Python 으로 훈련하는 AI 로 게임 놀기

LI LIUYANG

SONG SHENGHUI

요약

몇년전부터 시작한 자유주행 자동차의 연구,요즘에 엄청 핫한 chat gpt(자연언어를 처리하는 AI)은 모두 AI 를 가르키고있습니다.그래서 AI 은 당연하게도 4 차 산업혁명의 핵심입니다.이므로써 AI 를 더많이 알고 더 많은 AI 를 만드것이도 중요한일입니다.

1. 서론

1.1. 연구배경

지금 사회의 경쟁이 점점 심해지고 스마프폰의 유행으로 사람들은 점점 재미를 위해 사교성을 위해 자존심을 의해 게임을 접촉하게됩니다.스마트폰과 컴퓨터의 성능을 따라 게임의 종류나 화질등등이 많이 늘어지고있습니다.게임의 상업가치도 늘어나 많은 돈이 게임이나 케리터쪽으로 모으기 시작합니다.

스마폰과 컴퓨터의 성능이 늘어나는 가운데 게임만 진화하는 것이 아니라 AI 이라는 4 차혁명을 이끄는 새로운 "생명체"도 폭발적으로 진화하고있다.Alpha , Siri,Deep Blue, Tesla Autopilot, Google Translate,Chat gpt 등등 유명한 AI 들이 사람의 일상에 들어가고있습니다.

게임으로 AI 을 연구하는 이유는 첫쨰 AI 개발에는 알고리즘과 모델의 지속적인 최적화 , 그리고 많은 량의 데이터와 컴퓨팅파워 지원이 필요합니다.그래서 게임을통하여 A 에게 상대적으로 밀페한 환경을 제공할수있고 연구자가 알고리즘과 모델연구 및개선에 집중할수있게 만들수있습니다.게임은 또 새로운 알고리즘과 모델의 테스트용으로도 사용할수있습니다.

둘쨰,게임으로 AI 를 연구하는것이 중요한 실용적 가치가 가지고있습니다.게임은 복잡한 결정 및 비상 대응등,AI 가 게임에서 보여준 성능은 현실속 응용가치과 AI 의 숨어있는 재능 개발에 반응하기도 합니다

셋쨰,게임의 복잡성에 따라 컴퓨터 비전, 강화학습,자연언어처리,등 컴퓨터 관한 많은 분야들이 서로 교차 적용을 도와줍니다.인해 만들어 지는 AI 은 더 스마트하고 점점 "사람"처럼 될수있습니다.

마지막으로는 종종게임은 점점 사람들이 인정하는 스포츠로 변하고있습니다.어떤 게임의 아시아 게임(리그 오브 레전드 , 하스스톤)의 정식 시합항목으로도 되고있습니다.인해 게임의 지명도에 의하여 게임을 노는 AI 를 만들어 내면 많은 사람에게관심을

유도할수있으며 AI 에 대한 대중의 인식과 이해를 향상시킬수있다.

1.2. 연구목표

1.게임에 대한 강화 학습의 적용하는 방법을 배운다.간단하지만 도전적인 게임으로 AI 가 게임에서 더 높은 점수를 얻도록 훈련함으로써 딥 로딩,심화 학습 알고리즘의 성능과 적용효과를 게임에서 테스트할구있습니다

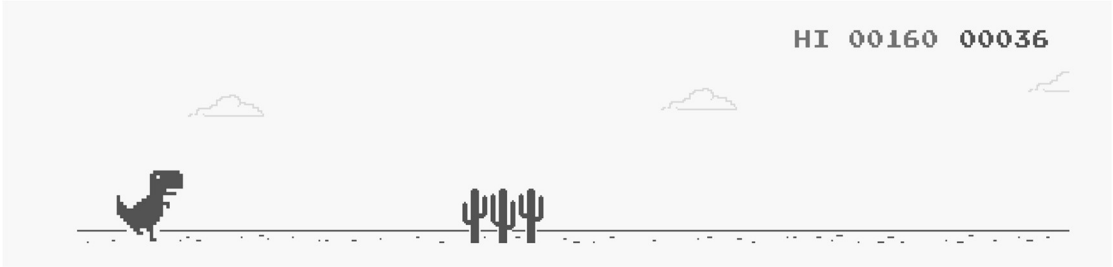
2.게임 환경에서 실제 상황에 있는 AI 의 의사 결정 기능을 연구합니다.물리엔지을 기반으로하는 게임을 선택하여 AI 가 게임규칙과 물리 모델을 장악하여 자애물을 피하기 위해 게임 캐릭터를 컨트록하기 위한 결정을 내려야합니다.이를 통해 실제 결정 프로세스를 더잘이해하고 시뮬레이션을 할수있다.

3. 게임 분야에서 AI 의 일반화 능력을 탐색합니다. 간단한 게임 규칙과 물리 모델로 훈련한 AI 가 다른 복잡한 게임의 표현을 연구할 수 있습니다.

