Journal of the Korean Institute of Industrial Engineers
Vol. 50, No. 5, pp. 304-312, October 2024.
ISSN 1225-0988 | EISSN 2234-6457
https://doi.org/10.7232/JKIIE.2024.50.5.304

강화학습을 통한 윷놀이 최적 전략의 학습 및 윷놀이에 대한 고찰

송승규[1] · 천다호[2] · 민승기[2†]

[1]서울대학교 데이터사이언스 대학원 / [2]한국과학기술원 산업및시스템공학과

Learning to Play Yut Nori

Seunggu Song[1] · Daho Cheon[2] · Seungki Min[2]

[1]Graduate School of Data Science, Seoul National University
[2]Department of Industrial & Systems Engineering, KAIST

In our research, we adopt a reinforcement learning approach to analyze Yut nori, a traditional Korean board game. We present a succinct Markov game formulation of Yut nori and devise an effective training procedure based on temporal difference learning. The strategy developed through our model achieves a 93% win rate against random strategies and a 55% win rate against human players. Self-play simulations reveal several stylized facts about Yut nori. For instance, the first player has a slight advantage over the second player, and the predicted win rate increases by approximately 10% point each time a game piece (Mal) reaches the finish position.

Keywords: Yut Nori, Board Game, Reinforcement Learning, AI Player

1. 서 론

윷놀이는 오랜 기간동안 많은 대중들이 즐겨왔던 놀이임에도 불구하고, 윷놀이에 대한 과학적이고 체계적인 연구는 찾아보기 힘들다. 이에 본 연구는 강화학습 방법론을 도입하여 윷놀이 (유사) 최적 전략을 도출하고 윷놀이에 대하여 고찰해보고자 한다.

강화학습(reinforcement learning)은 시행착오를 겪으며 최적의 의사결정 전략을 스스로 학습해나가는 방법론으로, 깊은 신경망 기술과 결합되었을 때 매우 복잡한 문제에서 전문가를 뛰어넘는 성능을 달성할 수 있음이 최근 수년간의 연구들에서 거듭 확인되고 있다. 바둑에서 딥마인드사의 AlphaGo(Silver *et al.*, 2016)가 프로기사 이세돌을 상대로 승리를 거둔 것을 시작으로, 강화학습은 보드게임들뿐만 아니라 금융 투자(Jiang and Liang, 2017), 자율주행(Zhu *et al.*, 2020)과 같은 실제 산업 현장의 문제들로 그 적용 범위를 넓혀가고 있다.

본 연구는 윷놀이에 적합한 강화학습 방법론을 제안·구현한다. 구체적으로, 본문 3.1절에서 윷놀이 규칙을 반영하여 윷놀이를 마르코프 게임으로 표현하는 방식을 제안하고, 3.2절에서 신경망을 통해 학습하기에 적합한 형태로 게임상태를 인코딩하는 방식 및 가치함수를 근사하는 신경망을 효율적으로 업데이트하는 방식을 제안하며, 4.1절에서 사람 혹은 다른 휴리스틱 전략과의 대결을 통해 학습 과정의 효율성 및 최종 학습된 전략의 성능을 검증한다.

윷놀이는 약 10^{15}개의 게임상태를[1] 가지는데, 이는 약 10^{171}

이 논문은 2022년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업의 지원을 받아 수행된 연구임(과제번호: 2022R1C1C1013402).
† 연락저자 : 민승기 교수, 34141 대전광역시 유성구 대학로 291 산업경영학동 3104호, 한국과학기술원 산업및시스템공학과, Tel : 042-350-3112, E-mail : skmin@kaist.ac.kr
송승규와 천다호, 두 저자는 동등한 기여를 하였음.
2024년 4월 11일 접수; 2024년 9월 24일 게재 확정.

1) 윷가락을 던져 윷 혹은 모를 20번 연속으로 얻으면 게임에 무조건 승리한다. 21번 이상 윷가락을 던지는 경우를 제외하고 경우의 수를 계산한다면 사위의 결과로 $4\times{}_3H_{20}\approx 10^3$개의 경우의 수가 존재한다. 윷판에는 31개의 칸에 8개의 말이 존재하게 되므로 윷판의 경우의 수는 $31^8=10^{12}$이다. 즉, 총 가능한 게임 상태의 수는 10^{15}이다.

개(Tromp, 2016)의 바둑이나 약 10^{43}개(Shannon, 1950)의 체스에 비해 작은 값이다. 하지만 동시에 확률적 요소가 많이 개입되는 보드게임이다. 따라서 비교적 얕은 신경망으로도 가치함수 근사가 가능하고 의도적인 탐색을 하지 않더라도 가치함수 학습이 이루어지는 등의 용이함이 있지만, 그만큼 승패가 운에 의해 결정되고 학습데이터에 노이즈가 많이 존재하는 등의 어려움이 있다. 이에 본 논문에서는 바둑이나 체스 등의 보드게임에서 그 효과성이 검증된 target network 및 temporal difference learning과 같은 강화학습 기법들을 활용하되, 액션 가치함수 대신 상태 가치 함수를 학습한다거나 Monte-carlo tree search 대신 one-step look-ahead을 사용하는 등 윷놀이에 적합하도록 기법을 변형・적용하였다. 또한 성능 검증 시에 확률적 요소 개입의 영향을 최소화하기 위한 시뮬레이션 방법을 고안하였다.

본 연구는 우수한 성능을 보이는 전략을 도출하는 것에서 나아가 윷놀이라는 게임을 더욱 깊게 이해하는 데에도 그 목적을 둔다. 윷놀이에서는 확률적 요소가 많이 개입되기에 전략에 따른 압도적인 성능 차이가 존재하지 않는 것으로 보이며, 따라서 학습된 전략의 자가 대결 양상이 일상적인 윷놀이 경기 양상과 크게 다르지 않을 것으로 예상된다.

선공의 승률이 50.1%이라는 것과 같이, 본 논문 4.3절에서 제시되는 통계적 사실들이 대중들로 하여금 윷놀이를 더욱 재미있게 즐기는 데에 도움이 되기를 기대한다.

