목차

1. Background
2. Problem Definition
3. Environment Description
4. Build MDP Model
5. Example
6. Reinforcement Learning
7. Pseudo Code
8. Pseudo Test
9. Reference

1. Background

강화학습으로 푼 미로 문제는 이전까지 많은 시도와 연구가 있었다. 그 중 구체적인 예로서 Samyzaf 의 machine learning 블로그에서 Feedback system(rewards and penalties) 을 agent 에 적용하여 rat-cheese maze 문제를 강화학습으로 푼 사례가 있었다. 강화학습을 이해하기 위한 예시로서 보통 미로 문제는 널리 적용되고 있다. 또한 2014 년에 알파고를 선보인 구글의 인공지능 자회사 딥마인드가 Deep Q-Networks (DQN)을 이용하여 아타리 게임을 푼 사례가 있다. 이 중 Pacman 이라는 미로 문제와 비슷한 게임에서도 DQN 알고리즘은 인간 수준의 퍼포먼스를 보여주었다. 더 나아가 아타리 57 개의 모든 게임에서 인간의 능력을 추월하는 성과를 보였으며 전종목에서 인간 최고수를 뛰어넘는 능력을 구현하였다. 이 내용은 논문 [Agent57: Outperforming the Atari Human Benchmark]에서 확인할 수 있다. 이 후에 이러한 연구를 바탕으로 바둑인공지능 알파고가 2016 년에 <네이처>에 소개되었고 알파스타(스타크래프트용 인공지능), 알파제로 등으로 적용 범위를 확대한 인공지능의 개발로 이어졌다.

2. Problem definition

Sequential decision model 중 Grid world 의 응용으로 미로 찾기 게임을 주제로 선택하였다. Grid world 에서 말인 토끼가 입구에서 출구까지 장애물과 호랑이를 피해 가장 짧은 길을 찾아 탈출하는 것으로 maximum total reward 을 받는 것을 목표로 한다. 이 게임은 Q-learning with deep neural networks(DQN) 의 강화학습으로 푼다.토끼는 마지막 state 에 도착하기 위해 일련의 state 을 거쳐가며 움직여야 하고 Exploitation 과 Exploration 을 통해서 과거의 에피소드의 경험을 바탕으로 목표를 이룬다. 토끼는 실패를 반복하겠지만 수많은 시도를 통해서 학습할 것이고 문제를 풀게 될 것이다. 문제를 푼다는 것은 agent 인 토끼가 가장 최적의 route 를 찾아서 total reward 를 최대로 하며 게임에서 이기는 것이다. 최적의 route(optimal sequence of states)를 찾기 위해 토끼는 매움직임에서 penalty 를 조금씩 받게 될 것이다.

3. Environment description

10*10 matrix 셀(cell)의 환경 에서 Agent 는 토끼이며 토끼가 지나다닐 수 있는 길은 하얀색 셀이며. 파란색으로 된 셀은 장애물으로 지날 수 없다. 토끼의 적인 호랑이는 하얀색 셀을 랜덤으로 지나 다니고 토끼가 지나갈 수 있는 길은 호랑이도 지나다닐 수 있다. 토끼가 호랑이를 한 셀에서 만난다면 토끼는 호랑이에게 잡아 먹혀 죽게 된다. Target 셀은 출구 이고 토끼의 목적은 호랑이를 잘 피해 가장 단 거리로 출구에 도착하는 것이다. 토끼는 호랑이를 피해 가장 단거리로 출구에 도착하기 위해 exploitation 과 Exploration 을 한다. 또한 토끼의 미로 문제에서 토끼가 출구를 찾지 못하고 무한 루프에 빠지는 것을 방지하기 위해 여러가지 조건을 설정한다. 위 내용을 정리하면 아래와 같다.

환경

-10*10 matrix Cell
-Occupied cell→장애물
-Free cell→ 토끼가 지나다닐 수 있는 길
-Target cell→ 출구
-Enemy→호랑이가 랜덤으로 움직임
-Exploitation→움직임의 90%는 policy 에 따른다.
-Exploration→10%의 케이스는 새로운 경험을 축적하기 위해 랜덤으로 행동을 선택한다. 즉 10 개의 행동 중 1 개는 랜덤이다.