2. 관련연구

2.1. 게임 선택

2.1.1 google Chrome Dino



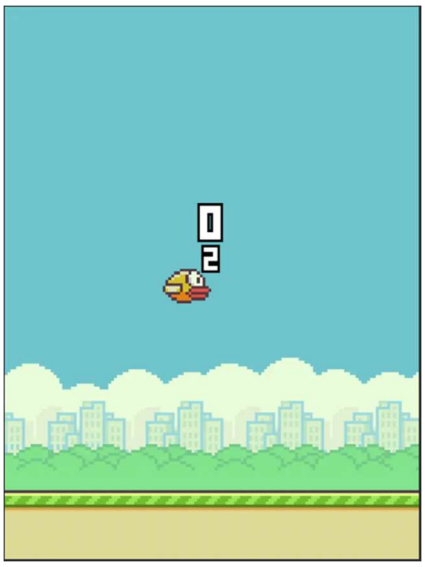
이 게임은 아주 간단합니다. Dino(공룡)을 조종하여 멀리 달리면 되는 게임 입니다. 처음에는 아주 늦습니다 속도가 점점 빨라 집니다.물리엔진을 사용하였습니다

보다싶이 저의 최고 기록은 645 입니다.놀면 놀수록 빠져드는 마력이 있습니다 만든 AI 로 이게임의 최고로 가는 것도 나쁘지 않다고 생각합니다.

2.1.2. Flappy Bird

"Flappy Bird"는 베트남의 독립 게임 개발사인 Dong Nguyen 이 개발한 작품으로 2013 년 5 월 24 일에 출시되어 2014 년 2 월에 갑자기 인기를 끌었습니다. 2014 년 2 월 개발자

자신이 "Flappy Bird"를 Apple 및 Google 앱 스토어에서 제거했습니다. 2014 년 8 월 공식적으로 APP STORE 로 돌아와 Flappy 팬들이 오랫동안 기다려온 멀티플레이어 모드에 공식적으로 합류했습니다. 게임에서 플레이어는 다양한 길이의 송수관으로 구성된 장애물을 건너기 위해 새를 조종해야 합니다.보다싶이 간단한 게임 룰이지만 인류의 힘으로는 높은 점수를 얻기 엄청힘드는 게임입니다.

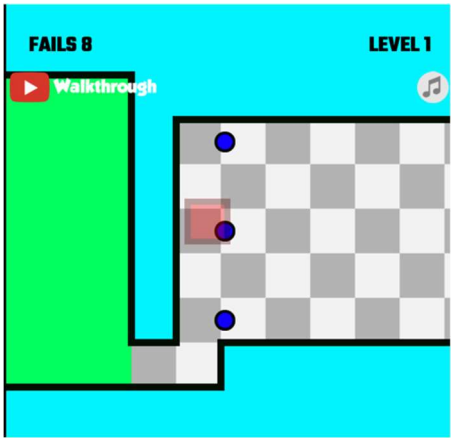


최고 기록 2 입니다..

2.1.2. The World's Hardest Game

미국의 플래시 게임 사이트인 스너비 랜드(Snubby Land)의 유저 중 한 명인 Stevie 가 만든 게임이다. 현재 4 탄까지 나왔으며, 줄여서 TWHG1, TWHG2, TWHG2.1[1] 등으로 불린다. 국내에서는 제목을 직역한 '세상에서 가장 어려운 게임'으로 유명하다. 줄여서 '세가게'[2]로 부르기도 한다.

일단 게임의 조작 자체는 방향키만 사용하면 되므로 매우 쉬우며, 게임의 목적 또한 장애물들을 피하며 비콘을 먹고 목적지에 도착하는 것으로 매우 단순하다. 그러나 괜히 "가장 어려운"이라는 수식어가 장식이 아니라는 것을 증명하듯, 그 장애물들을 웬만한 요령 없이는 쉽게 빠져나갈 수 없도록, 그러나 어떻게든 파고들면 결국 헤쳐나갈 수 있는 말 그대로 플레이어의 경험을 시험하는 미로 게임이다.

