2019 OSS 개발자 포럼 겨울 캠프

AlphaZero - Day 2

옥찬호

Nexon Korea, Microsoft MVP

utilForever@gmail.com

오늘 다룰 내용

- PyTorch 사용법
- 신경망 구조 설계
  - 입력 데이터 전처리
  - 신경망 구조
- MCTS 변형
- 학습 시키기
  - 학습 데이터 만들기
  - 학습
- Multiprocessing

PyTorch 사용법

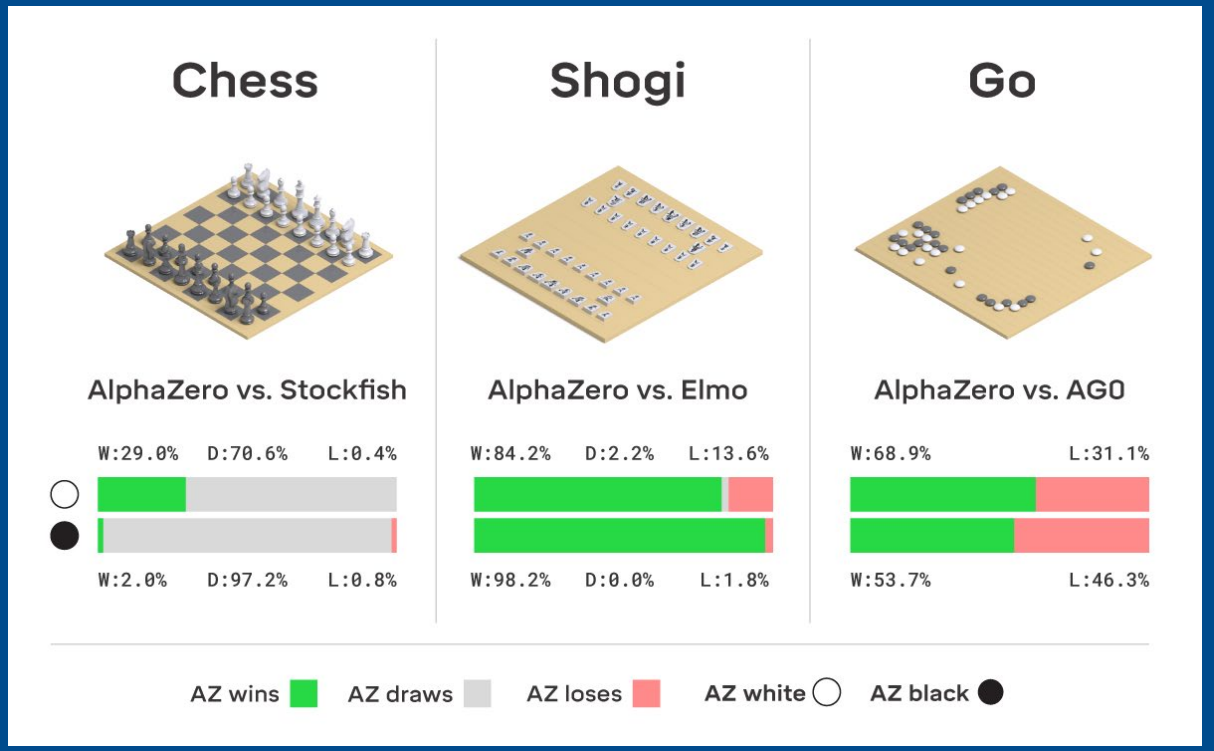
2019 OSS Winter
AlphaZero 오목 AI - Day 2

Google Colab 링크
http://bitly.kr/oqX1Bac

AlphaZero

Google DeepMind에서 2017년 발표한 강화학습 모델이다.

게임의 규칙만 알려주면 **사람의 지식 없이** 보드게임을 통달할 수 있다.