2. 윷놀이 규칙

윷놀이는 두 명의 플레이어가 턴(turn)을 주고 받으며 4개의 윷가락을 던지고 그 결과에 따라 <Figure 1>과 같은 윷판에서 각자 4개의 말을 움직이는 게임이다. 시작 지점(0번 위치)에 위치한 4개의 말을 모두 도착 지점(30번 위치)로 먼저 도착시키는 플레이어가 승리한다.

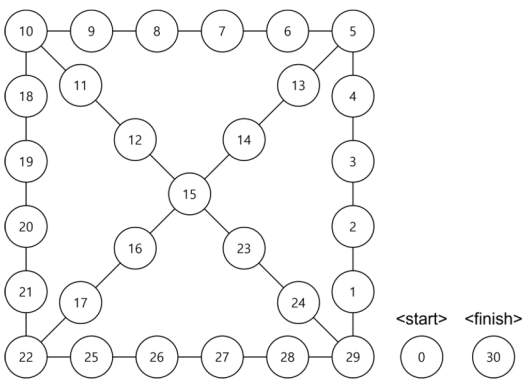

Figure 1. Board Configuration

각 윷가락은 배(평평한 면)와 등(둥근 면)이 있으며, 4개의 윷가락을 던져서 배가 1, 2, 3, 4, 0개가 나온 것을 각각 '도', '개', '걸', '윷', '모'라고 한다. 단, 4개의 윷가락 중 하나는 특별히 구분되어, 이 윷가락만 배가 나온 경우는 도가 아닌 '뒷도'라고 한다. 본 논문에서는 배가 나올 확률이 60%로, 각 사위[2]가 나올 확률은 <Table 1>과 같다고 가정한다. 윷이 멍석 밖으로 나가는 경우인 낙을 고려하지 않는다.

Table 1. The Assumed Probability Distribution of Yut Score

Back-do	Do	Gae	Geol	Yut	Mo
3.8%	11.5%	34.6%	34.6%	12.0%	2.6%

윷가락을 던진 플레이어는 도착 지점에 있지 않은 본인의 말 중 하나를 골라 해당 사위가 지정하는 거리만큼 이동시킨다. 도, 개, 걸, 윷, 모는 각각 말을 1, 2, 3, 4, 5칸 전진시키며, 뒷도는 말을 1칸 후진시킨다. 말을 이동시킨 위치에 본인 말이 존재한다면 이를 '업었다'고 하며, 이후 해당 위치에 있는 모든 말들을 함께 움직인다. 반대로 만약 말을 이동시킨 위치에 상대방 말이 존재한다면 이를 상대방 말을 '잡았다'고 하며, 상대방 말은 시작 지점으로 돌아간다. 이때, 윷가락을 던져 윷 혹은 모가 나오거나 윷이나 모가 아닌 사위로 상대방 말을 잡는 경우 해당 플레이어는 한 번 더 윷가락을 던진다.[3] 따라서 말을 이동시킬 때 사용할 수 있는 사위가 여러 개 존재할 수 있으며, 플레이어의 재량에 따라 그 사용 순서를 정할 수 있다. 사용 가능한 사위가 없고 새로 윷가락을 던져야 하는 상황 또한 아니라면, 현재 턴은 종료되고 상대 플레이어가 윷가락을 던지며 새로운 턴이 시작된다.

추가로, 5, 10, 15번에 위치한 말을 움직일 때는 반드시 지름길을 통해(13, 11, 23번 위치) 움직여야 한다. 15, 22, 29번 위치의 말을 뒷도를 사용하여 움직이는 경우에는 지름길로 되돌아갈 것인지 먼 길로 되돌아갈 것인지 선택할 수 있다(각각 12 또는 14번, 17 또는 21번, 24 또는 28번 위치).[4] 또한 뒷도로 1번 위치의 말을 29번 위치로 이동시킬 수 있으며, 진출하여 있는 말이 없을 때 뒷도가 나온 경우 곧바로 턴이 종료된다.

3. 강화학습 기반 최적 전략 학습

3.1 마르코프 게임으로서의 윷놀이

(1) 게임 상태(game state)

게임에 참여하고 있는 두 플레이어를 각각 A와 B라고 칭한다. 시점 t에서의 윷놀이 게임 상태 s_t는 현재 턴의 소유자(현 시점에 말을 움직일 권한을 가진 플레이어)를 표현하는 상태

2) 사위란 윷을 던져서 나온 결과를 의미하며, 뒷도, 도, 개, 걸, 윷, 모 중 하나를 가질 수 있다.
3) 말의 잡기와 관련된 규칙의 경우 지역마다 차이를 보이기도 한다.
4) 뒷도와 관련된 규칙의 경우 지역마다 차이를 보이기도 한다.

변수 $w_t \in \{A,B\}$, 플레이어 A의 말들의 위치를 표현하는 상태 변수 l_t^A, 플레이어 B의 말들의 위치를 표현하는 상태변수 l_t^B, 현재 사용 가능한 사위들을 표현하는 상태 변수 y_t, 그리고 현재 턴의 소유자가 윷가락을 던져야 하는 상황인지 표현하는 상태 변수 $c_t \in \{0,1\}$로 기술될 수 있다.

- 상태 변수 $l_t^A \in \{0,\cdots,30\}^4$는 플레이어 A가 소유한 4개 말들의 위치를 <Figure 1>에 표기된 정수값들로 표현한다. 상태 변수 $l_t^B \in \{0,\cdots,30\}^4$도 마찬가지로 정의된다.
- 상태 변수 $y_t \in \mathbb{Z}^6$는 현재 플레이어 w_t가 사용 가능한 사위들(아직 사용하지 않은 윷 던짐 결과들)의 개수를 뒷도, 도, 개, 걸, 윷, 모의 순서로 표현한다. 예를 들어, 새로운 턴에서 윷가락을 던져서 (윷, 윷, 개)가 나온 상황은 $y_t = (0,0,1,0,2,0)$로 표현된다.
- 상태 변수 $c_t \in \{0,1\}$은 현재 플레이어(w_t)가 윷가락을 던져야 하는 상황인지 나타낸다. 예를 들어, 직전 시점에 상대 플레이어의 턴이 종료되어 현재 플레이어가 새로 윷가락을 던져야 하는 상황 혹은 상대 플레이어의 말을 잡아 윷가락을 추가적으로 던져야 하는 상황이라면 $c_t = 1$로 표현된다.