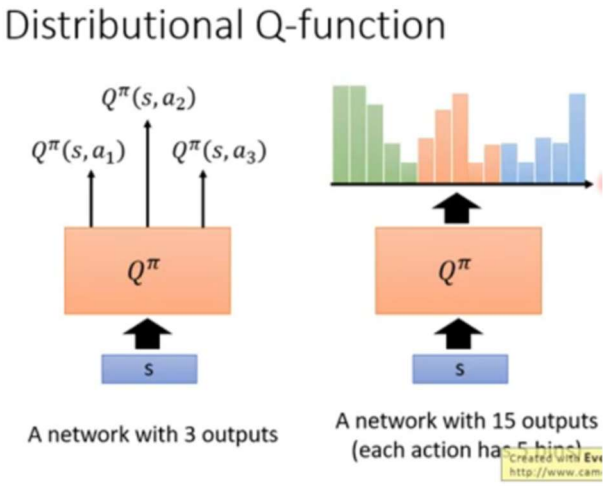


2.2. 알고리즘 선택

2.2.1 DQN

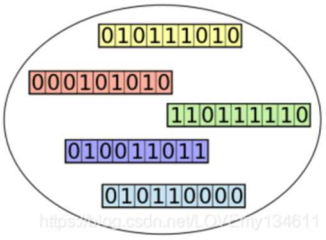
DQN(Deep Q-Network)은 2013 년 Google DeepMind 에서 제안한 딥 로딩 알고리즘입니다. DQN 알고리즘은 심층 신경망을 사용하여 정책 학습을 달성하기 위해 Q 함수를 근사화합니다.

DQN 알고리즘의 핵심은 강화 학습의 Q-러닝 알고리즘을 심층 신경망과 결합하여 복잡한 환경에서 지능형 결정 학습을 실현하는 것입니다. DQN 알고리즘은 심층 신경망을 사용하여 고차원 상태 공간을 처리할 때 기존 Q-러닝 알고리즘이 직면한 문제를 효과적으로 해결할 수 있습니다. 동시에 DQN 알고리즘은 학습 효율성과 안정성을 향상시키기 위해 경험 재생 및 대상 네트워크와 같은 일부 기술을 사용합니다.



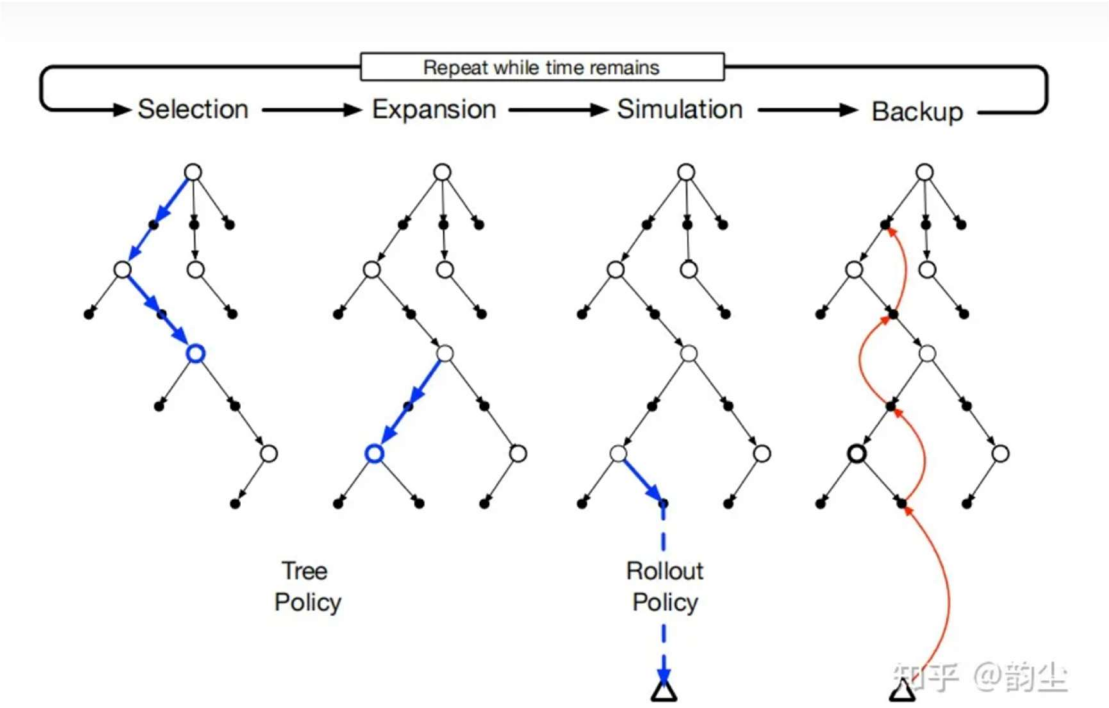
2.2.2 Genetic Algorithm

유전 알고리즘(Genetic Algorithm, GA)은 미국의 John Holland 이 1970 년대에 처음 제안한 알고리즘이며, 이 알고리즘은 자연계에서의 생물체의 진화 원리를 기반으로 설계되었습니다. 다윈의 진화론에서 생물 진화 과정의 자연 선택 및 유전학 기구를 모사하는 계산 모델로, 최적해를 탐색하기 위한 모의 자연 진화 과정을 모사하는 방법입니다. 이 알고리즘은 문제 해결 과정을 염색체의 유전자 교차, 돌연변이 등과 같은 생물 진화와 유사한 과정으로 변환하여 컴퓨터 시뮬레이션 연산을 수행합니다. 보다 복잡한 조합 최적화 문제를 해결하는 경우, 일부 일반적인 최적화 알고리즘에 비해 보통 더 빠르게 우수한 최적화 결과를 얻을 수 있습니다.