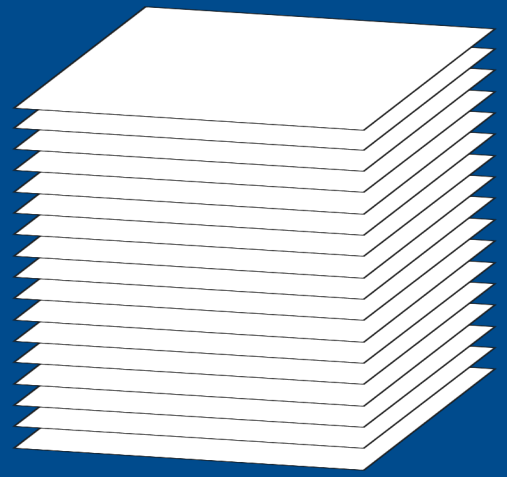
입력 데이터 전처리

오목판의 상태를 신경망에 넣어 주기 위한 처리를 해야 한다.
→ 이를 전처리(Preprocess)라 부른다.

입력 데이터의 모양은 $7 \times 7 \times 17$ 이다.

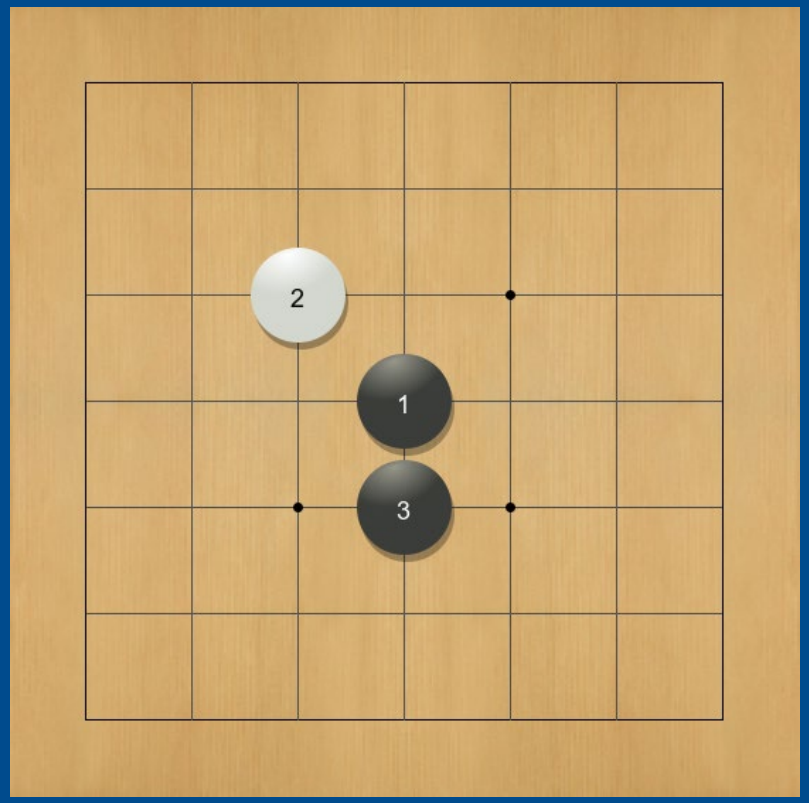
- X_t : 나의 돌 위치 정보
- Y_t : 상대의 돌 위치 정보
- C : 나의 돌 색상 정보 (흑이면 전부 1, 백이면 전부 0)