게임 초기 상태는 모든 말들이 0번 위치에 있기에 $l_1^A = l_1^B = (0,0,0,0)$로, 사용 가능한 사위는 아직 없지만 새로 윷을 던져야 하기에 $y_1 = (0,0,0,0,0,0)$ 및 $c_1 = 1$로 표현된다.

(2) 액션(action)과 게임 상태 변이

게임 상태 $s_t = (w_t, l_t^A, l_t^B, y_t, c_t)$가 주어졌을 때, 현재 플레이어 w_t가 취할 수 있는 액션과 그에 따른 게임 상태 변화에 대하여 기술한다. 시점 t에서 취할 수 있는 액션들의 집합을 $\mathcal{A}_t$ (혹은 $\mathcal{A}(s_t)$)로 표현한다.

우선 현재 플레이어가 윷가락을 던져야 하는 상황인 경우 ($c_t = 1$), 플레이어가 의사결정을 내릴 사항은 없으며($\mathcal{A}_t = \phi$), 윷가락을 던져 얻게 되는 결과에 따라 게임 상태가 업데이트된다. 예컨대, $y_t = (0,0,0,0,0,1)$, $c_t = 1$인 상황에서 윷을 던져 (윷, 윷, 개)가 나왔다면 $y_{t+1} = (0,0,1,0,2,1)$ 및 $c_{t+1} = 0$로 업데이트되며 나머지 상태 변수들의 값은 현재 값을 유지한다.

윷가락을 던져야 하는 상황이 아닌 경우($c_t = 0$), 현재 플레이어는 어떤 사위를 사용하여 어떤 말을 움직일지 결정해야 한다. 즉, 액션 a_t은 사용하고자 하는 사위 $a_t^y \in \{$뒷도,$\cdots$,모$\}$ 및 움직이고자 하는 말 $a_t^l \in \{1,2,3,4\}$로 표현되며, 엄밀하게는 뒷도를 사용할 때 다음 위치에 대한 선택권이 생기는 예외적인 상황이 있기에 지름길로 돌아갈지 선택하는 액션 변수 $a_t^b \in \{0,1\}$가 추가된다. 이러한 액션의 결과로 (1) 두 플레이어의 말의 위치 l_t^A, l_t^B가 2절에서 기술된 윷놀이 규칙에 따라 변경되고, (2) 사용 가능한 사위들 중 하나를 소비하게 되며($y_{t+1} = y_t - e_{a_t^y}$), (3) 윷이나 모가 아닌 사위로 상대 말을 잡은 경우 새로 윷가락을 던질 기회를 얻게 된다($c_{t+1} = 1$). 현재 플레이어에게 허용된 액션 집합 $\mathcal{A}_t$의 크기는 사용 가능한 사위의 종류 수(y_t에서 0이 아닌 요소의 개수)와 현재 움직일 수 있는 말 개수의 곱으로 결정된다. 이때 시점 t에서 사용 가능한 사위가 하나밖에 없고($\sum_{i=1}^{6} y_{t,i} = 1$) 그 사위를 사용했을 때 새로 윷가락을 던질 기회를 얻지 못한다면 이는 턴의 종료를 의미한다. 따라서 턴의 소유자가 변경($w_{t+1} \neq w_t$)되며 상대 플레이어가 새로 윷가락을 던질 기회를 얻는다($c_{t+1} = 1$).

본 논문에서는 상태 변수 c_t를 도입하여 윷가락을 던져야 하는 상황($c_t = 1$)과 의사결정을 내려야 하는 상황($c_t = 0$)을 분리하여 고려하고 있으며, 따라서 후자의 상황($c_t = 0$)에서 의사결정 직후의 게임 상태 s_{t+1}는 확률적 요소의 개입 없이 오직 현재 상태 s_t와 현재 액션 a_t으로만 결정된다. 이러한 설정은 후술한 전략 구현 및 학습에 있어서 많은 편의성을 가져다준다.

(3) 보상(reward)과 가치 함수(value function)

게임 종료 시점에, 승리한 플레이어에게 +1의 보상이, 패배한 플레이어에게 -1의 보상이 주어지며, 게임 진행 과정 도중에는 보상이 발생하지 않는다. 이때 플레이어 A 입장에서의 가치함수 $V^A(s_t)$는 s_t로 특정되는 게임 상황에서 플레이어 A의 총 미래 보상의 기댓값로 정의되며, 이는 A가 이길 확률을 -1과 1 사이의 값으로 나타내는 것과 같다. 제로섬 게임이기에 플레이어 A의 가치함수는 플레이어 B의 가치함수를 유일하게 결정짓는다($V^B = -V^A$).

3.2 가치 기반 강화학습

본 논문에서는 가치 기반 강화학습 알고리즘을 적용하여 최적 가치 함수 및 윷놀이 최적 전략을 학습한다. 인공신경망으로 value network를 구성하여 최적 가치 함수를 근사 추정하며, 반복적인 게임 시뮬레이션을 통해 value network의 추정 오차 및 행동 전략의 성능을 지속적으로 개선시킨다.

(1) 게임 상태 인코딩

보다 구체적으로, 현재 상태 $s_t = (w_t, l_t^A, l_t^B, y_t, c_t)$가 주어졌을 때 플레이어 A 입장에서의 인코딩 $\hat{s}_t^A$은 다음과 같이 결정된다. 첫 124(4×31)개의 값 $\hat{s}_{t,1:124}^A$은 본인 말들의 위치 l_t^A를, 그 다음 124개의 값 $\hat{s}_{t,125:248}^A$은 상대 말들의 위치 l_t^A를 one-hot vector로 인코딩한다. 그 다음 6개의 값 $\hat{s}_{t,249:254}^A$은 본인이 현재 사용 가능한 사위들의 개수를 표현하는데, 현재 턴의 소유자가 본인이라면 y_t값을, 아니라면 0의 값을 갖도록 한다. 마지막 3개의 값 $\hat{s}_{t,255:257}^A$은 다음 3가지 상황을 구분하는 one-hot vector로 주어진다 - 상대가 윷을 던져야 하는 상황($c_t = 1, w_t = B$), 본인이 말을 놓아야 하는 상황($c_t = 0$, $w_t = A$), 본인이 윷을 던져야 하는 상황 ($c_t = 1, w_t = A$). 플레이어 B 입장에서의 인코딩 $\hat{s}_t^B$도 마찬가지

방식으로 규정된다.