2.2.3 Monte Carlo Tree Search

Monte Carlo Tree Search 은 인공지능 분야에서 사용되는 탐색 알고리즘 중 하나입니다. 이 알고리즘은 대규모 게임 트리에서 최적의 움직임을 결정하는 데 사용됩니다. MCTS 는 무작위 시뮬레이션과 트리 탐색을 결합하여 게임 상황에서 가장 최적의 행동을 선택하는 데 사용됩니다. 이 알고리즘은 트리의 깊이를 제한하면서도 매우 높은 수준의 탐색 정확성을 보장할 수 있습니다. MCTS 는 현재 가장 인기있는 강화 학습 기술 중 하나이며, 인공지능 분야에서 많이 사용됩니다.



2.3 게임 및 알고리즘 선택

위의 있는 게임과 알고리즘을 분석하고 토론후의 결과 우리는 DQN 알고리즘과 Flappy Bird 을 선택하였습니다

선택하는 이유는 재일중요한 것이 실력입니다.우리는 pygame 로 게임을 만들 생각입니다 Dino 은 좀간단하고 만든 AI 은 한 10 만 번 돌리면 쉽게 게임을 크리얼할수있고 Dino 런의 최고 기록을 만들수있는 Bug 가 인터넷에서 떠들고 있습니다. 그리고 The World's

Hardest Game 은 한 라운드에서 게속 게임하는 것이아니라 하나를 크리어하면 또다시 다른 라운드로 들어갑니다 게임을 크리어 하면 너무 많은 시간이 투입해야합니다.그래서 너무 간단한것 말고 어려운것말고 쉽게 만질수있을것 같지만 않되는 중간 나이도를 선택하였습니다.

알고리즘의 선택은

1. DQN 알고리즘은 딥러닝 방법론으로써, 환경의 특징과 동적 변화에 대한 적응적 학습이 가능하므로, 지속적으로 변화하는 Flappy Bird 게임에서 우수한 성능을 나타낼 수 있습니다.

Flappy Bird 는 상태 공간과 행동 공간이 매우 큰 문제이며, DQN 알고리즘은 합성곱 신경망 등의 딥러닝 모델을 사용하여 이러한 고차원 데이터를 효과적으로 처리할 수 있습니다. 반면, Genetic Algorithm 알고리즘과 Monte Carlo Tree Search 알고리즘은 더 복잡한 코딩 와 적합성 함수를 설계해야 합니다.

3. 프로젝트 내용

3.1 Project 구성

3.1.1 언어 선택

python 은 학습 및 사용이 쉬운 프로그래밍 언어로 NumPy, Pandas, Scikit-learn 등과 같은 엔지니어가 데이터 처리, 데이터 시각화 및 모델링과 같은 작업을 신속하게 수행할 수 있는 풍부한 타사 라이브러리와 도구가 있다.

python 은 간단한 문법과 이해하기 쉬운 코드 구조를 가지고 있어 쉽게 배우고 사용할 수 있는 언어가 된다.

python 의 대화형 환경(예: Jupyter Notebook)은 엔지니어가 데이터를 더 잘 이해하고 시각화하고 신속하게 반복 모델링하는 데 도움이 될 수 있기 때문에 기계 학습 작업에 적합한다.

3.1.2 컴퓨터 비전 라이브러리 선택

OpenCV 는 이미지 처리, 특징 추출, 물체 감지, 표적 추적과 같은 컴퓨터 비전 작업에 사용할 수 있는 많은 강력한 기능과 알고리즘을 제공한다. 따라서 OpenCV 는 게임에서 이미지 데이터를 획득하고 처리하는 데 도움이 되며 이를 딥 러닝 모델에 전달하여 훈련 및 예측을 할 수 있다.

Flappy Bird 게임에서는 게임 인터페이스에서 지속적으로 스크린샷을 캡처하고 이를 입력 데이터로 심층 신경망에 전달하는 훈련이 필요한다. OpenCV 는 이미지 캡처, 크기 조정, 그레이스케일링, 이치화 등과 같은 간단하고 사용하기 쉬운 많은 이미지 처리 함수를 제공하여 게임 인터페이스에서 이미지 데이터를 쉽게 처리하고 심층 신경망 훈련에 적합한 데이터 형식으로 변환할 수 있다.

또한 OpenCV 는 에지 감지, 특징 추출, 표적 감지 등과 같은 일반적인 컴퓨터 비전 알고리즘을 구현하는 데 사용할 수 있으며 이러한 알고리즘은 게임에서 더 높은 수준의 인공지능 기능을 구현하는 데 도움이 된다. 따라서 컴퓨터 비전 라이브러리로 OpenCV 를 선택하면 개발 작업을 단순화하고 더 나은 게임 AI 효과를 얻을 수 있다.