조건

-토끼의 움직일 수 있는 방향은 좌,우,위, 아래 이다.
-토끼의 시작점은 free cell 중 어느 곳이나 될 수 있으며 랜덤이다.
-토끼가 움직일 수 있는 최대의 움직임은 100 칸이다.
-호랑이의 움직임은 랜덤이다. 토끼를 잡아 먹을 경우 게임은 끝이 난다.
-토끼는 점점 호랑이를 피해 가장 단거리로 출구에 도착하는 길을 찾도록 학습되어진다.

4. Build MDP(Markov Decision Process) Model

토끼 미로 문제를 MDP(Markov Decision Process)로 푼다면 Agent, State, Action, Reward, Transition Probability, Discount factor 등으로 정의할 수 있고 아래와 같다.

→ Agent

토끼, 호랑이

→ State

S: state 는 토끼의 셀 위치로 정의된다. 토끼의 다음 state 를 정할 때 토끼의 현재의 state, 토끼의 액션, 호랑이의 위치, 호랑이의 액션, 장애물의 위치, 벽의 위치가 고려된다. 토끼의 $t+1$ 시점 state 는 아래와 같이 정의할 수 있다.

S= $[s_1, s_2 \dots s_t]$ (0<=t<=100)

s_{t+1}=$(s_t, a_{1t}, d_t, a_{2t}, B, C)$

D: 호랑이의 셀 위치이다. 호랑이도 토끼와 마찬가지로 움직임을 갖기 때문에 각 state 를 갖는다. 토끼와 호랑이의 움직임의 속도가 같다는 조건 아래 토끼의 각 state 에 따라 호랑이의 state 즉 위치로 정의하였다.

D=$[d_1, d_2 \dots d_t]$ (0<=t<=50)

B: 장애물의 위치 (6.Example 의 grid world 기준으로 작성하였다)

B =$[b_{24}, b_{28}, b_{45}, b_{27}, b_{57}, b_{59}, b_{45}, b_{27}, b_{71}, b_{73}, b_{710}, , b_{96}, , b_{99}]$

C:벽의 위치(벽의 총 개수는 40 개이다.)

C=$[c_1, \dots, c_{40}]$

*B 와 C 는 위치의 변화가 없다.

→Action

토끼가 매 state 에서 취하는 행동

A1= $[a_{11} a_{12}, a_{13}, a_{14}]$

호랑이가 state 에서 취하는 행동

A2=[$a_{21}a_{22}, a_{23}, a_{24}$]

→ transition probability

현재 state s 에서 토끼가 action a_{1t}를 취하고 호랑이가 a_{2t} 를 취하고 벽과 장애물의 위치를 고려할 때 다음 스테이트는 s_{t+1}=(s_t, a_{1t}, d_t, a_{2t}, B, C) 이다. 이 때 transition probability 는 만약 토끼가 어떤 움직임이 더 좋은 지 학습하게 된다면 더 좋은 액션을 취하기 위해 확률은 업데이트되고 바뀔 수 있다. transition probability 을 식으로 표현하면 아래와 같다.

$P(s'|s, a)$

→Reward

R=[R(s_1, a_1)+R(s_2, a_2)+⋯+R(s_t, a_t)]
r_{t+1} =R(s_t, a_t)

Reward 는 토끼에게만 부여된다. Reward 는 -1 에서 1 까지의 범위의 제한을 두고 구체적인 조건은 아래와 같다.

-각각의 state 에서 reward 가 존재한다
-바로 옆 칸으로 이동할 경우 -0.04 를 부여한다. 토끼가 길을 잃고 맴도는 것을 방지하고 가장 최단거리로 출구로 나가게 하기 위함이다.
-토끼가 출구를 찾았을 때 1 을 부여한다
-토끼가 장애물에 부딪혔을 때 -0.05 를 부여한다
-토끼가 벽에 부딪혔을 때, -0.075 를 부여한다.
(*만약 벽에 부딪혔을 때 Pause 를 주어서 시간상 리워드를 추가하게 되면 매우 복잡해지기 때문에 pause 를 주지 않고 감점 값을 낮추는 방향으로 진행하였다.)
-토끼가 이미 왔던 셀에 또 온다면 -0.02 를 부여한다
-토끼가 호랑이에게 잡아 먹히면 -1 로 자동으로 게임이 끝난다.
-토끼가 100 번을 움직였을 때 자동으로 게임이 끝난다.
-호랑이의 움직임은 50 번으로 제한한다. 게임이 너무 일찍 끝나는 것을 방지하기 위함이다.

