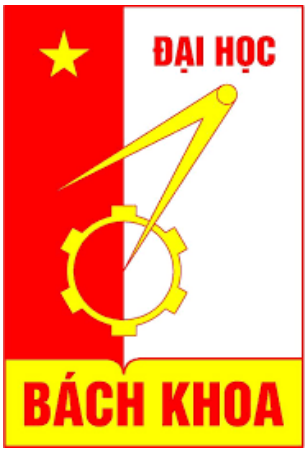
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO PROJECT I: HUẤN LUYỆN GAME RẮN SĂN MỒI SỬ DỤNG MẠNG NƠ RON

Sinh viên: LÊ XUÂN NAM

MSSV: 20162814

GIẢNG VIÊN HƯỚNG DẪN: TS. NGUYỄN TUẤN DŨNG

HÀ NỘI 2021

MỤC LỤC

Chương 1: Tổng quan về đề tài

1.1 Giới thiệu

Những năm gần đây, AI- Artificial Intelligence (Trí tuệ nhân tạo) và cụ thể hơn là Machine Learning (Máy học)-tập con của AI nổi lên như một minh chứng cho cách mạng công nghiệp lần thứ 4 (1- Động cơ hơi nước, 2- năng lượng điện, 3- công nghệ thông tin). AI hiện diện trong mọi lĩnh vực của đời sống con người, từ kinh tế, giáo dục, y khoa cho đến công việc nhà, giải trí hay thậm chí là trong quân sự. Những ứng dụng nổi bật của AI đến từ nhiều lĩnh vực để giải quyết nhiều vấn đề khác nhau. Nhưng những đột phá phần nhiều đến từ Deep Learning (Học sâu) một tập con của Machine Learning. Deep Learning đã giúp máy tính thực hiện những công việc tưởng chừng như không thể vào 15 năm trước như phân loại hàng ngàn vật thể khác nhau trong bức ảnh, tự tạo chú thích cho ảnh, bắt trước giọng nói và chữ viết của con người, …Chính vì vậy em thực hiện đề tài "Huấn luyện game rắn săn mồi sử dụng mạng nơ ron" để hiểu hơn về mạng nơ ron và cơ chế hoạt động của nó.

1.2 Mạng nơ ron là gì?

Con người có thể phân biệt được các đồ vật khác nhau, các loài vật khác nhau. Một việc nghe chừng rất đơn giản đối với con người nhưng lại cực kỳ khó để có thể thực hiện bằng máy tính. Sự khác biệt chính là ở bộ não của con người – nơi có hàng triệu các nơ ron thần kinh liên kết với nhau.

⇨ Mạng nơ ron ra đời.

Mạng nơ ron (Neural network) là một hệ thống đồ thị tính toán lấy cảm hứng từ sự hoạt động của các nơ ron trong hệ thần kinh.

Mỗi mô hình mạng nơ ron gồm có 1 input layer, 1 output layer và có thể có hoặc không có các hidden layer. Tổng số layer trong mô hình được quy ước là: tổng số layer -1 do không tính input layer.