→ 입력 데이터 $s_t = [X_t, Y_t, X_{t-1}, Y_{t-1}, \cdots, X_{t-7}, Y_{t-7}, C]$



입력 데이터 전처리

돌의 위치 정보



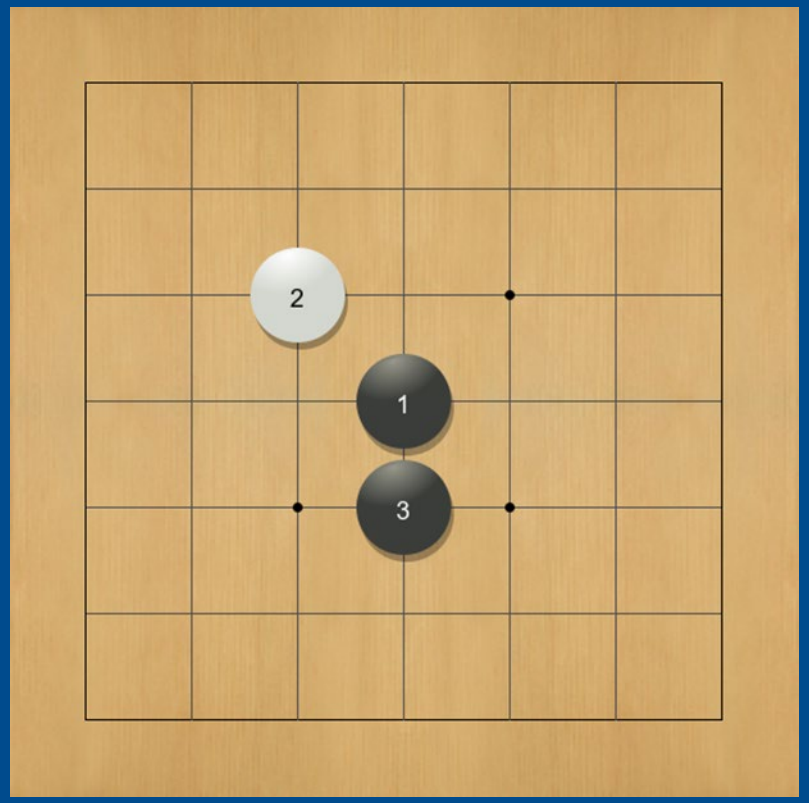
백이 둘 차례

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

0번 채널

입력 데이터 전처리

돌의 위치 정보



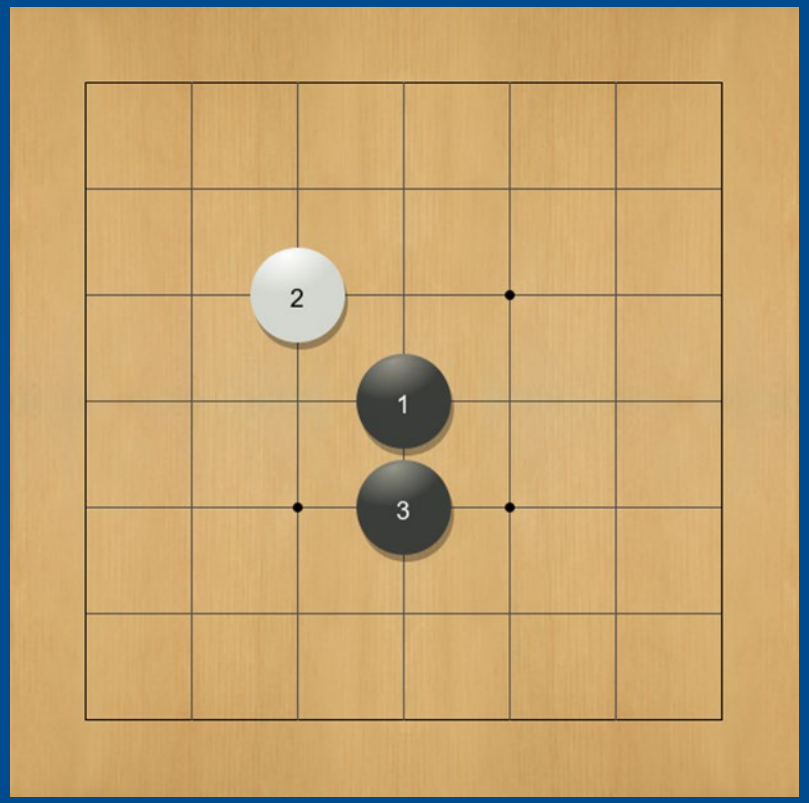
백이 둘 차례