3.1.3 Deep Learning Framework

TensorFlow 를 딥 러닝 프레임워크로 선택한 주된 이유는 딥 신경망 모델을 구축하고 훈련하는 데 적합한 널리 사용되는 오픈 소스 라이브러리이기 때문이다. TensorFlow 는 신경망의 정의, 훈련 및 예측에 사용할 수 있는 많은 강력한 기능과 알고리즘을 제공한다. 또한 TensorFlow 는 TensorBoard, tf.data 등과 같은 많은 실용적인 도구를 제공한다. Flappy Bird 게임에서 딥 러닝 모델은 적절한 게임 제어 신호를 생성하기 위해 게임 규칙과 플레이어 행동을 학습하는 데 사용된다. TensorFlow 는 게임 점수를 극대화하고 제어 전략을 최적화하기 위해 신경망 모델을 구축하고 훈련하는 데 도움이 되는 일련의 최적화 알고리즘 및 모델 아키텍처를 제공한다.

3.2 요구

3.2.1 게임 초기 상태 설정

```
GAME = 'bird' # 로그 파일 이름
ACTIONS = 2 #유효 오퍼랜드
GAMMA = 0.99 #감쇠율
OBSERVE = 100000. #이전 OBSERVE 라운드, 네트워크 트레이닝은 하지 않고 데이터만 모아 메모리에 저장
#OBSERVE에서 OBSERVE+EXPLORE 라운드 중 네트워크를 훈련하고 엡실론을 어닐링하여 엡실론을 FINAL_EPSILON으로
#EXPLORE 라운드에 도달하면 epsilon이 최종값 FINAL_EPSILON에 도달하여 더 이상 업데이트하지 않음
EXPLORE = 2000000. #상한
FINAL_EPSILON = 0.0001 #EPSILON 최종값
INITIAL_EPSILON = 0.0001 #EPSILON 초기값
REPLAY_MEMORY = 50000 #메모리
BATCH = 32 #훈련 로트
```

먼저 기초적인 수치틀을 설계한다

3.2.2 심층 신경망 모델의 정의와 훈련

```
    import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Conv2D

def create_q_model(input_shape, num_actions):
    model = tf.keras.Sequential([
        Conv2D(32, (8, 8), strides=(4, 4), activation='relu',
        input_shape=input_shape),
        Conv2D(64, (4, 4), strides=(2, 2), activation='relu'),
        Conv2D(64, (3, 3), strides=(1, 1), activation='relu'),
        Flatten(),
        Dense(512, activation='relu'),
        Dense(num_actions, activation='linear')
    ])
    return model
```

3 개의 컨볼루션 레이어와 2 개의 전체 연결 레이어를 포함하는 컨볼루션 신경망 모델이 정의되었다. 이 모델의 구조는 다음과 같다.

입력 레이어: Conv2D 레이어 정의를 사용하여 게임 인터페이스의 스크린샷을 입력하고 모양은 [None, input_shape[0], input_shape[1], input_shape[2]이며, 여기서 None 은 입력 샘플 수가 임의의 값일 수 있음을 나타낸다.

3 개의 컨볼루션 레이어: 범위 설정 수행

Flatten 레이어는 컨볼루션 레이어의 출력을 평평하게 하여 전체 연결 레이어에 쉽게 입력할 수 있다.

전체 연결층 1: 활성화 함수는 ReLU 인 512 개의 뉴런을 사용한다.

전체 연결 레이어 2: 출력 레이어, num_actions 뉴런을 사용하고 활성화 함수는 선형 활성화 함수(linear)이다.

이 중 num_actions 는 선택적 동작 수이며, 이 항목에서 num_actions=2 는 점프와 비점프의 두 가지 동작만 있음을 나타낸다.

3.2.3 훈련 정책을 설정

경험 재생기(Replay Buffer)를 사용하여 게임의 샘플 데이터를 저장할 수 있다. 각 훈련 라운드에서 심층 신경망 모델을 훈련하기 위해 경험적 재생기에서 샘플 배치를 무작위로 선택했다.

각 훈련 샘플에 대해 회색 그래프로 변환하고 지정된 크기로 축소해야 한다. 그런 다음 심층 신경망 모델의 매개변수를 업데이트하기 위해 DQN(Deep Q-Network) 알고리즘을 사용하여 Q 값과 목표 Q 값을 계산한다.

훈련 동안 DQN 모델과 표적 네트워크(target_q_model)의 매개변수를 동시에 업데이트했다. 특정 라운드마다 DQN 모델의 매개변수를 목표 네트워크에 할당하여 Q 값의 목표 값을 계산한다. 이렇게 하면 훈련을 더욱 안정시킬 수 있다. 구체적인 훈련 모델과 오차 함수는 계속 테스트해야 한다

3.2.4 머신러닝의 훈련 데이터를 보존

이 프로젝트에서 훈련된 데이터는 주로 경험적 재생기를 통해 관리 및 저장됩니다.경험 재생기는 현재 상태, 실행된 동작, 보상, 다음 상태, 게임 종료 여부 등 게임 상태 전환을 저장하는 버퍼다.

