# Convolutional Neural Network와 Monte Carlo Tree Search를 이용한 인공지능 바둑 프로그램의 구현

기철민* · 조태훈*

*한국기술교육대학교

# Implementation of Artificial Intelligence Computer Go Program Using a Convolutional Neural Network and Monte Carlo Tree Search

Cheol-min Ki* · Tai-Hoon Cho*

*Korea University of Technology and Education

E-mail : kigoon@koreatech.ac.kr

## 요 약

바둑, 체스, 장기와 같은 게임은 사람들의 두뇌발달에 도움을 주어왔다. 이 게임들은 컴퓨터 프로그램으로도 개발되었으며, 혼자서도 게임을 즐길 수 있도록 많은 알고리즘들이 개발되었다. 사람을 이기는 체스 프로그램은 1990년대에 개발된 것에 비해 바둑은 경우의 수가 너무 많아서 프로 바둑 기사를 이기기는 불가능한 것으로 여겨졌다. 하지만 MCTS(Monte Carlo Tree Search)와 CNN(Convolutional Neural Network)의 이용으로 바둑 알고리즘의 성능은 큰 향상을 이루었다. 본 논문에서는 CNN과 MCTS를 사용하여 바둑 알고리즘의 개발을 진행하였다. 바둑의 기보가 학습된 CNN을 이용하여 최적의 수를 찾고, MCTS를 이용하여 게임의 시뮬레이션을 진행하여 이길 확률을 계산한다. 또한 기존 기보를 이용하여 바둑의 패턴 정보를 추출하고, 이를 이용하여 속도와 성능 향상을 도모하였다. 이 방법은 일반적으로 사용되는 바둑 알고리즘들에 비해 성능 향상이 있었다. 또한 충분한 Computing Power가 제공되면 더욱 성능이 향상될 것으로 보인다.

## ABSTRACT

Games like Go, Chess, Janggi have helped to brain development of the people. These games are developed by computer program. And many algorithms have been developed to allow myself to play. The person winning chess program was developed in the 1990s. But game of go is too large number of cases. So it was considered impossible to win professional go player. However, with the use of MCTS(Monte Carlo Tree Search) and CNN(Convolutional Neural Network), the performance of the go algorithm is greatly improved. In this paper, using CNN and MCTS were proceeding development of go algorithm. Using the manual of go learning CNN look for the best position, MCTS calculates the win probability in the game to proceed with simulation. In addition, extract pattern information of go using existing manual of go, plans to improve speed and performance by using it. This method is showed a better performance than general go algorithm. Also if it is receiving sufficient computing power, it seems to be even more improved performance.

## 키워드

Machine Learning, Convolutional Neural Network, Monte Carlo Tree Search, Computer Go

## Ⅰ. 서 론

바둑, 체스, 장기와 같은 게임은 과거부터 사람들의 두뇌발달에 도움을 주었고, 이 게임들은 컴퓨터 프로그램으로 개발되어 많은 사람들이 컴퓨터로 게임을 즐길 수 있게 되었다. 하지만 위에서

언급한 게임들은 모두 두 명의 사람이 진행을 하는 게임이기 때문에 혼자서도 게임을 즐길 수 있도록 알고리즘들이 개발되어 왔다. 가장 뛰어난 인공지능 체스 프로그램인 IBM의 Deep Blue는 1997년 체스마스터 Gary Kasparov를 이기며 체스를 정복한 것과 달리 바둑은 체스에 비해 경우의 수가 상당히 많아 American Go Association에 따르면 $10^{700}$개의 가능한 게임이 있다고 하며, 훌륭한 알고리즘 전략이 발견되지 않아 프로 바둑 기사를 이기기에는 불가능한 것으로 여겨졌다.

하지만 MCTS(Monte Carlo Tree Search)[1]를 바둑에 접목시키면서 게임 트리의 탐색 범위가 상당히 줄어 바둑 프로그램의 상당한 성능향상이 있었는데, 바둑 프로그램 중 하나인 Pachi[2]는 바둑판의 Pattern 정보[3]와 MCTS를 이용하였고, MCTS Playout 수를 10만 번으로 맞췄을 때 KGS Server[4] 3단 정도의 성능을 보이게 되었다.