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

1번 채널

입력 데이터 전처리

돌의 위치 정보



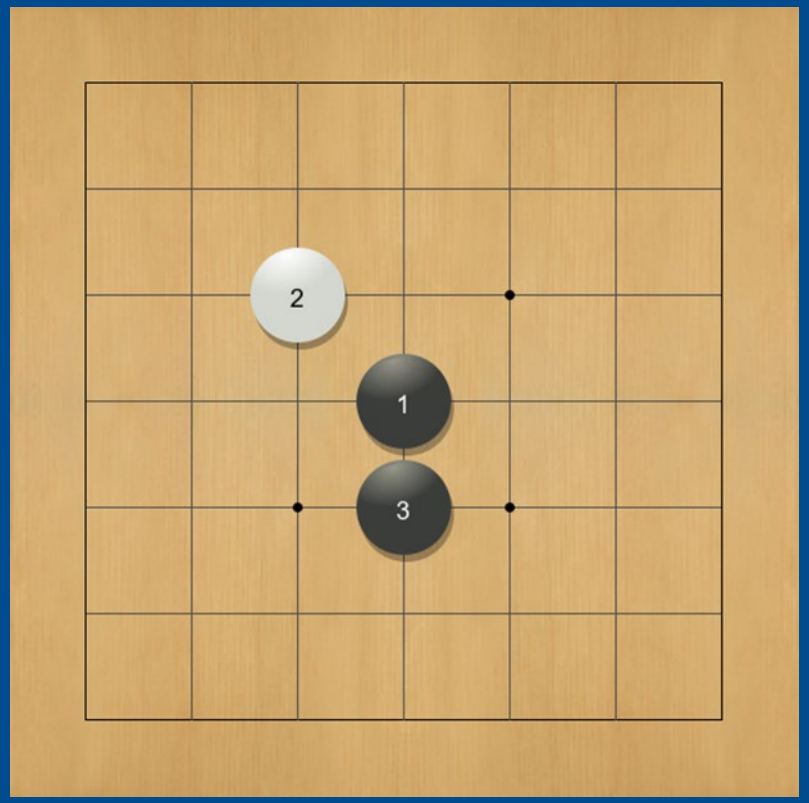
백이 둘 차례

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

2번 채널

입력 데이터 전처리

돌의 위치 정보



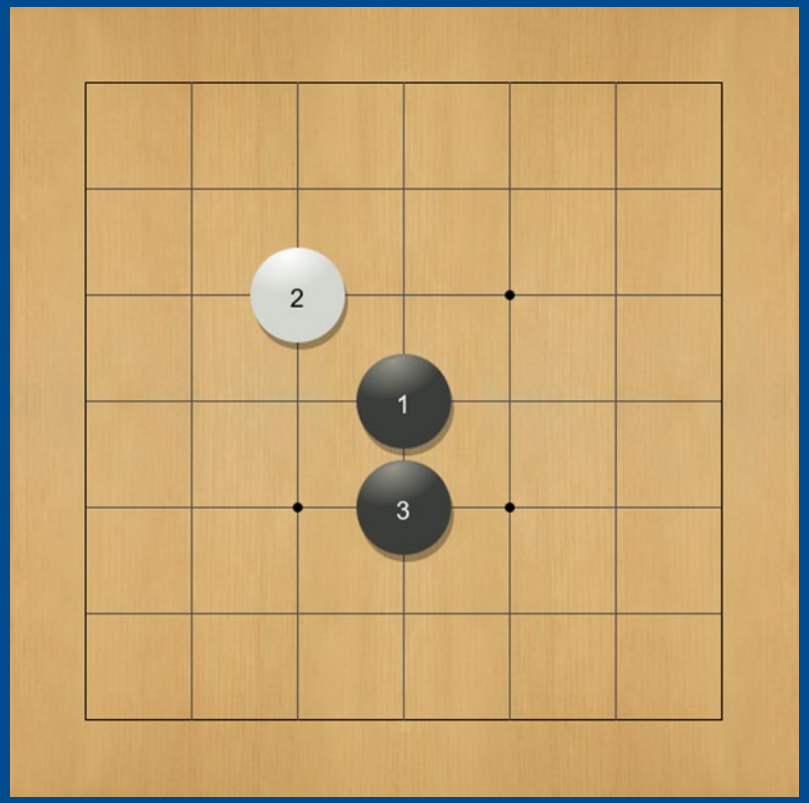
백이 둘 차례

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

3번 채널

