcement_learning
http://www.scholarpedia.org/article/Reinforcement_learning#Background_and_History
https://medium.com/swlh/a-brief-history-of-reinforcement-learning-in-game-play-d0861b2b74ef
http://matt.colorado.edu/teaching/highcog/readings/sb98c1.pdf
https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf