블록 피하기 게임에서 강화 학습을 이용한 블록 피하기 에이전트 구현

이경호*, 강병섭O
O*한라대학교 정보통신방송공학과
e-mail: khlee@halla.ac.kr*, gurasackggi@naver.comO

Implementation of Reinforcement Learning Agent to Avoid Blocks in Block Avoidance Game

Kyong-Ho Lee*, Byong-Seop KangO
O*Dept. of Information Communication & Broadcasting Engineering, Halla University

● 요 약 ●

본 논문에서는 2차원 공간상에서 상부에서 하부로 떨어지는 블록을 하부에서 피하는 게임에서 강화 학습에 사용되는 DQN 알고리즘을 이용하여 블록 피하기 에이전트를 구현하고 학습 통해 점점 더 높은 점수를 받는 모습을 확인하였다. 파이썬을 이용하여 게임을 구현한 후 텐서플로우를 이용하여 DQN를 이용한 에이전트를 구현하였다. 에이전트는 보상을 통한 학습을 통하여 점점 강화되도록 하였는데, 초기에는 무작위로 움직였으나, 환경으로부터 받는 보상으로 점점 더 능숙하게 피하는 모습을 관찰할 수 있었다. 본 구현에서는 4000번 정도의 게임 시행에서 아주 능숙하게 피하는 결과를 얻을 수 있었다.

키워드: 인공지능(Artificial Intelligence), 기계학습(Machine Learning), 강화학습(Reinforcement Learning)

I. Introduction

최근 구글 딥마인드의 컴퓨터 바둑 인공지능 프로그램 알파고가 이세돌 9단과의 경기에서 이루어낸 4:1의 승리에 인공지능 학습에 대한 열풍이 불며, 딥러닝과 같은 방법으로 학습을 시키는 컴퓨터 프로그램 에 관심이 많아졌다. 알파고가 바둑의 규칙을 모르지만 직접 바둑을 두면서 어떻게 해야 상대방을 이길 수 있는지 스스로 학습을 하는데, 바둑의 규칙을 모르면서 학습할 수 있었던 것은 강화학습이라는 방법을 사용했기 때문이다.

강화학습 아이디어는 행동심리학에서 차용한 개념으로 '동물이 이전에는 배우지 않았지만 직접 시도하면서 행동과 그 결과로 나타나는 좋은 보상 사이의 상관관계를 학습하는 것'을 말한다. 어떤 행동이 왜 그런 결과로 이어지는지 동물이 이해는 못하더라도 학습자는 행동과 행동의 결과를 보상을 통해 연결하는 것이다. 이렇게 강화학습 아이디어는 간단하며, 컴퓨터 프로그램으로 연결되며 구현되고 있지만, 이 간단한 아이디어를 구체화하고 현실화하기는 쉽지 않다고 알려져 있다[1]. 강화학습을 프로그램으로 구현하기 위한 쉽게 설명한 자료가 많지 않아 이해하기도 쉽지 않으며, 이해한 후도 어떤 구조로 구현해야 하는지 알기 어렵고, 그래서 구현에 많은 시간과 노력이 필요하다.

본 논문에서는 블록 피하기 게임을 만든 후, 이 게임에 참여하는 에이전트에게 강화학습을 이용하여 학습되도록 만들고, 이 에이전트가 주어진 상황에서 하는 행동에 적당한 보상을 주어 학습이 되게 하고, 그래서 떨어지는 블록을 잘 피하게 되는 강화 학습 프로그램의 구현을 하였다. 본 구현을 목표로 수행하면서 [1]에서 주장하는 바와 같은 어려움을 겪었다. 위와 같은 어려움을 겪으며 이론을 배우는 것에 그치는 것이 아니라 게임을 만들어 게임 속에서 주인공을 학습시키는 구현을 할 때 어떤 점들을 고려해야 되는지 알아보았다.