→Optimal Policy

$$\pi^*(s) = \arg max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

정책에 관해서는 $\pi(s)$ 은 토끼가 현재 state S 에서 취할 수 있는 가장 최적의 액션을 나타낸다. 각 state 에서 여러가지 정책을 계산하고 가장 최적의 정책을 도출하여 주어진 state 에서 토끼가 가장 좋은 결과를 낼 수 있도록 한다. 최적의 정책(optimal policy)는 $\pi^*(s)$으로 표현될 수 있다.

→Discount factor γ

할인율 γ은 transition model 에 적용할 수 있고, 미래 보상 가치를 뛰어넘는 현재 가치라고 정의할 수 있고 할인율이 0 에 가깝다면 미래 보상은 현재 보상 만큼 중요하지 않다.

→Utility

토끼가 최적의 보상을 찾기 위해 value iteration 알고리즘을 적용한다. Bellman Equation 에 따라서 아래 유틸리티는 아래와 같이 풀이할 수 있고 아래 식과 같이 표현한다.

The immediate reward for the state + The discounted utility of the next state multiplied by the maximum utility of the next state.

$$\mathrm{U}(s) = \mathrm{R(s)} + \gamma\, max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

→Bellman Update

$$U_{i+1}(s) \leftarrow R(s) + \gamma\, max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

다음 스텝으로 유틸리티는 계속해서 반복되며 마지막에 최대 유틸리티를 얻기 위해 각 스테이트에서는 equilibrium(평형 or 균형)이 필요하다. 아래의 알고리즘은 policy evaluation 과 policy improvement 사용한다. policy evaluation 은 앞 스테이트에서 주어진 정책의 가장 최대의 값(maximum utility of a given policy from the previous state)을 적용하는 것이며 policy improvement 는 한 단계 앞서(one-step ahead from U_i) 새로운 최대 유틸리티 정책(new maximum expected utility policy)을 계산하는 것이다. 더이상 유틸리티값의 변화가 없다면 업데이트를 멈춘다.

5. example

시작	→ -0.04	↓ −0.04							
		→↓ -0.05 -0.04							
		↓ -0.04							
	↓ -0.04	← -0.04							
	↓ -0.04								
	↓ -0.04	→ -0.04	→←→ -0.04- 0.02-0.04	○ → -0.04	○ -1	-			
									출구

각 칸의 크기가 같다고 가정한다. 토끼는 초록색 , 호랑이는 빨간색이다. 위의 경우에서 화살표는 토끼가 움직인 뱡향이라고 두면 아래와 같이 최종 리워드를 계산할 수 있을 것이지만 상태변환확률을 모르기 때문에 그 값은 컴퓨터가 계산해 줄 것이고 인간의 능력으로 알 수 없을 것 같다.

토끼의 위치까지의 리워드:

-0.04*p1-0.04 *p2-0.05*p3-0.04*p4-0.04*p5-0.04*p6-0.04*p7-0.2*p8-0.04*p9-0.04*p10-0.04*p11-0.02*p12-0.04*p12-0.04*p13-0.02*p13-0.04*p14 = ?

다음 state 에서호랑이와 마주쳐서 잡아 먹혔을 때 리워드:

-1 감점으로 자동으로 게임 오버된다.

6. Reinforcement Learning

Q-learning 적용

모든 state 의 Q function 을 테이블의 형태로 저장하고, 모든 state 의 Q function 을 방문할 때마다 하나씩 업데이트하는 방법으로 강화 학습을 만든다. 각 State 는 점수의 득점, 호랑이의 위치, 토끼의 위치 모두 고려하여 바뀐다는 점과 토끼가 매 회 episode 를 반복할 수록 학습하고 스마트해진다. 그리하여 토끼는 출구까지 최적의 루트를 찾아낼 것이다.