이 후 CNN(Convolutional Neural Network)[5,6]과 MCTS를 접목시켜 더욱 성능 향상을 이끌어낸 Facebook AI Research의 DarkforestGo[7]가 KGS Server 5단의 성능을 보였으며, 2016년 Google Deepmind의 AlphaGo[8]는 프로 바둑 기사 이세돌을 이김으로써 바둑도 정복될 수 있음을 보여주었다. 본 논문에서는 MCTS와 CNN을 이용하여 바둑 알고리즘의 구현을 진행했으며, 테스트는 기존에 존재하는 바둑 알고리즘과의 테스트를 진행하였다.

## Ⅱ. 관련연구

바둑 알고리즘에서 최적의 수를 판별하는데 사용하는 알고리즘은 MCTS와 CNN이다.

MCTS는 모든 경로를 탐색하기 불가능한 상황에서 효율적인 트리 탐색 방법이다. MCTS는 Minimax 알고리즘의 성능을 개선한 방법이기 때문에 최대, 최소값이 존재해야 하며, 규칙이 정해져있고, 게임의 길이가 제한적일 때 사용가능하다. 또한 가장 최적의 결정을 위해 검색 공간에서 무작위 추출에 기초한 탐색 트리를 확장하는데 중점을 둔다. MCTS는 선택(Selection), 확장(Expansion), 시뮬레이션(Simulation), 역전달(Backpropagaion)의 4단계로 구성되어있다.

Selection 과정에서는 현재 노드에서 어떤 자식노드를 선택해서 움직일지를 결정하기 위해 각 노드에서 Selection Function을 이용하여 Selection Value를 계산하고, 가장 높은 Selection Value를 가진 자식노드를 선택하여 이동한다.

Expansion 과정에서는 선택 과정에서 선택할 때, 다음에 이동해서 멈출 노드의 확률을 생성한다. 보통 한 노드의 방문 횟수가 일정한 임계값에 도달하면 자식노드를 확장한다.

Simulation 과정에서는 확장 과정을 거쳐 만들어진 노드부터 임의의 게임을 시뮬레이션 하는 과정이다. 여기서 MCTS의 제한조건으로 게임의 길이가 제한적이어야 한다는 조건이 적용되며, 게임이 끝날 때까지 Random한 위치를 선정하여 움직이고, 최종적으로 어떤 Player가 이겼고, 선택한 노드의 경로를 유지한다.

Backpropagation 과정에서는 시뮬레이션 후 최종적인 결과와 노드의 경로를 다시 Root로 올라가는 과정이며, 각 확률 노드에 대해 Visit Count와 Win Count를 증가시키게 된다.

이 4개의 과정을 설정한 Playout 횟수나 설정한 시간만큼 반복하며 Win Count와 Visit Count를 증가시키며 최종적으로 자식노드 중 Win/Visit Count 비율이 가장 높은 쪽으로 움직이는 것이 이기는 경로라고 판별한다. 아래 <그림 1>은 MCTS의 4단계를 그림으로 보여준다.

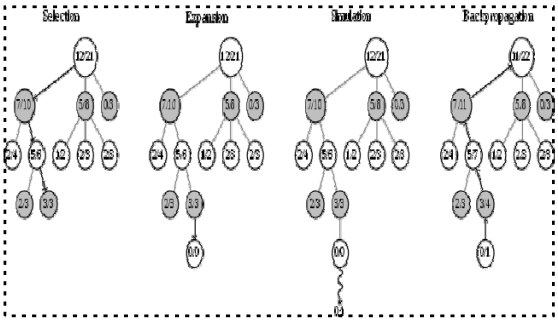


그림 1 . Monte Carlo Tree Search의 4단계

CNN은 다양한 컴퓨터 비전 분야에서 많은 연구가 활발히 진행 중인 인공신경망의 한 종류이다. CNN은 일반적으로 몇 개의 층으로 이루어져 있으며 기본적으로 Convolution Layer, Pooling Layer, Feedforward Layer의 3가지의 다른 층을 가지고 있다.