입력 데이터 전처리

돌의 위치 정보



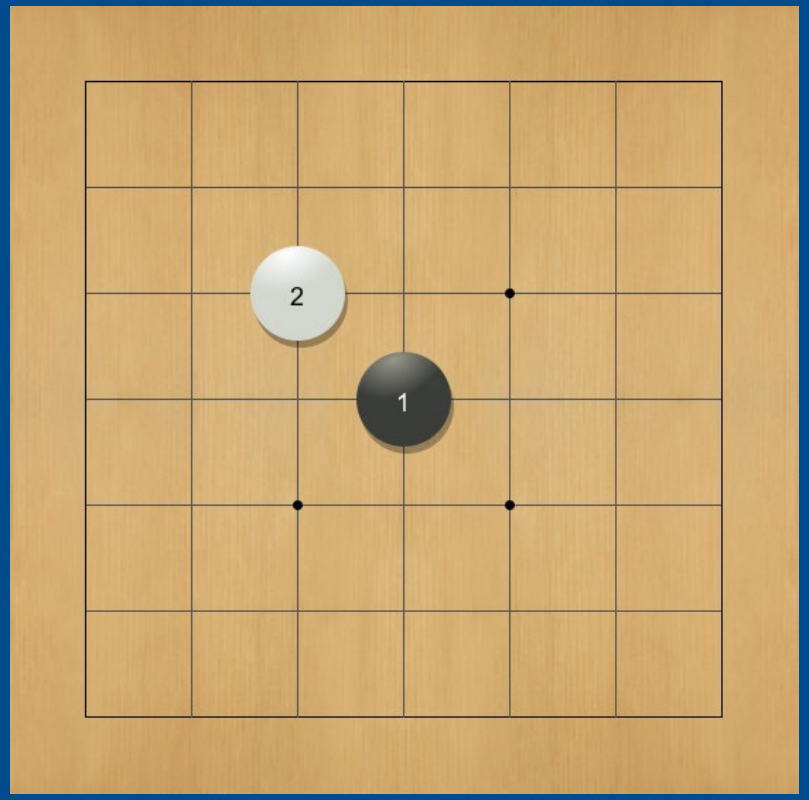
백이 둘 차례

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

마지막 채널

입력 데이터 전처리

돌의 위치 정보



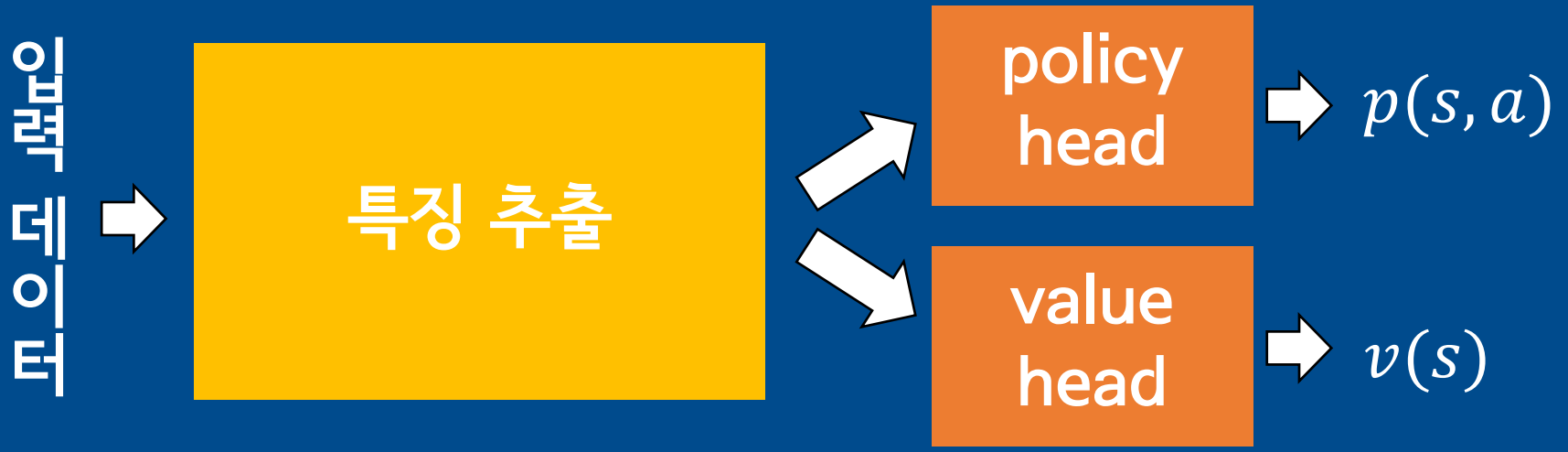
백이 둘 차례

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

마지막 채널

신경망 구조 설계

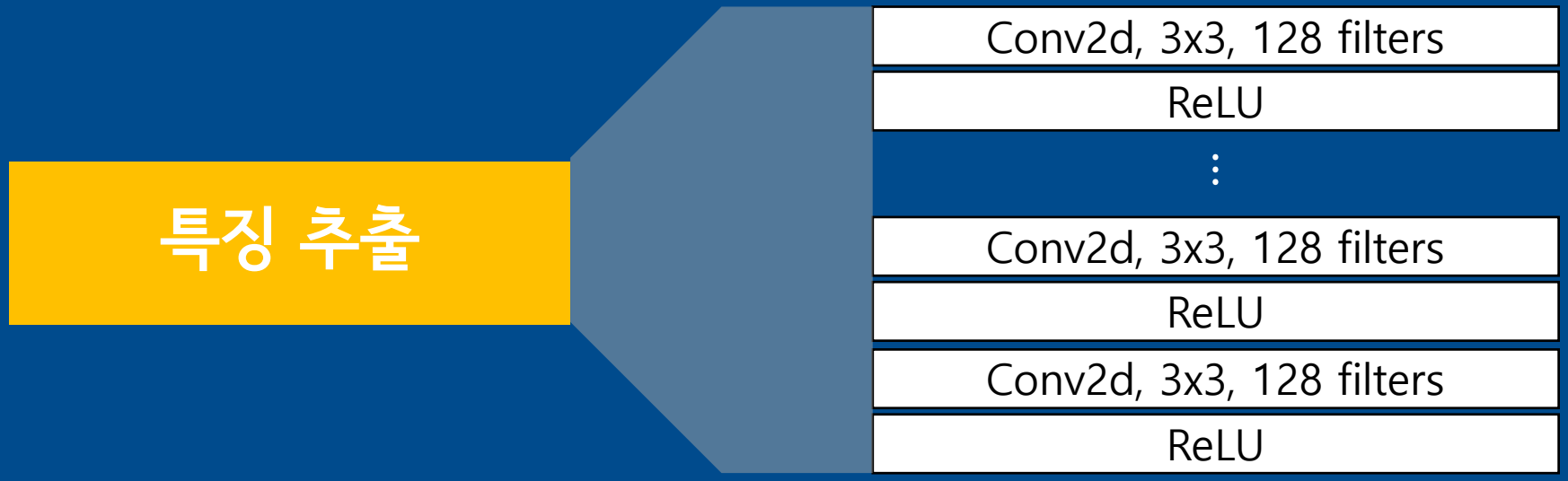