II. Preliminaries

강화학습은 머신러닝의 한 부류이다. 머신러닝은 지도 학습과 비지도 학습, 준지도 학습, 강화 학습이 있다. 지도 학습은 정답을 알고 있는 데이터를 이용해 컴퓨터가 답을 말했을 때 맞고 틀림을 알려주어 컴퓨터가 학습되도록 하는 것이고, 비지도 학습은 정답이 없는 데이터들을 묶도록 하되 초기에 원하는 분류 집단만큼 임의 중심점을 부여하고 이를 중심으로 집단화한 후, 집단 내에서 새로운 중심점을 정하고 다시 분류하는 과정을 반복함으로서, 반복 수행을 통해 결국 비슷한 것끼리 묶이게 하는 식의 학습이다. 준지도 학습은 데이터 중 일부는 정답을 아는 것이고 나머지는 정답을 모르는 자료를 이용한 학습이다.

강화(Reinforcement) 학습은 보상을 통해 학습하는 것으로 보상은 컴퓨터가 수행한 행동(Action)에 대해 만들어진 환경(Environment)의 반응을 말한다. 보상(Reward)은 직접적인 답은 아니지만 컴퓨터에게는 간접적인 정답 역할을 한다. 지도 학습에서는 정답을 통해 오차를

계산해서 학습했지만 강화 학습에서는 자신의 행동 결과로 나타나는 보상을 통해 학습한다. 즉 컴퓨터는 강화처럼 보상을 얻게 하는 행동을 점점 많이 하도록 학습하는 것이다. 강화 학습을 통해 스스로 학습하는 컴퓨터 또는 프로그램을 에이전트(Agent)라고 한다. 물론 에이전트는 환경에 대한 사전 지식이 없는 상태에서 학습을 한다. 그래서 에이전트는 자신이 놓인 환경에서 주어진 상태를 해석(Interpreter)하여 인식한 후 처음에는 자신이 할 수 있는 행동을 한다. 그러다가 우연한 행동이 상황에 맞으면 환경은 에이전트에게 보상을 주고 다음 상태를 알려준다. 물론 에이전트는 다음 상태도 해석을 통해 인지한다. 이런 과정에서 에이전트는 보상을 통해 어떤 행동이 좋은 행동인지 간접적으로 알게 된다.

강화 학습은 '결정을 순차적으로 내려야하는 문제'에 잘 적용된다. 순차적 의사결정 문제란 자동차를 제어할 때 빗길, 비포장 도로 등의 원인으로 인해 우리가 원치 않는 대로 움직이는 상황에서도 안전하게 운전하기 위해서 매 순간, 연속적인 결정들을 내리고, 일련의 결정들의 결과로 얻어지는 상태들이 모두 안전해야 하는 문제들이다. 바둑이나 오목도 그런 문제라고 볼 수 있다. 이렇게 결정을 계속적으로 선택해야 하는 문제들을 '순차적 의사 결정 문제'라고 한다. 순차적 의사 결정 문제는 강화학습과 다이내믹 프로그래밍, 진화 알고리즘 등으로 해결한다. 그러나 다이내믹 프로그래밍과 진화 알고리즘은 각각 한계를 가지고 있으나 강화 학습이 그 한계를 극복할 수 있다고 알려져 있다[1].