3.3 network 구성

```
def createNetwork():
    W_conv1 = weight_variable([8, 8, 4, 32])
    b_conv1 = bias_variable([32])

    W_conv2 = weight_variable([4, 4, 32, 64])
    b_conv2 = bias_variable([64])

    W_conv3 = weight_variable([3, 3, 64, 64])
    b_conv3 = bias_variable([64])

    W_fc1 = weight_variable([1600, 512])
    b_fc1 = bias_variable([512])

    W_fc2 = weight_variable([512, ACTIONS])
    b_fc2 = bias_variable([ACTIONS])

    s = tf.placeholder("float", [None, 80, 80, 4])
```

```
    h_conv1 = tf.nn.relu(conv2d(s, W_conv1, 4) + b_conv1)
    h_pool1 = max_pool_2x2(h_conv1)

    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2, 2) + b_conv2)

    h_conv3 = tf.nn.relu(conv2d(h_conv2, W_conv3, 1) + b_conv3)

    h_conv3_flat = tf.reshape(h_conv3, [-1, 1600])

    h_fc1 = tf.nn.relu(tf.matmul(h_conv3_flat, W_fc1) + b_fc1)

    readout = tf.matmul(h_fc1, W_fc2) + b_fc2

    return s, readout, h_fc1
```

```
def trainNetwork(s, readout, h_fc1, sess):
    a = tf.placeholder("float", [None, ACTIONS])
    y = tf.placeholder("float", [None])
    readout_action = tf.reduce_sum(tf.multiply(readout, a),
reduction_indices=1)
    cost = tf.reduce_mean(tf.square(y - readout_action))
    train_step = tf.train.AdamOptimizer(1e-6).minimize(cost)

    game_state = game.GameState()

    D = deque()

    a_file = open("logs_" + GAME + "/readout.txt", 'w')
    h_file = open("logs_" + GAME + "/hidden.txt", 'w')

    do_nothing = np.zeros(ACTIONS)
    do_nothing[0] = 1
    x_t, r_0, terminal = game_state.frame_step(do_nothing)
    x_t = cv2.cvtColor(cv2.resize(x_t, (80, 80)), cv2.COLOR_BGR2GRAY)
    ret, x_t = cv2.threshold(x_t,1,255,cv2.THRESH_BINARY)
    s_t = np.stack((x_t, x_t, x_t, x_t), axis=2)

    saver = tf.train.Saver()
    sess.run(tf.initialize_all_variables())
    checkpoint = tf.train.get_checkpoint_state("saved_networks")
    if checkpoint and checkpoint.model_checkpoint_path:
        saver.restore(sess, checkpoint.model_checkpoint_path)
        print("Successfully loaded:", checkpoint.model_checkpoint_path)
```

```
    else:
        print("Could not find old network weights")

    epsilon = INITIAL_EPSILON
    t = 0
    while "flappy bird" != "angry bird":
        readout_t = readout.eval(feed_dict={s : [s_t]})[0]
        a_t = np.zeros([ACTIONS])
        action_index = 0
        if t % FRAME_PER_ACTION == 0:
            if random.random() <= epsilon:
                print("----------Random Action----------")
                action_index = random.randrange(ACTIONS)
                a_t[random.randrange(ACTIONS)] = 1
            else:
                action_index = np.argmax(readout_t)
                a_t[action_index] = 1
        else:
            a_t[0] = 1

        if epsilon > FINAL_EPSILON and t > OBSERVE:
            epsilon -= (INITIAL_EPSILON - FINAL_EPSILON) / EXPLORE

        x_t1_colored, r_t, terminal = game_state.frame_step(a_t)
        x_t1 = cv2.cvtColor(cv2.resize(x_t1_colored, (80, 80)),
cv2.COLOR_BGR2GRAY)
        ret, x_t1 = cv2.threshold(x_t1, 1, 255, cv2.THRESH_BINARY)
        x_t1 = np.reshape(x_t1, (80, 80, 1))
        #s_t1 = np.append(x_t1, s_t[:,:,1:], axis = 2)
        s_t1 = np.append(x_t1, s_t[:, :, :3], axis=2)

        D.append((s_t, a_t, r_t, s_t1, terminal))
        if len(D) > REPLAY_MEMORY:
            D.popleft()

        if t > OBSERVE:
            minibatch = random.sample(D, BATCH)

            s_j_batch = [d[0] for d in minibatch]
            a_batch = [d[1] for d in minibatch]
            r_batch = [d[2] for d in minibatch]
            s_j1_batch = [d[3] for d in minibatch]

            y_batch = []
```

```
            readout_j1_batch = readout.eval(feed_dict = {s :
s_j1_batch})
            for i in range(0, len(minibatch)):
                terminal = minibatch[i][4]
                if terminal:
                    y_batch.append(r_batch[i])
                else:
                    y_batch.append(r_batch[i] + GAMMA *
np.max(readout_j1_batch[i]))

            train_step.run(feed_dict = {
                y : y_batch,
                a : a_batch,
                s : s_j_batch}
            )

        s_t = s_t1
        t += 1

        if t % 10000 == 0:
            saver.save(sess, 'saved_networks/' + GAME + '-dqn',
global_step = t)

        state = ""
        if t <= OBSERVE:
            state = "observe"
        elif t > OBSERVE and t <= OBSERVE + EXPLORE:
            state = "explore"
        else:
            state = "train"

        print("TIMESTEP", t, "/ STATE", state, \
            "/ EPSILON", epsilon, "/ ACTION", action_index, "/ REWARD",
r_t, \
            "/ Q_MAX %e" % np.max(readout_t))
```

