강화학습을 이용한 체스 인공지능

손병욱, 최지원, 홍수경, 박희민

Chess AI under Reinforcement Learning

Byungwook Son, Jiwon Choi, Sugyeong Hong, Heemin Park

요 약

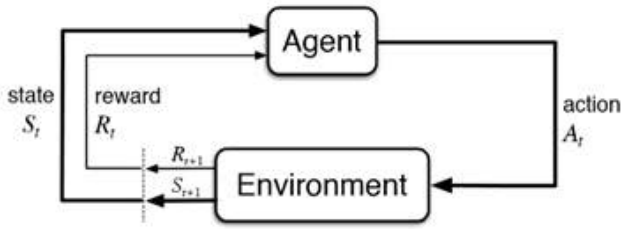
4차 산업혁명 시대가 도래하면서 인공지능 분야가 급속하게 발전함에 따라 우리 생활에서 다양한 형태로 접할 수 있게 되었다. 특히 게임 분야에서 알파제로와 같은 인공지능이 연구되고 있다. 이에 우리는 강화학습 알고리즘을 이용해서 사람처럼 게임하는 체스 인공지능을 개발하고자 한다. 2장에서 강화학습의 개념과 구현하고자 하는 체스 인공지능을 자세히 설명하고, 3장에서는 학습에 사용한 알고리즘과 모델의 구조, 학습 과정 등을 설명한다. 4장에서는 연구 결과를 설명한다.

Key words : Reinforcement Learning, Chess, Game AI, Epsilon-Greedy, MCTS

1. 서 론

4차 산업혁명 시대가 도래하면서 인공지능 분야가 급속하게 발전함에 따라 우리 실생활에서 흔하게 인공지능을 접할 수 있게 되었다. 특히 게임 분야에서는 알파 제로와 같은 인공지능이 많이 연구되고 있다[1]. 이에 맞게 우리는 바둑의 알파 제로와 유사한 체스 인공지능을 구현해보고자 한다[2]. 체스를 두는 인공지능 모델을 구현하고 강화학습 시켜 실제 게임 시 자신에게 유리한 수를 둘 수 있도록 하는 것이 본 프로젝트의 목표이다. 이번 프로젝트를 통해서 인공지능을 심층적으로 학습하면서 최근 개발된 다른 모델들을 이해할 수 있고, 또한 기존 개발되었던 모델들을 공부하고 장단점을 분석하여 더욱 발전된 모델을 개발할 수 있다.

2-1. 강화학습



<그림 1. 강화학습 모델>

강화학습이란 어떤 환경에서 정의된 에이전트(agent)가 현재 상태(state)에서 어떤 행동(action)을 취할 때 주어지는 보상(reward)을 근거로 학습하는 방식을 말한다[3]. 강화학습의 목적은 주어진 환경에서 최적의 선택을 하도록 하는 것이다. 에이전트는 매 시점(t)마다 상태(St)와 가능한 행동(St)을 가진다. 보상은 현재 상태와 행동에 대해서만 평가하는 즉각적인 값이기 때문에 장기적으로 최적의 보상을 얻기 위해서는 누적 보상이 가장 큰 행동을 골라야 한다. 즉, 에이전트가 현재 상태에 따른 정답 행동을 알 수는 없지만 무수히 많은 경험을 통해 최적의 선택을 학습할 수 있다.

2-2. 체스 인공지능

우리가 개발하고자 한 체스 인공지능은 강화학습 알고리즘을 통해 학습된 모델이 실제 체스 게임에서 마치 사람처럼 자신에게 유리한 수를 두며 게임을 하는 모델이다. 체스 게임에서 강화학습을 이용해 학습하는 과정은 이렇다. 모델이 현재 상태(s)인 보드에서 행동(a)을 취하는데, 이때 행동 집합으로는 게임 규칙에 위배되지 않고 움직일 수 있는 말의 움직임들로 구성된다. 행동(a)을 취해 그에 대한 보상을 받게 되는데, 이는 한 게임이 끝나면 보상 갱신 식에 의해 상태-행동에 대한 보상 값이 주어진다. 체스 인공지능 모델은 무수히 많은 학습을 통해 최선의 선택을 할 수 있다.