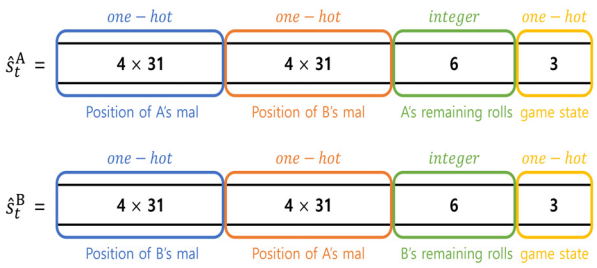

Figure 2. Game State Encoding for Each Player

Value network $\hat{V}_\theta(\cdot)$는 인코딩된 게임 상태를 입력 받아 -1과 1 사이의 값을 출력하며, 가치 함수를 표현하는 것을 목표로 한다($\hat{V}_\theta(\hat{s}_t^A) \approx V^A(s_t), \hat{V}_\theta(\hat{s}_t^B) \approx V^B(s_t)$). <Figure 3>과 같이 257개의 유닛을 가진 input layer, 각각 256개의 유닛을 가진 2개의 hidden layer로 구성하였고, hidden layer 유닛의 activation function으로는 ReLU를, output function으로는 tanh를 채택하였다. Value network를 구성하는 총 132,097개의 파라미터를 θ로 표기한다.

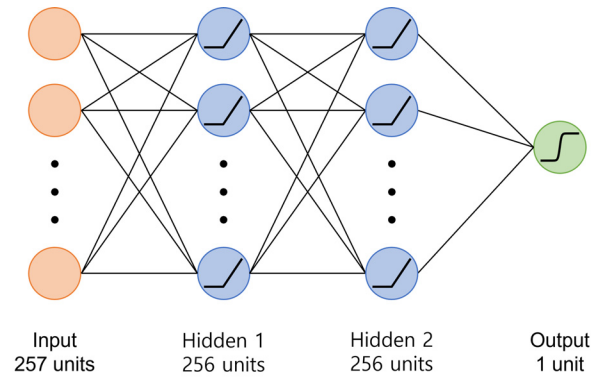

Figure 3. Value Network Architecture

(2) 행동전략(behavior policy)

Value network의 학습 데이터 생성을 위해 사용하는 행동 전략으로써 현재까지 학습된 value network에서 파생되는 greedy policy를 이용한다. 보다 구체적으로, $\hat{V}_\theta$에서 파생되는 행동 전략 π_θ은 매 의사결정 시점에 다음과 같은 방식으로 액션을 선택한다: $a_t = \arg\max_{a \in A(s_t)} \hat{V}_\theta\left(\hat{s}_{t+1}^{w_t}(s_t,a)\right)$.

여기서 $\mathcal{A}(s_t)$는 현재 게임 상태 s_t에서 취할 수 있는 액션들의 집합을 의미하며, $\hat{s}_{t+1}^{w_t}(s_t,a)$는 액션 a를 취한 직후의 게임 상태 인코딩을 의미한다. 이때 의사결정을 내려야 하는 상황($c_t = 0$)은 윷가락을 새로 던져야 하는 상황($c_t = 1$)과 분리되어 있기 때문에 직후의 게임 상태 $s_{t+1}(s_t,a)$는 확률적 요소의 개입 없이 현재 상태와 액션만으로 결정된다.

(3) Value network 학습

파라미터 학습 방식으로 target network(Hasselt *et al.*, 2016)를 동반한 temporal difference method(Tesauro, 1995)를 적용하였다. $i-1$번째 epoch을 마친 뒤의 모델 파라미터를 θ_{i-1}라고 할 때, i번째 epoch에서는 다음 과정을 통해 파라미터를 업데이트한다.

먼저 현재까지 학습된 value network $\hat{V}_{\theta_{i-1}}$에서 파생되는 행동 전략 $\pi_{\theta_{i-1}}$을 자가대결(self-play)시켜 게임 데이터를 생성·수집한다. 한 번의 경기를 통해 게임 상태 궤적 $(s_1,\ldots,s_T)$와 승패 여부에 따른 보상값 r^A 및 r^B를 수집하게 되고, 이를 두 플레이어 각각의 입장에서 인코딩하여 $\xi^A = (\hat{s}_{1,\ldots,}^A \hat{s}_T^A r^A)$와 $\xi^B = (\hat{s}_{1,\ldots,}^B \hat{s}_T^B r^B)$를 얻는다. 매 epoch마다 총 1,024판의 자가대결을 진행하며, 따라서 2,048개의 학습데이터를 얻는다.

다음으로 32번의 iteration을 거쳐 파라미터 업데이트를 진행한다. 매 itertaion마다 2,048개의 학습데이터 중 64개를 무작위로 선택하여 minibatch를 구성하고, minibatch에서 발생하는 loss의 총합을 줄이는 방향으로 value network 파라미터 θ를 업데이트한다. 이 과정에서 하나의 학습데이터 ξ^A(혹은 ξ^B)가 만들어내는 loss는 다음과 같이 주어진다:

$$L(\theta;\theta^{target},\xi^A) := \sum_{t:(w_t=A)\wedge(c_t=0)} (v - \hat{V}_\theta(\hat{s}_{t+1}^A))^2.$$

$$v := \begin{cases} \hat{V}_{\theta^{target}}(\hat{s}_{\tau(t)+1}^A), & \text{if } t < \max\{\alpha : (w_t = A) \wedge (c_t = 0)\}, \\ r^A, & \text{if } t = \max\{\alpha : (w_t = A) \wedge (c_t = 0)\}. \end{cases}$$

$$\tau(t) = \min\{\tau : (\tau > t) \wedge (w_\tau = A) \wedge (c_\tau = 0) \textit{RIGHT}.$$

플레이어는 $c_t = 0$인 시점 t에서만 액션을 필요로 하고, 이때 액션을 취하기 위해서 액션을 취한 직후의 게임 상태 인코딩인 $\hat{s}_{t+1}^A$의 가치만을 이용한다. 따라서 $c_t = 0$인 t에 대해 $\hat{V}_\theta(\hat{s}_{t+1}^A)$를 정확하게 평가하는 것이 중요하고, 이를 위해 액션을 취한 직후의 게임 상태만을 사용해 위와 같이 loss를 구성하였다.