Q value

토끼는 게임의 마지막에서 total reward 을 최대로 얻기 위하여 a series of optimal actions 을 취한다. Q value 는 현재의 reward 에 현재부터 마지막 state 의 최대 보상의 기대 값을 더하여 다음의 Q value 를 업데이트하여 학습한다. 식으로 표현하면 아래와 같다.

$$Q(s_{t+1}|s_t + a) = r(s_t, a) + \mathrm{E}\sum_{k=1}^{j} \gamma^k \, r(s_{t+k}, a_{t+k})$$

(1<j<=100) : j 는 각각의 에피소드에서 토끼의 움직임에 따라 바뀔 수 있다.

Q value 는 아래와 같은 rule 로 업데이트 된다.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max Q(s', a') - Q(s - a))$$

α =learning rate
γ = discount rate

$Q(s,a)$:state s 에서 action a 를 취했을 때 얻을 수 있는 최대의 종합적 보상(the maximum total reward we can get by choosing action a in state s)

토끼가 각 스테이트에서 취 할 수 있는 가능한 액션들을 모두 취하고 가치(value)를 숫자의 형태로 나타내준다. 가치가 현재 스테이트만 적용하는 것이 아니라 마지막에 얻을 보상을 고려한다. 각 시뮬레이션에서 이 가치들은 업데이트 된다. Q-function 에서는 독립적으로 선택한 정책(policy)을 바탕으로 최적의 정책(optimal value)를 바로 찾는다. 이러한 방법은 state 와 action 이 한 쌍으로서 계속적으로 업데이트되야한다는 조건만 충족하면 알고리즘은 분석(analysis)와 융합(convergence)을 매우 간단하게 만든다.

7. Pseudo Code

변수목록

```
# 토끼가 지나간 길은 표시. 수치는 리워드 값이 아님
visited_mark = 0.8

# 토끼의 현재 위치를 표시
rabbit_mark = 0.5

# 호랑이의 현재 위치를 표시
tiger_mark = 0.3

# 토끼 액션은 좌,우, 위, 아래
r_LEFT = 0
r_UP = 1
r_RIGHT = 2
r_DOWN = 3

# 호랑이 액션은 좌,우, 위, 아래
t_LEFT = 0
t_UP = 1
t_RIGHT = 2
t_DOWN = 3

#장애물과 벽은 그리드 워드 매트릭스로 표현 장애물=1, 벽=0.5, free cell=0
matrix = [[0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]
          [0.5,0,0,0,0,0,0,0,0,0,0.5],
          [0.5,0,0,0,1,0,0,0,1,0,0,0.5],
          [0.5,0,0,0,0,0,0,0,0,0,0,0.5],
          [0.5,0,0,0,0,1,0,0,0,0,0,0.5],
          [0.5,0,0,0,0,0,0,1,0,1,0,0.5],
          [0.5,0,0,0,0,0,0,0,0,0,0,0.5],
          [0.5,1,0,1,0,0,0,0,0,0,1,0.5],
          [0.5,0,0,0,0,0,0,0,0,0,0,0.5],
          [0.5,0,0,0,0,0,1,0,0,1,0,0.5],
          [0.5,0,0,0,0,0,0,0,0,0,0,0.5],
          [0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]]

#토끼 액션의initial 값은 0이다.
r_num_actions = 0

#호랑이 액션의initial 값은 0이다.
t_num_actions = 0

# Exploration rate는 0.1으로 둔다.
epsilon = 0.1

# episodes는 0으로 initialize 한다.
episodes =0;
```