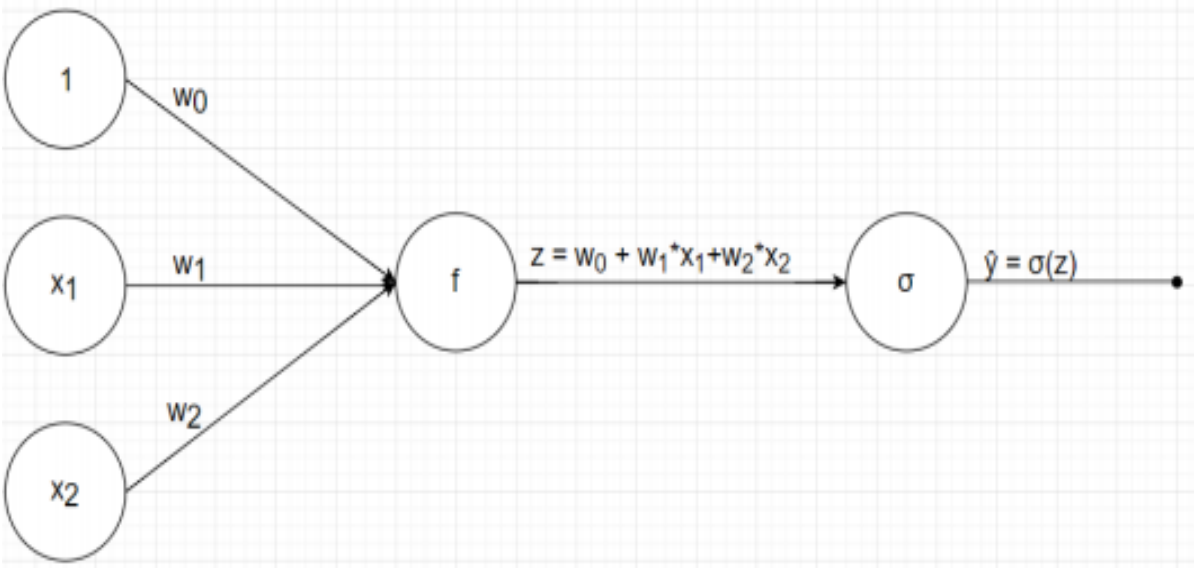


Figure 1 Mô hình Logistic Regression- Mô hình mạng nơ ron đơn giản nhất

Mô hình trên là mô hình mạng nơ ron đơn giản nhất, nó chỉ gồm input layer và output layer mà không có hidden layer.

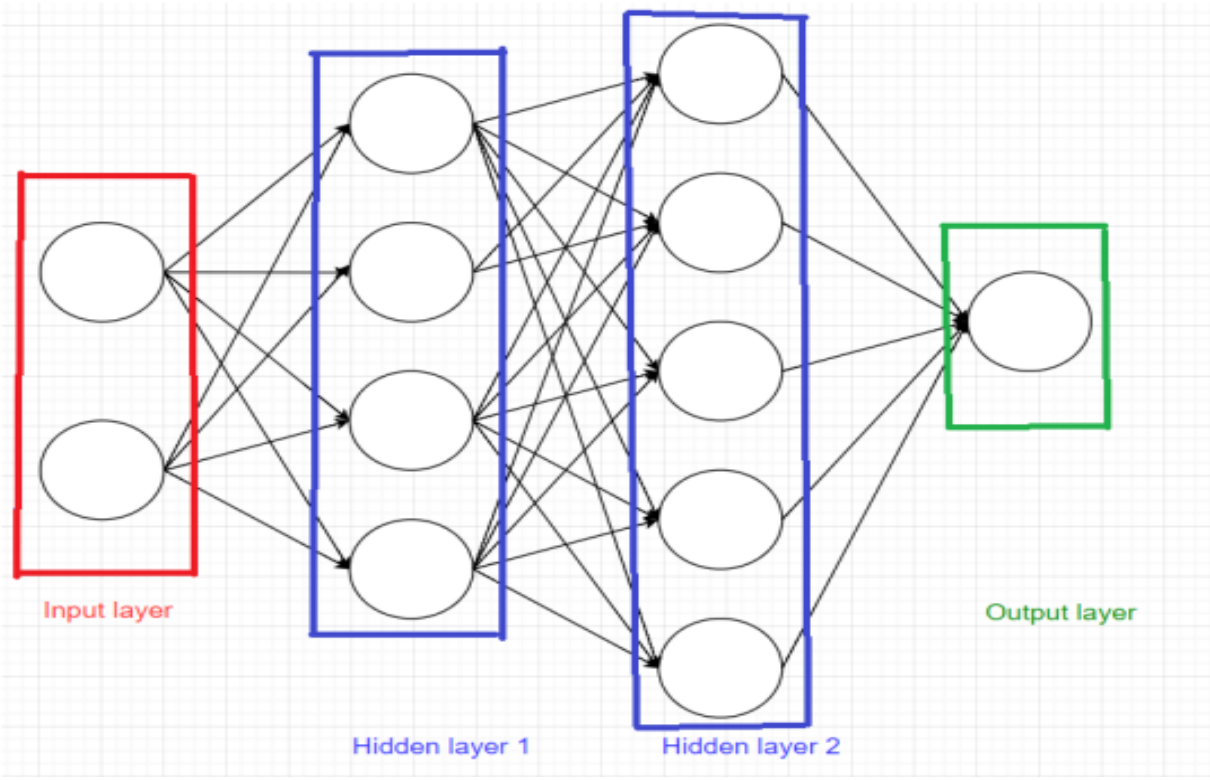


Figure 2 Mô hình mạng nơ ron tổng quát

Mô hình trên gồm 3 layer: 2 hidden layer và 1 output layer.