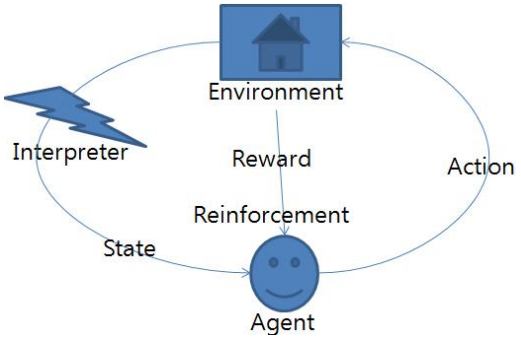

Fig. 1. Reinforcement Learning Structure

순차적 행동 결정 문제를 해결하기 위해서 문제를 해결할 에이전트가 학습하고 발전할 수 있게 문제를 수학적으로 표현해야 한다. 이때 Markov Decision Process(MDP)를 이용한다. MDP는 순차적 행동 결정 문제를 수학적으로 정의해서 에이전트가 순차적 행동 결정 문제에 접근할 수 있게 한다. 순차적 행동 결정 문제의 구성 요소는 상태(state), 행동(action), 보상(reward), 정책(policy)이 있다. 상태는 관찰에 의하여 얻어지는 정보로 정적인 요소와 동적인 요소가 결합된 것이다. 탁구를 치는 문제라면 여기에서 에이전트의 행동을 위한 자료는 탁구공의 위치와 자신의 배트 위치 뿐 아니라 속도, 가속도 등의 정보가 필요한 데 이것이 문제의 상태라고 볼 수 있다. 행동은 앞에서 얻은 상태를 바탕으로 취할 수 있는 행위를 말한다. 상/하/좌/우 이동 같은 것인데, 초기에는 학습이 되어 있지 않아 에이전트가 무작위 행동을 취하나 보상에 의한 학습이 되면서 특정 상태에 알맞은 행동을 할 확률이 높아진다. 순차적 행동 결정 문제에서는 에이전트가 행동을 하면 환경이 보상을 주면서 다음 상태를 알려준다. 보상은 강화학습의 핵심요소로 에이전트의 행동을 변화시킬 유일한 정보이다. 보상은 환경으로부터 나오며 이를 통해 에이전트는 자신을 평가할 수 있고, 어떤 행동이 좋은 행동인지 알 수 있다. 강화 학습의 목표는 순차적 행동 결정 문제에서 해인 '시간에 따라 얻는 보상들의 합을 최대로 하는 정책'을 찾는 것이다. 중요한 것은 보상은 에이전트가 아니라 환경에 속하며, 어떤 상황에서 얼마의 보상이 나오는지는 에이전트는 알지 못하나 환경에 의한 보상으로 학습되어 진다. 에이전트는 보상을 얻으려면 행동을 해야 하는데 최종 목표는 특정 상태가 아닌 모든 상태에 대해 어떤 행동을 해야 하는지 알아야 하는 것이다. 이렇게 모든 상태에 대해서 에이전트가 어떤 행동을 해야 하는지를 정해 놓은 것이 정책이다. 따라서 순차적 행동 결정 문제를 풀었다는 것은 발생된 상태에 따라 취한 행동이 보상의 합을 최대로 받은 것이다[2].

III. The Proposed Scheme

1. Game configuration and development environment

본 논문에서의 게임의 개발 환경은 Windows 10 Update(64-bit) 이며 개발 언어로는 Python을 사용하였고 알고리즘을 만드는데 사용한 라이브러리는 Tensorflow를 사용하였다.

Table 1. Development Environment

구 분	종 류	비 고
컴퓨터	데스크탑	CORE i7 7700-k
		OS Windows 10
		RAM 16GM
언어	Python	Anaconda
라이브러리	Tensorflow, matplotlib	게임화면 구성 및 알고리즘 구성

블록 피하기 게임 구성 화면은 Fig. 2과 같이 구성 하였다. 빨간색과 파란색 블록은 상단의 임의 위치에서 출발하여 서로 다른 속도로 떨어져 내려오는데 초록색 블록은 정지 또는 좌우로 이동하며 빨간색과 파란색 블록을 피하는 게임이다. 초록색 블록이 파란색과 빨간색 블록 중 어느 것과라도 부딪히게 되면 게임은 종료 된다.

게임판과 게임판에서의 블록들의 위치와 그리고 떨어지는 블록들의 속도 등은 환경에 해당하는 정적, 동적 정보이다.

에이전트에 해당하는 초록색 블록은 행동으로 좌우로 움직이거나 또는 정지할 수 있는데, 이런 행동이 빨간색과 파란색 블록들을 피하게 되면 내려오는 블록을 피할 때 마다 보상 점수가 주어지는데 처음에는 무작위로 움직이는 행동을 하나 보상에 의해 학습이 됨에 따라 잘 피하게 되며 그래서 점수는 올라가게 된다. 물론 초록색 블록이 상단의 빨간색 블록 또는 파란색 블록과 충돌하면 나쁜 보상을 받게 된다.