Pseudo Code

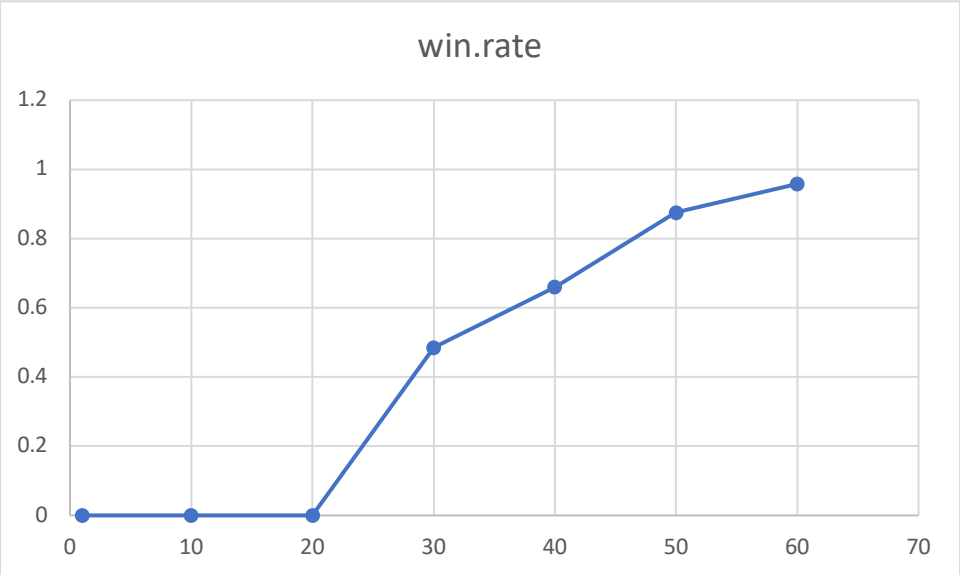
```
Initialize Q(s,a) arbitrarily;
Repeat
    #토끼의 액션 횟수는 100번으로 한정한다.
    if r_num_actions =<100:

        #호랑이의 액션 횟수를 50번으로 한정한다.
        if t_num_action>50:
            #호랑이 사라짐
            continue;

            Initialize s;

            if Q(s, a)  >-1:

                repeat
                    Choose a from s using policy derived from Q;
                    Take action a, observe r, s’;
                    Q(s, a) ← Q(s, a) + α[r + γmaxa′ Q(s′, a′) − Q(s, a)];
                    s ← s′;

                until s is terminal;

            else if Q(s, a)  >= 1:
                                    win
                                    break;


            else:
                lose
                break;


    else:
        game over
        break;


until no more episodes;
```

8. Pseudo Test

실제로 위 알고리즘을 뉴럴네트워크로 학습시킨다면 아래와 같은 결과가 나올 것이다. 아래는 실제 알고리즘을 적용한 것이 아니라 가상의 예이다. Epoch 가 증가할 수록 토끼의 학습율도 그에 따라 높아질 것이고 게임에 이길 확률(win rate) 또한 커질 것으로 예상된다. 그래프를 그려보면 win rate 가 epoch 가 증가함에 따라 같이 증가하고 epoch 60 에 다 달았을 때 거의 win rate 가 1 에 가까우므로 토끼의 완전한 학습이 가능할 것으로 예상된다

```
model = build model(rabbitMaze)
qtrain(model, maze, epochs =1000, max_memory =10*maze.size, data_size=100 )
```

epoch	loss	Episodes	win count	win.rate	time(s)
1	0.0043	102	0	0	4.2
10	0.0052	10	5	0	26.3
20	0.0661	4	10	0	47.1
30	0.0029	102	13	0.485	74
40	0.0039	20	23	0.66	80.9
50	0.0016	6	32	0.875	91.6
60	0.0014	25	43	0.958	102



9. Reference:

http://www.hani.co.kr/arti/science/future/935551.html#csidx7d8fb4a5ec786eda3b29a491a4b9432
https://www.samyzaf.com/ML/rl/qmaze.html
https://tykimos.github.io/warehouse/2018-2-7-ISS_Near_and_Far_DeepRL_4.pdf
https://maelfabien.github.io/rl/RL_1/#
https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690
http://www.lix.polytechnique.fr/~ntziortziotis/pubs/SETN14.pdf
https://leyankoh.com/2017/12/14/an-mdp-solver-for-pacman-to-navigate-a-nondeterministic-environment/