Để giúp máy tính thực hiện các công việc phức tạp thì mạng nơ ron cần có các thuật toán đặc thù: feedforward, backpropagation, …và thực tế cho thấy các mạng nơ ron phức tạp thường sẽ giải quyết được nhiều các bài toán khó hơn, chính vì vậy các mạng nơ ron phức tạp lần lượt ra đời để giải quyết các vấn đề khó: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), …

Chương 2: Xây dựng trò chơi

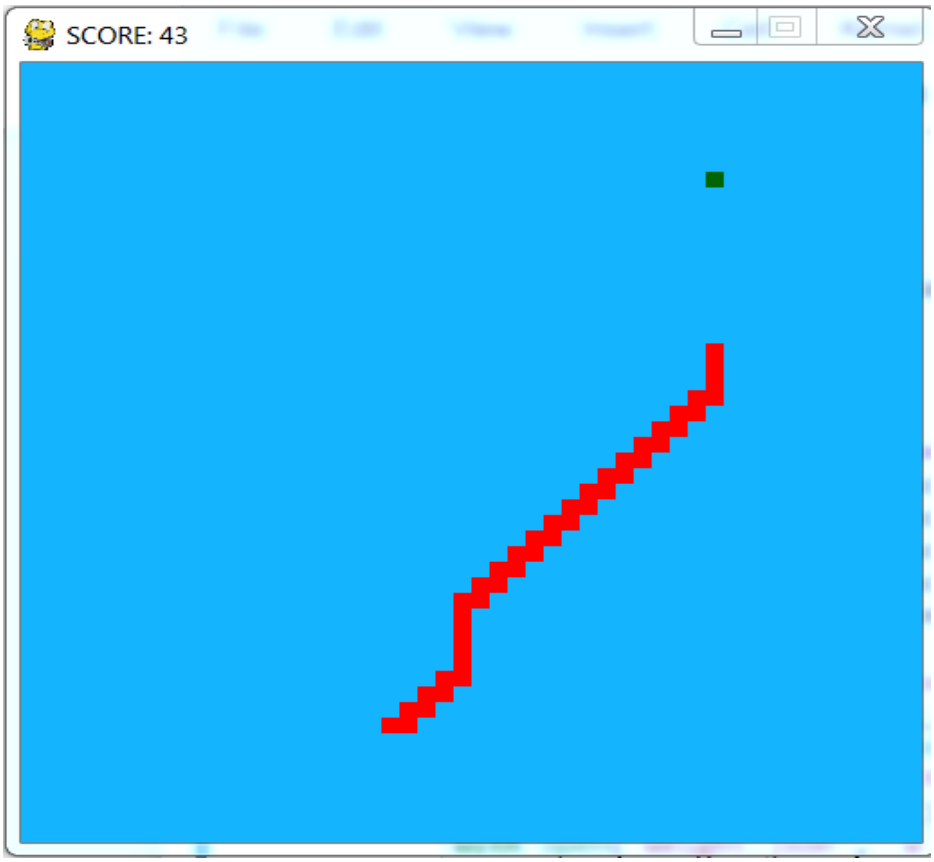
2.1. Giới thiệu

Game rắn săn mồi là tựa game khá nổi tiếng và gắn liền với tuổi thơ của nhiều người. Yêu cầu của game là người chơi cần vỗ béo cho con rắn, sử dụng các nút điều khiển; rẽ trái hoặc rẽ phải để điều khiển con rắn ăn mồi hoặc tránh các chướng ngại vật, nếu con rắn đâm vào tường hoặc tự cắn vào đuôi của nó thì trò chơi sẽ kết thúc.

2.2. Xây dựng trò chơi

- Trước tiên ta tạo màn hình của trò chơi bằng cách sử dụng hàm: pygame.display.set_mode
- Sau đó ta sẽ thực hiện cài đặt chiều dài, chiều rộng và màu của màn hình game.
- Tạo rắn và thức ăn
- Cài đặt thời gian di chuyển của con rắn
- Cài đặt cơ chế chơi game

```
from game import *

display_width = 500
display_height = 500
red = (255,0,0)
black = (0,0,0)
blue = (20,180,255)

pygame.init()
display=pygame.display.set_mode((display_width,display_height))
clock=pygame.time.Clock()

training_data_x, training_data_y = generate_training_data(display,clock)
```



Hình trên là giao diện của game đã được xây dựng.