- DNQ logic:

  메모리 D 의 메모리 초기화

  랜덤 가중치 θaction 값을 초기화하는 함수 Q(Q 추정)

  초기화 가중치 θ-= θ target-action 값의 함수^Q(Q 현실

  LOOP:

첫 번째 장면 s1=x1 을 초기화하고 장면 s1 에 해당하는 장면 처리 함수 Φ 를 전처리합니다

LOOP:

가능성 ε 에 따라 임의의 동작 at,or 를 선택합니다.

또는 함수 Q 에서 장면 st 아래에서 최대값 at 를 선택합니다.

액션 a 를 에뮬레이터에 실행하고 보너스 rt 와 다음 장면 xt+1 을 가져옵니다.

명령 st+1=st,at,xt+1 그리고 처리 Φt+1=Φ(st+1)

(Φt,at,rt,Φt+1)를 D 에 저장

D 에서 무작위로 작은 배치의 훈련을 샘플링합니다.

yj 값 설정:

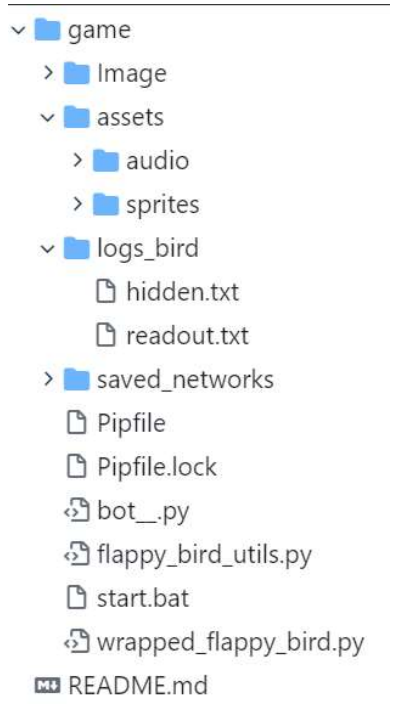
다음 장면 yj+1 이 중지된 경우: rj 만 반환됨

그렇지 않으면 rj+ (감마 ^Q(Φj+1,a,θ) 함수의 최대 a 값을 반환합니다.)

(Q 현실-Q 추정) 제곱 구배 회귀 분석을 수행하여 가중치 θ 를 업데이트합니다.

실행할 때마다 ^Q=Q(Q 현실=Q 추정, 주로 가중치 복사)

3.4 구현



assets: 게임에 필요한 멀티미디어

logs_bird: 로그 저장

saved_networks: 훈련 세트가 저장되고 훈련된 데이터 세트가 포함됩니다.만약 새로운 훈련이 필요하다면, 폴더를 비울 수 있다.

Pipfile, pipfile.lock: pipenv 환경 설치 패키지

bot__.py: 신경망 및 훈련 전략을 구성한다

wrapped_flappy_bird.py : 훈련에 사용되는 게임 파일

flappy_bird_utils.py :게임 파일에 필요한 내용을 로드한다

start.bat: 환경설치가 완료된 후 빠른 부팅 파일, 명령어(pipenv run python bot_.py)를 통해서도 시작할 수 있다.

4. 프로젝트 결과

4.1. 연구 결과

이 프로젝트는 신경망의 구성 세부 사항과 전략 선택이 중요한다. 결국 데이터 트레이닝을 위해 욕심 전략을 선택했다.

- DQN:

학습 과정에서 Memory 공간을 설정하는데, 이 공간은 각각의 MDP 과정, 즉 <s,a,r,s'>를 잘 기록합니다. 처음에 Memory 는 먼저 기록을 수집하고 기록이 일정 수에 도달하면 학습을 시작한다. 매번 메모리에서 적절한 크기의 메모리 블록을 무작위로 선택한다. 이러한 메모리 블록에는 경험(experience)도 포함되어 있으며 랜덤으로 선택되기 때문에 기록 상관성의 문제가 해결된다. 이러한 경험 중 s 를 입력으로 EvaluationNet 에 도입하여 q_evaluation 을 계산하고 s'tgar_Net 에 전달한다.