3. 강화학습 기법

본 프로젝트에서는 강화학습을 통해 모델을 학습시키기 위해 여러 가지 강화학습 기법을 적용하였다. 주어진 상태에서 행동을 고르는 방법적인 부분에서 epsilon-greedy 기법과 몬테카를로 트리탐색을 변형시킨 기법을 적용하였고, 학습을 하는 전체적인 과정을 자가 경기 학습으로 진행하였다.

3-1. Epsilon-Greedy

강화학습 모델을 여러 가지 행동을 선택할 수 있는 상황에서 어떠한 정책에 따라 하나의 행동을 결정하게 된다. 여러 정책 중 epsilon-greedy 정책은 변수 epsilon의 확률로 탐험을, (1-epsilon)의 확률로 최선의 수를 선택하는 정책이다[3]. 체스와 같은 게임은 주어진 상태에서 선택할 수 있는 행동이 많은 환경을 갖고 있다. 이러한 환경에서 학습 초반의 경우 탐험이 중요하고 학습 후반으로 갈수록 최선의 수에 가중을 두는 것이 적절하다. 따라서 본 프로젝트에서는 epsilon값이 시간이 지남에 따라 감소하도록 하였고 0에 수렴하여 탐색을 하지 않는 상황을 방지하기 위해 epsilon값이 0.2보다 작아지지 않게 설정하였다.

3-2. 변형 MCTS (Monte Carlo Tree Search)

몬테카를로 트리 탐색은 어떻게 움직이는 것이 가장 좋은 행동인지를 시뮬레이션을 통해 분석하는 알고리즘이다[4]. 검색공간에서 랜덤에 기초한 탐색을 통해 트리를 확장시키는 데 중점을 둔 알고리즘이다. MCTS는 시뮬레이션을 통해 각 행동의 reward값을 정해주지만 체스 게임에서 모든 수에 대해 시뮬레이션을 진행할 경우 너무 많은 시간이 걸려 본 프로젝트에서는 각 행동의 reward값에 특정 초깃값을 설정해주는 방식으로 MCTS를 변형하여 적용시켰다.

3-3. 자가 경기 학습

본 프로젝트의 경우 학습 모델에게 규칙에 어긋나는 움직임을 제한한 상태에서 자기 자신을 상대로 체스경기를 진행하여 학습을 진행하는 자가 경기 학습 알고리즘을 사용하였다. 이를 통해 한판에 하나의 모델로 흑과 백의 수 모두에 대하여 학습을 할 수 있게 하였다.

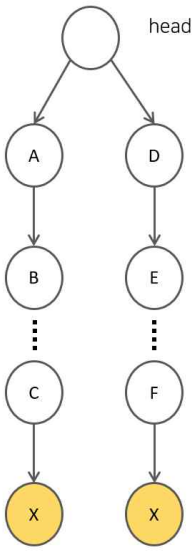
4. 모델의 학습 과정

```
While !is_game_over():
    1. pick action by epsilon-greedy
    2. record.append(current_node)
    3. if check_duplication() is True:
        current_node.append(found_node)
        current_node -> current_node.next[-1]
      else:
        current_node.next.append(new_node)
        current_node -> current_node.next[-1]
While record :
    current_node = record.pop()
    update current_node.reward
```

<그림 2. 학습 과정 pseudo code>

위 코드는 모델이 한 경기 동안 학습과 피드백 하는 과정을 나타낸 것이다. 먼저 경기가 시작되면 경기가 끝날 때 까지 주어진 상태에서 epsilon-greedy 기법에 따라 행동을 선택한 후 그 행동을 모델의 그래프에 노드의 형태로 추가해준다. 경기가 끝나게 되면 해당 판의 행동들을 거슬러 올라가면서 보상 값을 피드백 해준다. 이 과정을 반복하여 모델의 트리를 넓혀가고 보상 값을 갱신하여 점점 더 나은 모델로 학습하게 된다.