여기서 θ^{target}은 target network의 파라미터이다. 모든 iteration을 마치고 i번째 epoch가 끝나는 시점에서, 그때의 value network 파라미터를 θ_i라고 할 때, θ^{target}은 다음과 같이 업데이트된다.

$$\theta^{target} \leftarrow 0.1 \times \theta_i + 0.9 \times \theta^{target}.$$

Value network의 minibatch 업데이트 과정에서 사용되는 optimizer로 Adam을 채택했으며, learning rate는 10^{-3}으로 설정하였다.

(4) Look-ahead 전략

Value network 학습이 완료된 이후 성능을 분석하는 단계에서 성능 향상을 꾀하기 위해, 매 의사 시점에 그 다음 한 수를 더 고려해보는 one-step look-ahead 전략을 고안하였다.[5] 이 전략은 행동 전략 (1)과 유사하게 다음과 같은 액션 선택 방식을 구현한다.

$$a_t = \arg\max_{a \in A(s_t)} \hat{V}_\theta^{LA}(s_{t+1}(s_t, a)).$$

여기서 $\hat{V}_\theta^{LA}(s_{t+1})$은 액션 a를 취한 직후의 게임 상태 s_{t+1}의 가치를 평가함에 있어서 그 다음 한 단계를 더 진행한 게임 상태 s_{t+2}를 추가로 고려한다 :

$$\hat{V}_\theta^{LA}(s_{t+1}) := \begin{cases} \max_{a' \in A(s_{t+1})} \hat{V}_\theta(\hat{s}_{t+2}^{w_t}(s_{t+1}, a')), & \text{if } c_{t+1} = 0, \\ E_{y_{t+2} = y_{t+1} + Y}[\hat{V}_\theta(\hat{s}_{t+2}^{w_t})], & \text{if } c_{t+1} = 1. \end{cases}$$

다음 게임 상태 s_{t+1}가 한 번 더 의사결정을 해야 하는 상황 ($c_{t+1} = 0$)이라면 다음 시점에 취할 수 있는 최선의 액션 및 그 액션의 가치를 산출하게 되고, 윷을 새로 던져야 하는 상황 ($c_{t+1} = 1$)이라면 새로 얻게 될 사위들의 확률 분포(위 식에서 Y로 표현된 random vector의 확률 분포)를 고려하여 윷을 던진 후의 게임 상태 가치의 기댓값을 산출하게 된다.

4. 전략 성능 분석 및 고찰

4.1 전략 성능 분석

(1) 학습 진행에 따른 승률 추이

강화학습 진행 과정 중 성능 향상 정도를 파악하기 위하여, 매 10번의 epoch마다 점수 기반 휴리스틱 전략을 상대로 게임 시뮬레이션을 1,000회 반복하여 승률을 집계하였다. 해당 휴리스틱 전략은 말들의 현재 위치에서 골인 지점까지 거리를 기반으로 게임 상태에 대한 점수를 부여하고 이 점수를 최대화시키는 수를 선택하는 전략으로, 자세한 내용은 부록에서 서술한다.

<Figure 4>는 승률 추이를 보여준다. 학습 초기 30%의 승률에서 시작하여 약 200회의 epoch 이후에는 65%의 승률을 기록하며 상대 전략보다 월등히 좋은 성능을 보여주었다. 추가적으로 Monte-Carlo(MC) 학습 방식 및 target network를 사용하지 않는 temporal difference(TD without target) 학습 방식도 구현하여 비교하였다. 본 논문에서 채택한 학습 방식(TD with target)이 학습 극초반에는 상대적으로 저조한 승률을 보였지만, 이내 가파르게 상승하며 80회의 epoch 이후에는 세 가지의 학습 방식 중 가장 높은 성능을 보였다.

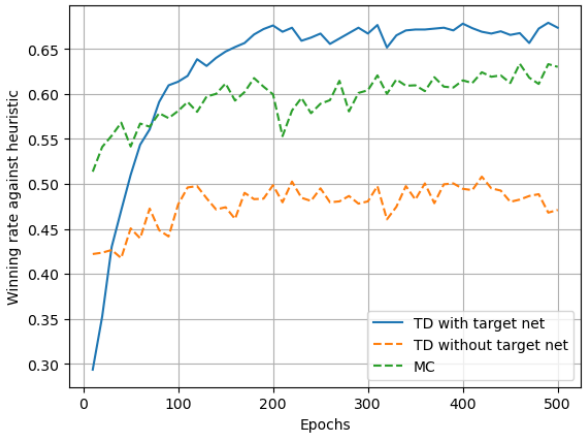

Figure 4. Win Rate Against the Score-based Heuristic Strategy Throughout the Training Procedure

(2) 성능 비교

총 500 epoch를 거치면서 생성된 500개의 전략들 중 점수 기반 휴리스틱 전략 상대로 승률이 가장 높았던 전략을 최종 전략으로 선택하였고, 해당 전략의 성능을 다른 휴리스틱 전략들 및 사람을 상대로 측정하였다. 이때 선공 여부 및 우연적 요소에 의하여 승패가 결정되는 현상의 영향을 최소화하기 위하여, 선공과 후공 플레이어가 얻게 될 윷 사위들을 개별적으로 무작위 생성하여 놓고, 선후공 순서만 바꾸어 매번 두 판의 경기를 진행하였다. 이러한 집계 방식 하에서 특정 전략을 자가대결(self-play)시켰을 경우 승률이 정확히 50%로 측정된다.