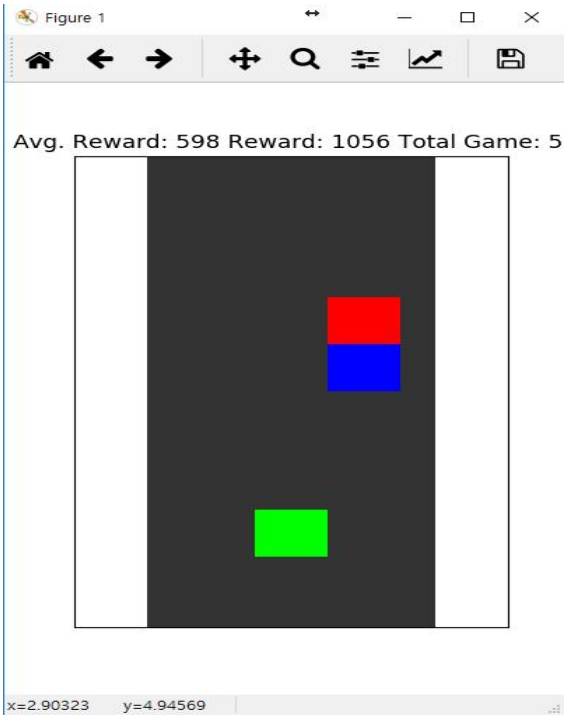

Fig. 2. Game Screen

좀 더 구체적으로 설명하면 본 게임을 위해 에이전트가 보게 되는 상태는 게임 판의 너비와 높이(6 x 10), 블록의 위치와 속도이다. 게임 판은 배열로 표현하였으며, 사물이 있으면 1, 없으면 0으로 수치화하였다. 행동으로 초록 블록이 좌/고정/우로 움직인다. 초록 블록에게 주는 보상은 떨어지는 블록에 부딪치지 않으면 얻는 보상(1~2점)과 움직임이 적을 경우 보상을 줘서 안정적으로 움직이게 하기위한 보상(1점), 장애물에 충돌할 경우(-2)점을 준다. 학습은 4프레임마다 한 번씩 학습하도록 하였다.

여기에서 사용한 인공 신경망은 Deep Q Network(DQN)이다. DQN으로 4프레임의 상태가 입력으로 들어오면 에이전트는 제자리, 왼쪽, 오른쪽 행동의 큐 함수를 출력으로 내놓는다. 에이전트는 출력한 큐 함수를 보고 큰 가치를 지니는 행동을 선택한다. 환경은 에이전트가 행동을 취하면, 에이전트에게 보상과 다음 상태를 알려준다. 에이전트는 환경과 상호작용하면서 DQN을 더 많은 보상금을 받도록 조금씩 조정한다.

2. Learning execution process and post-learning execution result

학습이 되는 모습을 관찰하기 위하여 그림2의 상단과 같이 평균 보상과 그 게임 횟수에서 보상을 확인하였다, 아래 그림은 게임 횟수별 취득한 보상을 추출하여 보인 것이다. 왼쪽은 초기 횟수별 취득한 보상 상태이며, 가운데는 500회 정도의 게임을 했을 때 횟수별 취득한 보상 상태이고, 오른쪽은 4860회 정도의 게임을 수행했을 때 횟수별 취득한 보상이다. 초기에는 학습이 되지 않아 보상 정도가 확연히 낮은 것을 알 수가 있다. 에이전트가 생각 없이 무작위로 동작할 뿐이며 그래서 -1과 -2의 보상이 많이 보인다. 그러나 게임 500회 정도로 가면 약간 학습이 되어 보상 점수가 음은 보이지 않는다. 본 구현 실험의 4860회 정도에서는 거의 3000점에 가까운 점수가 보이기도 한다.