Chương 3 Xây dựng mạng nơ ron

3.1 Tạo dữ liệu

Đầu vào của bài toán gồm 7 node:

- Bị chặn bên trái hoặc có chướng ngại vật ở bên trái (0 hoặc 1)
- Bị chặn phía trước hoặc có chướng ngại vật ở phía trước (0 hoặc 1)
- Bị chặn bên phải hoặc có chướng ngại vật ở bên phải (0 hoặc 1)
- Hướng vecto của quả táo với đầu con rắn (X)
- Hướng vecto của quả táo với đầu con rắn (Y)
- Hướng vecto của con rắn (X)
- Hướng vecto của con rắn (Y)

Đầu ra của bài toán gồm 3 node:

- Rẽ trái [1, 0, 0]
- Đi thẳng [0, 1, 0]

- Rẽ phải [0, 0, 1]

Để tạo dữ liệu đưa vào mô hình để train ta có thể tự điều khiển con rắn hoặc cho nó chơi tự động rồi lưu dữ liệu. Ở đây em chọn viết chương trình cho con rắn chơi tự động sau đó dữ liệu có được em sẽ đưa vào mô hình để train.

Để tạo dữ liệu tự động em đã sử dụng góc giữa con rắn và quả táo để tính toán xem con rắn nên đi theo hướng nào.

Trước tiên ta cần tính hướng vecto của con rắn, sau đó ta tính hướng vecto giữa đầu con rắn và quả táo. Góc giữa con rắn và quả táo chính là góc giữa 2 vecto. Đoạn code sau sẽ thực hiện tính 2 vecto trên: angle_with_apple là góc ta cần tính.

```
def angle_with_apple(snake_position, apple_position):
    apple_direction_vector = np.array(apple_position) - np.array(snake_position[0])
    snake_direction_vector = np.array(snake_position[0]) - np.array(snake_position[1])
```

Sau khi tính toán xong góc, ta cần quyết định xem con rắn nên đi theo hướng nào:

- Nếu góc giữa con rắn và quả táo lớn hơn 0, điều này có nghĩa là quả táo đang ở bên phải con rắn. Vì vậy con rắn nên rẽ sang bên phải.
- Nếu góc giữa con rắn và quả táo nhỏ hơn 0, điều này có nghĩa là quả táo đang ở bên trái con rắn. Vì vậy nó nên rẽ sang bên trái.
- Nếu góc giữa con rắn và quả táo bằng 0, điều này có nghĩa quả táo nằm trước con rắn. Vì vậy nó nên tiếp tục đi thẳng.
- Góc lớn hơn 0 đặt là 1, góc nhỏ hơn 0 đặt là -1 và góc bằng 0 đặt là 0.
- Và các hướng: lên, xuống, phải và trái đặt tương ứng là 3, 2, 1 và 0.

Khi đã tính toán xong các bước di chuyển của con rắn, ta cần tạo danh sách dữ liệu huấn luyện. Với mỗi bước đầu vào và đầu ra, ta sẽ thêm nó nào danh sách huấn luyện.

Thực hiện chơi 1000 lần. Với mỗi lần chơi, em sẽ khởi tạo lại vị trí của con rắn và điểm số. Sau đó tạo 2 danh sách trống: đầu vào (X) và đầu ra (Y).

```
#tao du lieu
from game import *

def generate_training_data(display, clock):
    training_data_x = []
    training_data_y = []
    training_games = 1000
    steps_per_game = 2000

    for _ in tqdm(range(training_games)):
        snake_start, snake_position, apple_position, score = starting_positions()
        prev_apple_distance = apple_distance_from_snake(apple_position, snake_position)

        for _ in range(steps_per_game):
            angle, snake_direction_vector, apple_direction_vector_normalized, snake_direction_vector_normalized = angle_with_appl
                snake_position, apple_position)
            direction, button_direction = generate_random_direction(snake_position, angle)
            current_direction_vector, is_front_blocked, is_left_blocked, is_right_blocked = blocked_directions(
                snake_position)

            direction, button_direction, training_data_y = generate_training_data_y(snake_position, angle_with_apple,
                                                                                     button_direction, direction,
                                                                                     training_data_y, is_front_blocked,
                                                                                     is_left_blocked, is_right_blocked)

            if is_front_blocked == 1 and is_left_blocked == 1 and is_right_blocked == 1:
                break

            training_data_x.append(
                [is_left_blocked, is_front_blocked, is_right_blocked, apple_direction_vector_normalized[0],
                 snake_direction_vector_normalized[0], apple_direction_vector_normalized[1],
                 snake_direction_vector_normalized[1]])

            snake_position, apple_position, score = play_game(snake_start, snake_position, apple_position,
                                                              button_direction, score, display, clock)
```

3.2 Xây dựng mạng nơ ron

Mạng nơ ron của em bao gồm:

- 7 node ở input layer.
- 3 node ở output layer.
- Và một số hidden layer.