Convolution Layer는 입력한 데이터에 Convolution을 적용하여 Convolution Feature를 추출하는 층이다. 이는 유의미한 자질을 추출하는 층이라고 할 수 있다.

Pooling Layer는 Convolution Layer에서 추출한 Convolution Feature를 Sub-Sampling하는 과정이다. 이미지의 특성상 픽셀의 개수가 너무 많고, 연산 횟수가 상당히 많아지기 때문에 속도적인 향상을 위해 영상을 줄여나가는 과정이라 할 수 있다.

Feedforward Layer는 일반적인 Neural Network와 구조가 동일하며 마지막으로 적용된다. 이는 Convolution Layer와 Pooling Layer의 반복 구조에서 나온 Feature들을 이용하여 분류를 할 때 사용된다.

CNN은 위에서 설명한 Convolution Layer와 Pooling Layer를 번갈아 사용하여 Convolution Feature를 추출하고 마지막으로 Feedforward Layer를 이용하여 분류를 진행한다. 학습 과정에서 모든 Layer에 대한 추측을 진행할 때는 Forward Propagation, 각 Layer의 Weight나 Bias

등의 Parameter를 갱신할 때는 Backpropagation을 진행한다. 아래 <그림 2>는 CNN의 대략적인 구조를 보여준다.

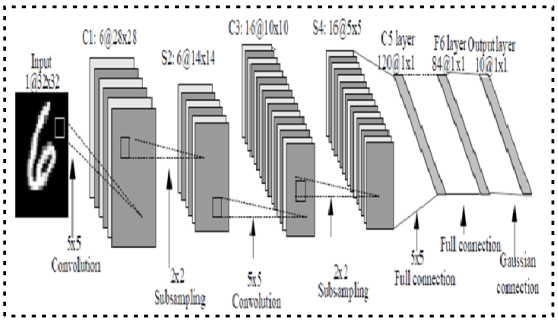


그림 2 . Convolutional Neural Network의 구조

## Ⅲ. Convolutional Neural Network와 Monte Carlo Tree Search를 이용한 인공지능 바둑 프로그램의 구현

본 논문에서는 위에서 설명한 CNN과 Monte Carlo Tree Search를 이용하여 현재 상태 이후의 착수 점을 선정한다.

MCTS의 Selection 과정에서는 UCB(Upper Confidence Bounds)를 결합한다. MCTS와 UCB를 결합한 알고리즘을 UCT(Upper Confidence bounds applied to Trees)라고 하는데 pure MCTS보다 UCT가 더욱 빠른 결과 수렴을 진행하여 속도가 빨라지는 효과를 보여준다. 다음 식 (1)은 본 논문에서 적용한 UCT의 UCB1 Value를 구하는 수식이다.

$$V = x_j + \sqrt{\frac{2\log n}{T_j(n)}}, \quad (1)$$

식 (1)에서 $x_j$는 어떠한 착수 $a_j$를 선택했을 때의 이길 확률이고, $T_j$는 $a_j$가 선택된 횟수, n은 전체 횟수를 나타낸다. 위의 식을 이용해서 나타난 값이 최대가 되는 착수 $a_j$를 선택한다. 또한 CNN을 결합하여 무작위 탐색에서 MCTS로 줄인 범위를 CNN을 이용하여 착수 후보 점들을 찾아내고, 위에서 계산한 UCB1 Value 값과 후보 점들의 값을 더해 최종 착수 점을 선택하게 된다.

여기서 사용한 CNN의 학습 데이터는 아마추어 4단 이상의 핸디캡이 없는 데이터만을 선정하여 13만개 가량의 데이터를 추출하고 각각의 바둑 기보에서 19x19의 영상을 만들어 Multi Channel로 다양한 Feature를 추출해낸다. 본 논문에서 사용한 Feature는 Facebook AI Research의 논문[7]에서 나타난 Standard Feature와 동일한 19x19의 21 Channel 영상이며 Feature에 대한 설명은 다음 <표 1>과 같다.