5. 모델의 구조

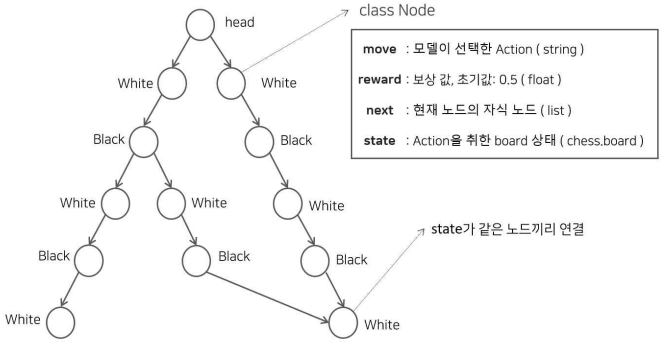


<그림 3. 초기 모델의 구조>

모델이 학습되면 위 그림처럼 그래프가 형성된다. 각 노드에는 모델이 선택한 action, reward 값, 현재 노드의 자식노드, 현재 노드의 board status에 대한 정보가 저장된다. 처음에 단순하게 액션을 취할 때 마다 노드를 계속 추가해주는 방식을 취했다. 하지만 같은 상태에 대해서 그 이전의 행동들이 다른 상황에 대해서 모델은 이 두 개의 노드를 서로 다른 노드로 저장하였다. 이에 따라 같은 상태의 노드가 모델의 그래프에 여러 개 존재하

는 취약점이 발생하였고 이 문제로 인해 모델이 몇 번 수를 두지 못한 상태에서 경험해보지 못한 상태에 빠지는 상황이 발생하여 성능 면에서 굉장히 비효율 적인 모습을 보였다.

6-1. 성능 개선



<그림 4. 개선된 모델의 구조>

첫 모델에서 발생한 중복되는 상태에 대해서 같게 처리하지 못하는 문제를 해결하기 위해 각 노드마다 보드의 상태를 저장하게 하였다. 모든 노드에 상태를 저장하게 한 후, 모델이 학습을 하는 과정에서 새 노드를 추가할 때 다른 모든 노드를 순회하여 같은 상태를 가진 노드의 여부를 알 수 있도록 하였다. 같은 상태를 가진 노드를 발견하였을 경우 자식노드를 생성하지 않고 같은 상태를 가진 노드에 연결을 해주고, 같은 상태의 노드를 찾지 못하였을 때 새 노드를 추가하게 하였다. 이러한 방법을 통해 같은 상태의 노드가 여러 개 생기지 않아 성능 면에서 이득을 보았다. 하지만 한번 수를 선택할 때 마다 모델의 모든 노드를 순회하기 때문에 속도적인 면에서 큰 저하가 발생하였다.

6-2. 속도 개선

위에서 소개한 성능을 향상시키려는 과정에서 학습을 하는데 걸리는 시간이 제곱배로 늘어났다. 강화학습에 있어서 학습의 질도 중요하지만 속도가 빠르지 못하다면 결코 좋은 학습을 할 수 없다. 이러한 속도저하를 해결하기 위해 상태를 저장할 때 모든 상태를 2차원 배열에 저장하는 방법을 사용하였다. 상태를 저장할 때 보드에 남아있는 말의 개수와 행을 맞추어 저장을 하여 체스말의 총 개수와 같은 32개의 행을 가진 2차원 배열을 만들어 저장을 하였다. 32행이 생김으로써 실제 학습하는 과정에서 굉장히 큰 성능향상을 보여주었다.

7. 실험 결과

성능 검증은 강화학습으로 학습된 모델과 랜덤으로 수를 두는 모델을 서로 겨루어, 강화학습 모델이 랜덤 모델을 상대로 얼마만큼의 승률을 보이는지 확인하여 검증하였다.