Table 2. Win Rate Against the other Strategies and Human Players

Random	Last	Score-based	No Lookahead	Human
93.4%	83.8%	68.1%	52.5%	55%
(99.8%)	(97.9%)	(88.1%)	(57.4%)	-

각 휴리스틱 전략을 대상으로 10,000판씩 그리고 10명의 사람을 대상으로 10판씩 경기를 진행하였고, 그 결과를 <Table 2>에 나타내었다. 괄호 안에 표기된 숫자는 같은 사위 결과들에서 선후공 순서만 바꾸었을 때 승패가 뒤바뀌는 경우를 제외하고 집계된 승률을 나타낸다. 'Random'는 아무 말이나 무작위로 선택하여 움직이는 전략, 'Last'는 가장 뒤쳐져있는 말을 움직이는 전략, 'Score-based'는 앞서 언급된 점수 기반 휴리스틱 전략을 의미하며, 이 전략들을 상대로 압도적인 우위를 보였다. 'No Lookahead'는 동일한 value network을 사용하되 3.2절에서 기술된 one-step look-ahead 방식이 적용되지 않은

5) Look-ahead 전략을 Value network 학습을 위한, 즉 학습 데이터 생성을 위한 행동전략으로써도 사용해 보았으나 성능상 유의미한 차이는 없었고 계산량만 증가하였기에 이를 학습 데이터 생성 과정에서는 사용하지 않았다.

전략을 의미하며, look-ahead 방식이 성능을 근소하게 개선시킴을 확인할 수 있다. 나아가, 사람을 상대로도 우위를 보임이 확인된다.

4.2 최적전략에 관한 분석

(1) 전략 동작 예시

특정한 상황들을 가정하여 학습된 전략이 합리적 판단을 내리는지 검토하였다. 예시로, 아래 <Figure 5>은 같은 사위가 주어졌더라도 말들의 위치에 따라 다른 선택을 내린다는 것을 보여준다. 두 상황 모두 뒷도가 사위로 주어진 상황을 고려하는데, 첫 번째 상황의 경우 6번에 위치한 말을 5번으로 옮기는 것보다 1번에 위치한 말을 29번으로 옮겨 말을 골인 시키는 선택이 더욱 높은 가치를 가지는 것으로 나타났다. 두 번째 상황에서는 뒷도를 이용해 5번에 위치한 상대방의 말을 잡는 것이 가능하며, 이로 인해 6번에 위치한 말을 이동시키는 것이 더 높은 가치를 가지는 것으로 나타났다. 학습된 전략이 실제 사람과 유사하고 매우 합리적인 판단을 하고 있음을 알 수 있다.

아래 <Figure 6>은 'Lookahead' 전략과 'No Lookahead' 전략이 다른 결정을 내리는 상황을 예시로 보여준다. 상대방의 말이 6칸 앞에 있고 윷과 개가 사위로 주어진 상황이며, 이때 Lookahead 전략은 두 단계 앞의 수까지 보기 때문에 윷과 개를 모두 활용하여 말 a를 움직여 상대방의 말을 잡는 선택을 보여주었다. 하지만 No Lookahead 전략의 경우 바로 직후의 상태만을 고려하기 때문에 말 a를 2칸 움직여 지름길인 5번 위치로 이동하고, 남은 사위인 윷으로 b를 이동시키는 모습을 보여주었다. 두 개의 선택 모두 어느 정도 타당성이 있는 움직임이나, Lookahead 전략이 근소하게 더 좋은 선택을 하므로 <Table 2>에서와 같은 성능 차이가 나타난 것으로 예측된다.

(2) 사위별 출발 선택 비율

개별적인 사례 분석에서 나아가, 자가 대결 시뮬레이션 결과를 통계적으로 분석하여 합리적인 윷놀이 전략에 관하여 고찰한다. 매 의사 결정 시점마다 취할 수 있었던 액션들과 전략이 실제로 취한 액션을 집계하여 분석하였다.

학습된 전략이 이미 출발하여 판에 나와 있는 말을 전진시

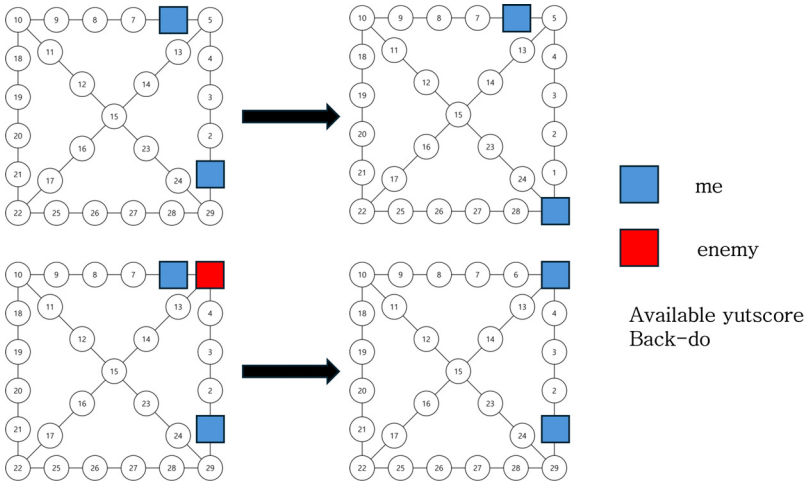

Figure 5. Simple Simulation of Value Network

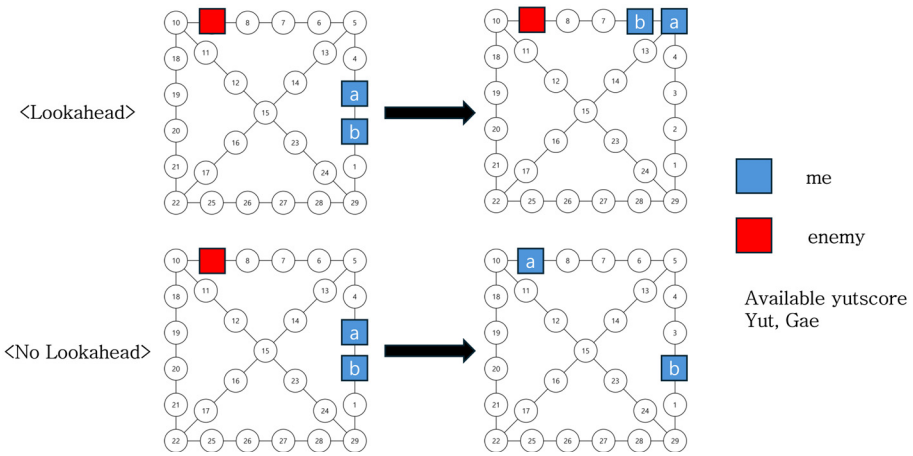

Figure 6. Difference between 'Lookahead' and 'No Lookahead'