4.2. 성능평가

5 만 번을 하면 계속 위로 올라가거나 아래로 내려가기만 한다.

20 만 번, 대략적인 방향이 잡히면 첫 번째 파이프라인을 우회해 보겠습니다.가끔 통과할 수 있다.

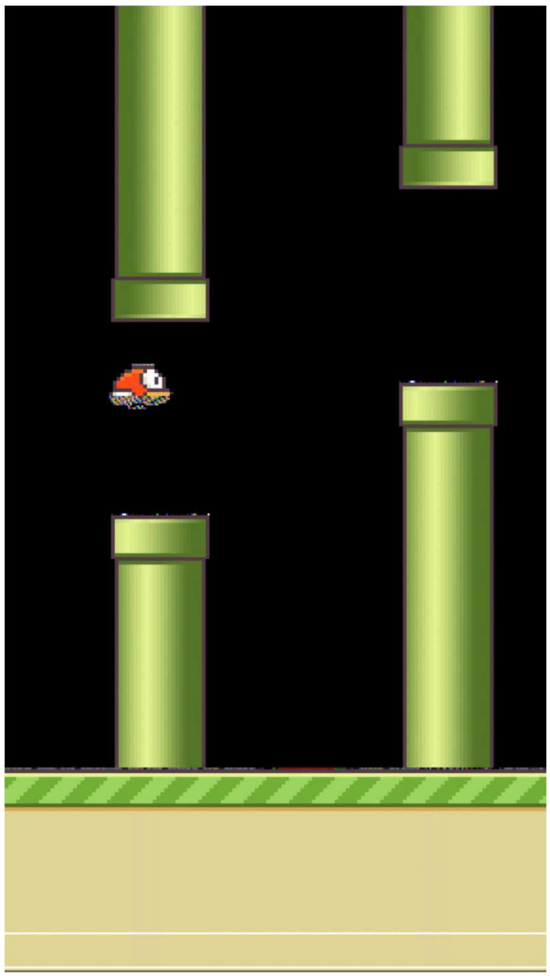
30 만 번, 파이프의 틈새를 거의 정확하게 찾았고 때로는 첫 번째 파이프를 통과할 수 있었다.

40 만 번이면 첫 번째 파이프를 통과할 확률이 높으며 가끔 두 번째 파이프를 통과할 수 있다.

100 만 번으로 일반 게이머 수준이며 평균 5-10 개의 파이프를 통과할 수 있다.

200 만 번이면 모든 파이프를 뛰어넘을 수 있고 거의 실수 없이 작은 확률로 실패한다.

250 만 회, 20 분 동안 한 번에 실수 없이 진행된다.



5. 결론 및 기대효과

5.1. 기대효과

신경망의 기본 구축을 완료한 후. 20 시간의 훈련을 거쳐 최종적으로 훈련 데이터를 얻었습니다. 트레이닝 세트 데이터를 로드하면 게임 도중 실수하지 않습니다.소기의 효과를 보다.

5.2. 추후 연구 방향

이 프로젝트의 주요 목적은 딥러닝 기술을 사용하여 자동으로 Flappy Bird 게임을 플레이할 수 있는 AI 모델을 훈련시켜 게임 지능에 대한 딥러닝의 응용을 보여주는 것이다. 딥러닝 기술을 사용하면 게임 AI 성능을 효과적으로 향상시킬 수 있어 AI 가 자체 훈련을 통해 **자신**의 의사 결정 전략을 지속적으로 최적화하여 인간 이상의 게임 성능을 달성할 수 있을 것으로 예상된다.

나는 프로젝트의 데이터와 신경망 설정을 보완하여 데이터 API 를 설정하기를 원한다. 보다 간단한 형식의 데이터 설정을 통해 다양한 게임에 적합한 AI 를 훈련할 수 있다. 그들을 전환하려면 데이터 프로필 하나만 있으면 된다. 또한 데이터 세트를 최적화하여 프로필을 보다 간편하고 편리하게 사용할 수 있다.

6. 참고문헌

[1] Why Study Games ai : https://www.engati.com/blog/ai-in-gaming

[2]Flappy Bird : https://namu.wiki/w/Flappy%20Bird

[3]The World's Hardest Game : https://namu.wiki/w/The%20World's%20Hardest%20Game

[4]Dqn algorithm : https://blog.csdn.net/Zhang_0702_China/article/details/123423637

[5]Genetic Algorithm : https://blog.csdn.net/LOVEmy134611/article/details/111639624

https://en.wikipedia.org/wiki/Genetic_algorithm

https://www.geeksforgeeks.org/genetic-algorithms/