초기 모델의 경우 총 25만판을 학습하였고, 약 21시간이 소요되었다. 모델이 학습한 데이터를 기반으로 몇 번째 수까지 선택할 수 있는지 확인해 본 결과 평균 2.3번째 수까지 학습을 기반으로 선택할 수 있었다.

두 번째로 성능을 개선한 모델을 학습하였다. 개선한 모델은 총 150판을 학습하였고, 약 2일 22시간이 소요되었다. 초기 모델에 비해 학습 시간이 매우 오래 걸렸고, 평균 1.3번째 수까지 학습을 기반으로 선택할 수 있었다.

세 번째로 속도까지 개선한 모델을 학습하였다. 두 번째로 소개하였던 학습 결과와 비교하기 위해 먼저 150판을 학습시켰다. 두 번째 모델의 경우 거의 3일의 시간이 소요된 반면, 세 번째 모델의 경우 2시간 2분 만에 150판의 학습을 마칠 수 있었다. 이어서 총 1000판을 학습한 결과, 약 5일 19시간이 소요되었고, 평균 2.1번째 수까지 학습을 기반으로 선택할 수 있었다.

초기 모델이 많은 판을 학습할 수 있는 대신 효율적으로 사용하지 못해서 학습한 경기 수에 비해 성능이 좋지 않았던 반면, 최종적으로 개선한 모델은 시간은 조금 더 오래 걸렸지만, 0.004%의 경기만으로도 유사한 수준의 결과를 낼 수 있었다.

8. 개선사항

가장 먼저 개선할 수 있는 부분은 학습하지 못한 수에 대응하는 방법이다. 현재 모델의 경우 한 번도 경험해보지 못한 상태에 빠졌을 때 행동할 수 있는 범위 안에서 랜덤하게 수를 고르는 방법을 택하고 있다. 하지만 그 상태에서 한번 랜덤하게 수를 두게 된다면 그 뒤에 따라오는 모든 수들에 대해서도 계속 랜덤하게 수를 둘 가능성이 높아진다. 그렇게 되면 어느 순간부터 학습된 데이터 기반이 아닌 랜덤하게 수를 두는 모델이 되어 성능 면에서 크게 손해를 볼 수밖에 없다. 이러한 문제를 해결하기 위해서는 처음 보는 상태를 만났을 때 그 상태에서 1~2번의 수를 두었을 때 모델이 가지고 있는 노드의 상태와 같은 상태가 되는 수가 있다면 그 수를 택하는 방법, 현재 상태와 비슷한 형태를 하고 있는 상태를 갖고 있는 노드에게서 어떻게 행동을 하면 좋을지 그 수를 모방하는 방법도 있을 수 있겠다. 이런 식으로 학습되지 않은 상태에서 랜덤이 아닌 다른 방법으로 수를 두게 한

다면 성능적인 면에서 더 뛰어난 모델로 학습될 수 있을 것이다.

모델의 성능이 아닌 학습 속도적인 측면에서 개선할 점이 있다면 GPU를 사용하는 방법이다. 현재 모델을 학습시킬 때 GPU가 아닌 CPU를 사용하고 있다. GPU의 경우 행렬에 대한 연산을 수행할 때 CPU보다 훨씬 뛰어난 처리속도를 갖고 있다. 이에 맞춰 모델의 학습 과정에서 상태를 비교하고, 저장하는 과정을 numpy 라이브러리의 행렬연산을 활용하여 처리 할 수 있게끔 해주면 GPU를 사용하여 학습속도를 끌어 올릴 수 있을 것이다.

9. 참고 문헌

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, Dec 5, 2017.

[2] “Research Blog: AlphaGo: Mastering the ancient game of Go with Machine Learning”. 《Google Research Blog》. Jan 27, 2016.

[3] Sudharsan Ravichandiran, 파이썬 예제와 함께하는 강화학습 입문 - OpenAI GYM과 TensorFlow 실습 가이드, June 20, 2019.

[4] Tom Vodopivec, Spyridon Samothrakis, Branko Ster, On Monte Carlo Tree Search and Reinforcement Learning, Dec 20, 2017.