키는 대신 새로운 말을 출발시키는 선택을 하는 비율을 각 사위별로 집계하여 <Table 3>에 나타내었다. 예를 들어, '도를 이용하여 출발 지점에 위치한 말을 1번 위치로 이동'과 '도를 이용하여 기존 말을 한 칸 전진'이라는 선택지가 주어져 있던 상황들 중 63.1%의 경우 학습된 전략은 전자를 선택하였음을 의미한다. 괄호 안의 숫자는 그러한 선택을 내린 경우들 중 상대의 말을 잡게 되었던 경우의 비율을 나타낸다.

Table 3. Likelihood of the Trained Strategy Deciding 'Advancing a New Mal' Against 'Advancing a Mal already on Board' Given a Yut Score Outcome when Both Actions are Available, where the Numbers in the Parenthesis Represent the Probability of Catching an Opponent's Mal in Such Moves

Do	Gae	Geol	Yut	Mo
63.1%	46.3%	43.9%	41.9%	45.0%
(31.1%)	(38.4%)	(35.1%)	(19.2%)	(22.8%)

다른 윷사위들에 비하여 도의 출발 선택 비율이 유난히 높은데, 이는 기존 말을 한 칸 전진하는 것에 비하여 새로운 말을 출발시키는 것이 위험 부담이 적으면서도 상대방의 새로운 말 출발도 견제할 수 있는 이점을 가지고 있기 때문이라고 해석된다. 또한 개나 걸로 출발을 선택하는 경우의 약 40% 정도는 상대방 말을 잡는 경우라고 파악되는데, 다시 말해 상대방 말을 잡는 경우가 아니라면 굳이 개나 걸로 새로운 말을 진출시키지 않는 것이 좋다고 할 수 있다. 앞서 언급된 첫 윷사위 결과에 따른 승률에 대한 해석과 일치하는 결과라고 할 수 있다.

(3) 잡기/업기/출발 선택 비율

위와 마찬가지로 학습된 전략이 상대 말을 잡는 선택, 본인 말을 업는 선택 및 새로운 말을 출발시키는 선택을 하는 비율을 산출하였다. 구체적인 상황에 따라 다르겠지만, '잡기'는 거의 대부분의 경우 최선의 결정이며, '업기'는 평균적으로 최선의 결정, '출발'은 평균적으로 최선이 아닌 결정임을 알 수 있다. 추가로 '업기'와 '잡기' 선택지가 모두 주어진 상황에서는, 업는 선택보다 잡는 선택을 약 7배 더 많이 하는 것으로 관찰되었다.

Table 4. Likelihood of The Trained Strategy Deciding to Move a Mal to Carry Another Mal, to Catch an Opponent's Mal, or to Start a New Run, when other Alternative Actions are Available

Catch	Carry	Start
96.9%	62.8%	42.1%

4.3 윷놀이에 관한 분석

(1) 게임 요약 통계

게임 당 턴(turn) 수, 윷을 던진 횟수, 말을 이동시킨 횟수(의사 결정 횟수), 본인 말을 업은 횟수 및 상대 말을 잡은 횟수에 대한 요약 통계를 <Table 5>에 기재하였다. 예를 들어, 윷을 한 번 던질 때마다 10초, 의사 결정을 한 번 내릴 때마다 10초가 소요된다면, 윷놀이 한 판을 진행하는 데 평균 14분이 소요된다고 할 수 있다.

Table 5. Game Statistics in the Self-play of the Trained Strategy

	Average	Minimum	Median	Maximum
No. of turns	29.21	2	28	90
No. of Yut castings	41.24	11	40	130
No. of Mal moves	41.05	11	39	130
No. of carries	3.21	0	3	11
No. of catches	7.02	0	6	43

(2) 선공 승률

선공 플레이어의 가장 첫 번째 윷사위 결과 별로 승률을 측정하여 <Table 6>에 정리하였다. 뒷도의 경우 말의 이동 없이 즉시 후공 플레이어의 턴으로 넘어가기에 제외하였다.

Table 6. Win Rate of the First Player Given the First Yut-score Outcome

Do	Gae	Geol	Yut	Mo	Total sum
53.1%	48.9%	48.6%	54.2%	55.5%	50.11%

우선 선공의 승률이 50.11%로, 아주 근소한 우위를 가진다는 것을 확인할 수 있다. 첫 윷사위로 도가 나온 경우 53.1%로 상당히 높은 승률을 보여주며, 오히려 개나 걸이 나온 경우 승률이 50%를 넘지 못하는 것으로 확인되는데, 이는 개나 걸의 경우 다음 턴에 후공 플레이어가 매우 높은 확률(34.6%)로 선공 플레이어의 말을 잡게 되기 때문이라고 해석 가능하다.

(3) 골인에 따른 기대 승률 변화

각 플레이어의 골인 지점에 도착한 말의 개수에 따른 선공 플레이어의 승률을 측정하여 <Table 7>에 나타내었다. 예를 들어, 어느 한 게임 진행 과정 중에 (선공의 골인한 말의 개수, 후공의 골인한 말의 개수)가 (0,0) → (1,0) → (1,2) → (1,3) → (4,3)의 순서로 변화하였다면(선공의 모든 말이 먼저 골인하였으므로 선공 승리), 아래 표에서 (0,0), (1,0), (1,2), (1,3), (4,3)에 해당하는 부분에서 선공이 승리한 경기로 집계되며, 나머지 부분에서는 집계되지 않는다.