표 1 . 기보에서 추출한 Feature

| Plane Name | Type | 개수 |
|---|---|---|
| Our/Opponent Liberty | binary | 6 |
| Ko | binary | 1 |
| Our/Opponent Stones/Empty | binary | 3 |
| Our/Opponent History | real | 2 |
| Opponent Rank | binary | 9 |

위와 같이 나타난 Feature들을 CNN의 Input Image로 사용하여 학습을 진행하는데 CNN의 구조를 나타낸 그림은 다음과 같다.

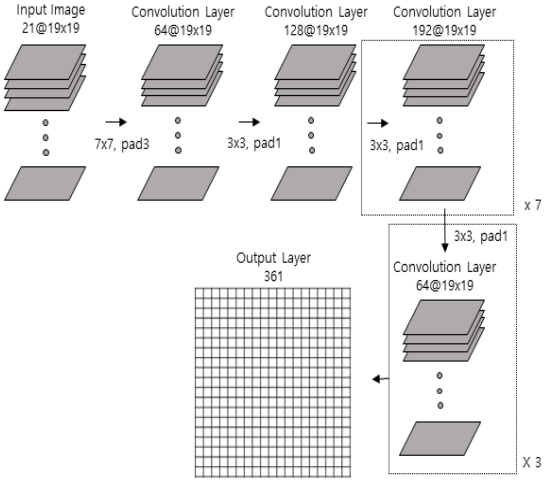


그림 3 . 착수 점 선택을 위한 CNN의 구조

위에서 나타난 <그림 3>과 같이 구성된 CNN으로 학습을 진행하여 최종적인 Output으로는 19x19의 바둑돌 좌표의 한 점이 출력된다. 본 논문에서는 CNN을 학습시킬 때 Mini Batch Size를 64로 설정하고 160만 번의 Iteration 후 테스트 세트에 대한 정확도는 59%의 정확도가 나타났다. CNN에서 선택하는 최적의 수는 5개의 수를 선택하며 이 수들에서 Expansion 과정을 진행하게 된다. 또한 같은 기보 파일들에서 Move Pattern[9]을 추출하여 MCTS의 Selection 과정에서 너비를 줄이는데 추가적인 도움을 주어 성능 향상과 속도의 향상을 도모했다.

Expansion 과정에서는 노드의 확장을 진행하는 기준을 방문 횟수가 10회 이상인 경우에 확장을 하는 것으로 선정하여 확장을 진행했으며, Simulation 과정, Backpropagation 과정 모두 위에서 설명한 기본적인 MCTS 방식과 동일하게 동작하여 최종적으로 가장 많이 방문된 위치의 돌이 최적의 수라는 판단을 하여 다음 착수 점으로 정해지게 된다.

이 후 정해진 착수 점이 Illegal Move이면 다음 후보 착수 점을 선택하여 돌을 두게 되고, 그렇지 않으면 최적의 착수 점을 두게 한다.

## Ⅳ. 실험 결과

본 논문의 실험 환경은 Xeon CPU E3-1225 V5 (3.30GHz, 4-Core), 16GB RAM, NVidia Geforce 750Ti의 시스템과 Ubuntu 14.04에서 컴파일러 gcc 4.9, g++ 4.9를 사용하였고, openCV 2.4.9와 딥러닝 프레임워크인 Caffe Framework를 이용하였다. 또한 GPU를 사용하여 Playout의 속도를 높이기 위해 Cuda Library, Cudnn을 사용하였고, 바둑의 기본적인 룰과 MCTS의 사용을 위해 Open Source 프로그램인 Oakfoam[10]을 이용하였으며, 바둑 프로토콜인 Gtp를 이용하여 다른 엔진과의 대국을 위해 Gtp Library와 GUI 화면을 보여주는 Gogui-1.4.9를 사용하였다. 테스트는 Open Source 바둑 프로그램인 Pachi의 Playout 횟수를 10k, 100k로 설정하여 테스트를 진행하였으며, 본 논문에서 구현한 바둑 프로그램은 UCT의 관련 파라미터들을 조율하며 테스트를 진행했다. 각 테스트는 흑돌 5판, 백돌 5판으로 총 10판의 테스트를 진행했으며, Pachi의 Playout에서 걸리는 시간과 동일하게 세팅을 맞추고 테스트를 진행했다.