Sau một vài thử nghiệm và tham khảo ở các tài liệu em đã tìm được kiến trúc mạng phù hợp với bài toán. Ở đây em dùng keras để xây dựng mạng. Mô hình mạng đã được xây dựng:

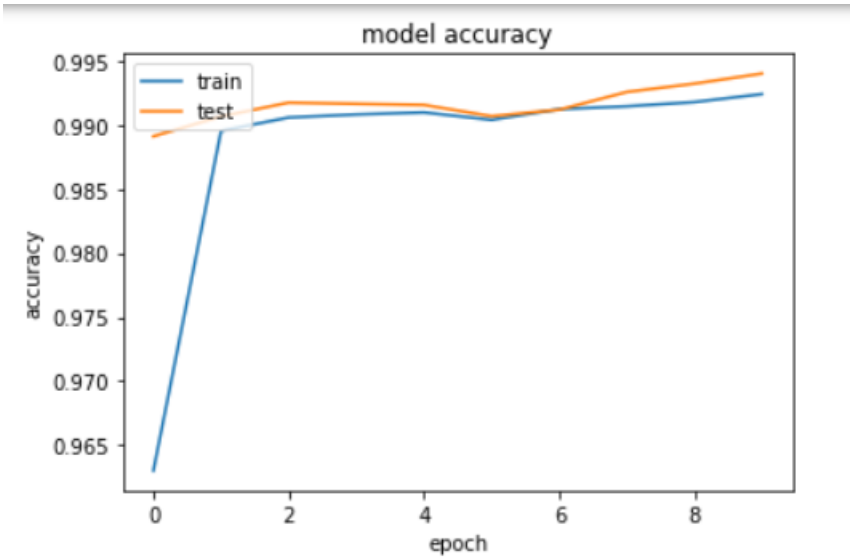
```
model = Sequential()
model.add(Dense(units=9,input_dim=7, activation='relu'))

model.add(Dense(units=15, activation='relu'))
model.add(Dense(units=30, activation='relu'))


model.add(Dense(3,  activation = 'softmax'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.summary()
history=model.fit(np.array(training_data_x).reshape(-1, 7),
                  np.array(training_data_y).reshape(-1, 3),
                  batch_size = 256,
                  epochs= 10,
                  validation_split=0.25,
                  verbose=1)
# luu weight da training vao file 'weight.h5'
model.save_weights('weight.h5')
model_json = model.to_json()
with open('weight.json', 'w') as json_file:
    json_file.write(model_json)
```

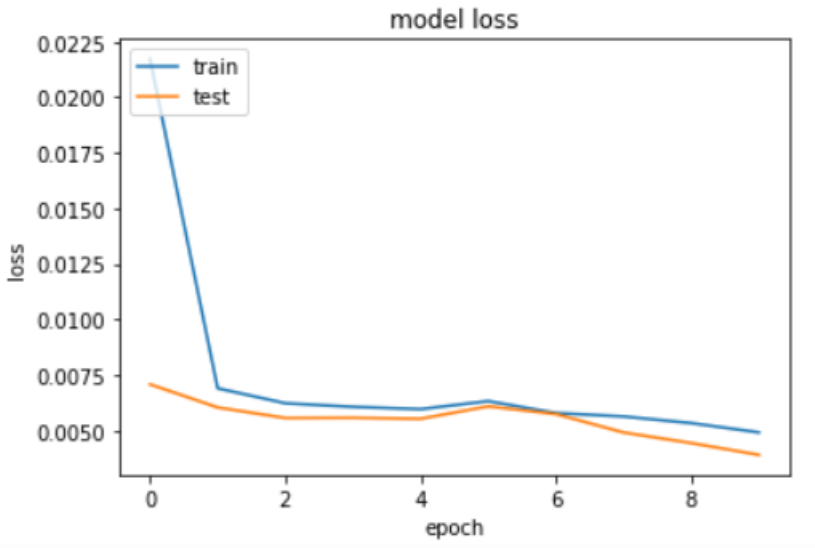
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 9)                 72
_________________________________________________________________
dense_24 (Dense)             (None, 15)                150
_________________________________________________________________
dense_25 (Dense)             (None, 30)                480
_________________________________________________________________
dense_26 (Dense)             (None, 3)                 93
=================================================================
Total params: 795
Trainable params: 795
Non-trainable params: 0
```

Hình trên là các tham số của mô hình đã xây dựng.

3.3 Train và Test.

Sau khi xây dựng model xong ta tiến hành train với dữ liệu đã được tạo ra từ trước. Ta tiến hành train với 10 epoch và kết quả thu được:





Sau khi train xong ta thực hiện test:

Điểm cao nhất là 54

Điểm trung bình đạt được là 22.48

```
Maximum score achieved is:   54
Average score achieved is:   2.248
```