하나의 말이 골인 지점에 도착할 때마다 해당 플레이어의 승률이 대략 10% 상승하는 것을 확인할 수 있다. 본인의 말은

모두 남아있고 상대의 말은 하나만 남아있는 매우 불리한 상황에서도 승률이 20% 정도로 유지되는 것으로 관찰되며, 이는 윷놀이에서 역전이 매우 빈번하게 일어난다는 점을 시사한다.

Table 7. Win Rate of the First Player Given the Number of Mals that Completed Running

First\Second	0	1	2	3
0	50.4%	40.3%	30.8%	19.2%
1	60.1%	50.9%	41.0%	30.2%
2	69.2%	59.1%	49.3%	38.6%
3	80.1%	70.3%	64.4%	49.8%

5. 결 론

본 연구에서 우리는 가치 기반 강화학습 알고리즘을 적용하여 윷놀이 인공지능을 구현하였다. 학습 방식 선택의 타당성 및 학습된 전략의 성능 확인을 위해 다른 인공지능 혹은 사람과의 대결을 통해 승률을 측정하였다. 이후 최종 학습된 전략의 자가대결(self-play)을 통해 윷놀이 게임에 대한 분석을 진행하였다. 추가적인 성능 향상을 위하여 더 깊은 신경망으로 가치함수를 근사하는 것이나 multi-step look-ahead를 통해 더욱 치밀한 수읽기를 수행하는 방법을 고려해볼 수 있을 것이다. 나아가 더욱 다양한 통계분석을 통해 윷놀이에 관한 재미있는 사실들을 추가적으로 밝혀낼 수 있을 것으로 기대한다.

참고문헌

Bertsekas, D. (2012), *Dynamic Programming and Optimal Control: Volume I* (Vol. 4), Athena scientific.

Jiang, Z. and Liang, J. (2017, September), Cryptocurrency Portfolio Management with Deep Reinforcement Learning, In *2017 Intelligent Systems Conference (IntelliSys)*, IEEE, 905-913.

Shannon, C. E. (1950), XXII. Programming a Computer for Playing Chess, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **41**(314), 256-275.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... and Hassabis, D. (2016), Mastering the Game of Go with Deep Neural Networks and Tree Search, *Nature*, **529**(7587), 484-489.

Tesauro, G. (1995), Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, **38**(3), 58-68.

Tromp, J. (2016, June), The Number of Legal go Positions, In *International Conference on Computers and Games*, Cham: Springer International Publishing, 183-190.

Van Hasselt, H., Guez, A., and Silver, D. (2016, March), Deep Reinforcement Learning with Double q-learning, In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 30, No. 1).

Zhu, M., Wang, Y., Pu, Z., Hu, J., Wang, X., and Ke, R. (2020), Safe, Efficient, and Comfortable Velocity Control based on Reinforcement Learning for Autonomous Driving, *Transportation Research Part C: Emerging Technologies*, **117**, 102662.

부 록

1. 점수 기반 휴리스틱 전략

4.1절에서 제시된 점수 기반 휴리스틱 전략은 본 논문에서 제안한 강화학습 기반 최적 전략의 성능 평가를 위해 도입되었다. 점수 기반 휴리스틱 전략은 가치 함수 $\hat{V}_H(\cdot)$를 통해 각 액션 직후 게임 상태의 가치를 평가하여 액션을 선택한다.

$$a_t = \arg\max_{a \in A(s_t)} \hat{V}_H\left(s_{t+1}^{w_t}(s_t, a)\right).$$

시점 t에서 현재 플레이어 입장에서의 점수 기반 휴리스틱 전략 가치 함수 $\hat{V}_H\left(s_t^{w_t}\right)$는 현재 플레이어 말의 위치 점수 $\hat{f}_H(s^{w_t})$, 상대 플레이어 말의 위치 점수에 -1을 곱한 값 $-\hat{f}_H(s^{e_t})$, 다시 한번 윷을 던져야 하는지를 나타내는 상태 변수 $c_t \in \{0,1\}$의 합이 된다.

$$\hat{V}_H\left(s_t^{w_t}\right) = \hat{f}_H(s_t^{w_t}) - \hat{f}_H(s_t^{e_t}) + c_t.$$

이 때, 플레이어 말의 위치 점수 $\hat{f}_H(\cdot)$은 현재 말의 위치에서 도착 지점(<Figure 1> 30번 위치)으로 가기까지 예상 이동 횟수 $\hat{d}_H(\cdot)$와 해당 위치에 있는 플레이어 말의 수 $o_H(\cdot)$를 고려한다.

$$\hat{f}_H(s_t^{w_t}) = \sum_{p \in P} -\hat{d}_H(p) h_H(o_H(p)).$$

$$h_H(x) = \begin{cases} 0, & x = 0, \\ 1, & x = 1, \\ 1.4, & x = 2, \\ 1.2, & x = 3, \\ 0.9, & x = 4. \end{cases}$$

여기서 P는 말이 갈 수 있는 위치들의 집합(<Figure 1> 0~30번 위치)을 의미하며, 말이 업힐수록 큰 위치 점수를 부여하기 위해 $o_H(\cdot)$를 인풋으로 받는 오목함수 $h_H(\cdot)$를 도입하였다. 또한 예상 이동 횟수 $\hat{d}_H(\cdot)$는 value iteration(Bertsekas, 2012)을 통해 계산하였다.

저자소개

송승규 : 서울대학교 산업공학과에서 2023년 학사학위를 취득하고 서울대학교 데이터사이언스학과 석사과정에 재학 중이다. 연구분야는 밴딧, 강화학습이다.

천다호 : 한국과학기술원 산업및시스템공학과 학사과정에 재학 중이다. 연구분야는 최적화, 강화학습이다.

민승기 : 서울대학교 전기정보공학부에서 2014년 학사학위를 취득하고, Columbia University 경영대학에서 2021년 박사학위를 취득하였다. 2021년부터 한국과학기술원 산업및시스템공학과 조교수로 재직하고 있다. 연구분야는 밴딧학습, 강화학습이다.