표 2. Pachi와 구현한 바둑 프로그램의 테스트 결과

| | Pachi 10k | Pachi 100k |
|---|---|---|
| MCTS+CNN (parameter tuning X) | 80% | 30% |
| MCTS+CNN (parameter tuning O) | 100% | 70% |

Pachi의 10k, 100k는 Playout 횟수를 각각 10000번, 100000번으로 세팅하고 진행한 결과이며, 위 <표 2>에서 나타난 MCTS+CNN은 본 논문에서 구현한 바둑 프로그램을 나타낸다.

MCTS+CNN의 Playout 수는 각각 10k, 100k의 수행시간과 동일하게 설정했을 때 각각 5000, 40000의 Playout을 설정했으며, Pachi 10k에 대해 나타난 결과는 UCT Parameter를 Default로 맞춘 결과가 80%, UCT Parameter를 튜닝한 결과가 100%의 결과가 나타났다. 또한 Pachi 100k와의 대국에서는 UCT Parameter를 Default로 맞춘 결과가 30%, 튜닝한 결과는 70%의 승률을 얻을 수 있었다.

## Ⅴ. 결 론

Open Source로 공개된 프로그램 중 가장 강력한 프로그램인 Pachi와의 대국을 통해 본 논문에서 구현한 방법이 더욱 좋은 성능을 보인 것을 테스트를 통해 확인할 수 있었다. Pachi의 KGS Server 추정 단수는 Pachi 100k가 Stable한 3단 가량으로 추정되고 있으며, 위의 결과에 따라 본 논문에서 구현한 방법은 KGS Server 3단 - 4단 가량으로 추정된다. 더욱 정확한 테스트 결과를 나타내기 위해선 더욱 많은 대국을 진행해야 하나, Single Machine의 속도적인 한계로 10판으로 제한되는 대국을 진행했다.

지금보다 더욱 성능을 높이기 위해서는 Playout 횟수를 증가시키는 방법과, CNN을 더 높은 단수의 기보 파일들로 학습시키는 방법, UCT Parameter의 최적화 그리고 AlphaGo에서 사용한 Reinforcement Learning과 Value Network를 이용하는 방법들이 있는데, 이는 차후 개선해야할 문제이며 MCTS만 이용했을 때 보다 CNN과 MCTS를 결합한 바둑 프로그램이 더 좋은 결과를 보였다.

## 참고문헌

[1] MCTS, https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

[2] Pachi, http://pachi.or.cz/

[3] Sylvain Gelly, Yizao Wang, Remi Munos, Olivier Teytaud, “Modification of UCT with Patterms in Monte-Carlo Go”, Technical report, INRIA, 2006.

[4] KGS Go Server, https://www.gokgs.com/

[5] Y. LeCun, L. Bottou, Y. Bengio, and P.Haffner, “Gradient-Based Learning Applied to Document Recognition“, Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[6] Patrice Y. Simard, Dave Steinkraus, John Platt, “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis“, External Link International Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society, Los Alamitos, pp. 958-962, 2003.

[7] Yuandong Tian, Yan Zhu, “Better Computer Go Player With Neural Network and Long-term Prediction”, ICLR, 2016.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. “Mastering the game of go with deep neural networks and tree search”, Nature, pp. 484-529, 2016.

[9] Remi Coulom, “Computing Elo Ratings of Move Patterns in the Game of Go”, ICGA, 2007.

[10] Oakfoam, http://oakfoam.com