게임횟수: 1 점수: 0	게임횟수: 491 점수: 0	게임횟수: 4851 점수: 13257
게임횟수: 2 점수: -1	게임횟수: 492 점수: 2	게임횟수: 4852 점수: 1715
게임횟수: 3 점수: 1	게임횟수: 493 점수: 0	게임횟수: 4853 점수: 11078
게임횟수: 4 점수: -2	게임횟수: 494 점수: 3	게임횟수: 4854 점수: 1451
게임횟수: 5 점수: 2	게임횟수: 495 점수: 2	게임횟수: 4855 점수: 18
게임횟수: 6 점수: 0	게임횟수: 496 점수: 6	게임횟수: 4856 점수: 193
게임횟수: 7 점수: 2	게임횟수: 497 점수: 0	게임횟수: 4857 점수: 399
게임횟수: 8 점수: -2	게임횟수: 498 점수: 6	게임횟수: 4858 점수: 221
게임횟수: 9 점수: -1	게임횟수: 499 점수: 3	게임횟수: 4859 점수: 263
게임횟수: 10 점수: 0	게임횟수: 500 점수: 10	게임횟수: 4860 점수: 146
게임횟수: 11 점수: 0	게임횟수: 501 점수: 2	게임횟수: 4861 점수: 80
게임횟수: 12 점수: 0	게임횟수: 502 점수: 3	게임횟수: 4862 점수: 216
게임횟수: 13 점수: 1	게임횟수: 503 점수: 9	게임횟수: 4863 점수: 417
게임횟수: 14 점수: 0	게임횟수: 504 점수: 0	게임횟수: 4864 점수: 686
게임횟수: 15 점수: 0	게임횟수: 505 점수: 2	게임횟수: 4865 점수: 368
게임횟수: 16 점수: 2	게임횟수: 506 점수: 0	게임횟수: 4866 점수: 103
게임횟수: 17 점수: 0	게임횟수: 507 점수: 0	게임횟수: 4867 점수: 29625
게임횟수: 18 점수: -2	게임횟수: 508 점수: 9	게임횟수: 4868 점수: 243
게임횟수: 19 점수: 0	게임횟수: 509 점수: 10	게임횟수: 4869 점수: 243
게임횟수: 20 점수: 1	게임횟수: 510 점수: 0	게임횟수: 4870 점수: 40

Fig. 3. Score by Game Round

본 게임 환경의 에이전트 구성에서는 5000번의 학습을 시킨 그 상태의 학습에서 게임을 실행 시켜본 결과 [그림 1]와 같이 약 5번 정도의 게임을 하였을 때 평균 점수가 598점 정도가 나오는 것을 확인 할 수 가 있었다.

IV. Conclusions

본 논문에서는 강화 학습을 통해 학습이 되는 에이전트가 블록 파하기 게임 환경에서 학습이 되어 잘 파하게 되는 상황을 확인하였다. 비록 강화학습 아이디어가 간단하며, 컴퓨터 프로그램으로 연결되어 여러 구현이 이루어져 있지만, 이 간단한 아이디어를 구체화하고 현실화하기는 쉽지 않다고 알려져 있는 의미를 실감하여 이루어 낸 결과이다. 강화학습을 프로그램으로 구현하도록 하는 쉽게 설명한 자료가 많지 않아 이해하기도 쉽지 않았으며, 이해한 후도 어떤 구조로 구현해야 하는지 알기 어려웠다. 그래서 구현에 많은 시간과 노력이 소요되었다.

현재 학습을 시키는 데에 있어서 학습량을 5000번의 학습을 시켜 높은 보상이 나오는 것을 확인 할 수는 있었지만 꾸준히 높은 보상이 나오는 것은 확인 하지 못하였다. 이런 이유로 좀 더 나은 개선이 필요하다. 또한 장애물의 확인을 좌표로 확인 했던 것을 다음에는 Convolutional Neural Network를 통한 이미지를 가지고 학습을 하는 것을 구현해 볼 필요도 느끼고 있다.

REFERENCES

[1] U. W. Lee, and H. R. Yang, G. W. Kim, Y. M. Lee, U.

R. Lee, "Reinforcement Learning," Wikibooks, pp.v, 2017.

[2] U. W. Lee, and H. R. Yang, G. W. Kim, Y. M. Lee, U. R. Lee, "Reinforcement Learning," Wikibooks, pp.7-58, 2017.