전처리한 입력 데이터를 받아 정책(policy)과 가치(value)를 추정하는 신경망을 만들어야 한다.



신경망 구조 설계

특징 추출

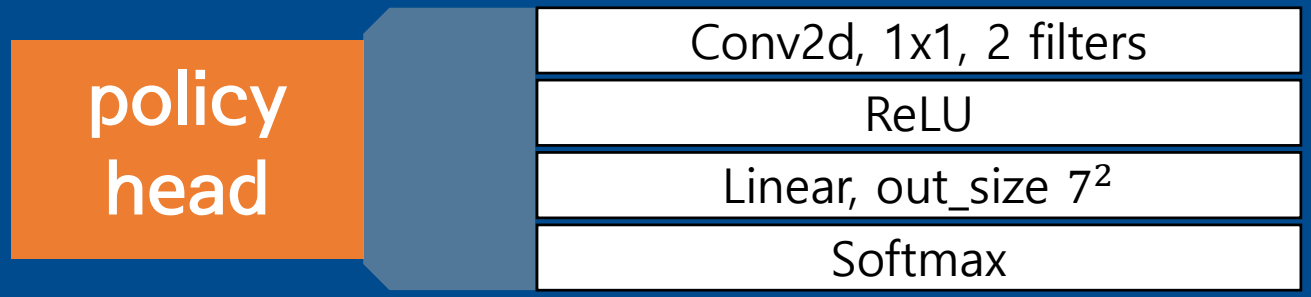
- 오목 판은 가로 세로 15픽셀의 이미지로 볼 수 있다.
- 이미지에서 특징을 추출할 때 사용하는 CNN(Convolutional Neural Network)를 사용.
- 최종적으로 $7 \times 7 \times 128$ 크기의 특징맵(feature map)이 나온다.



신경망 구조 설계

Policy Head

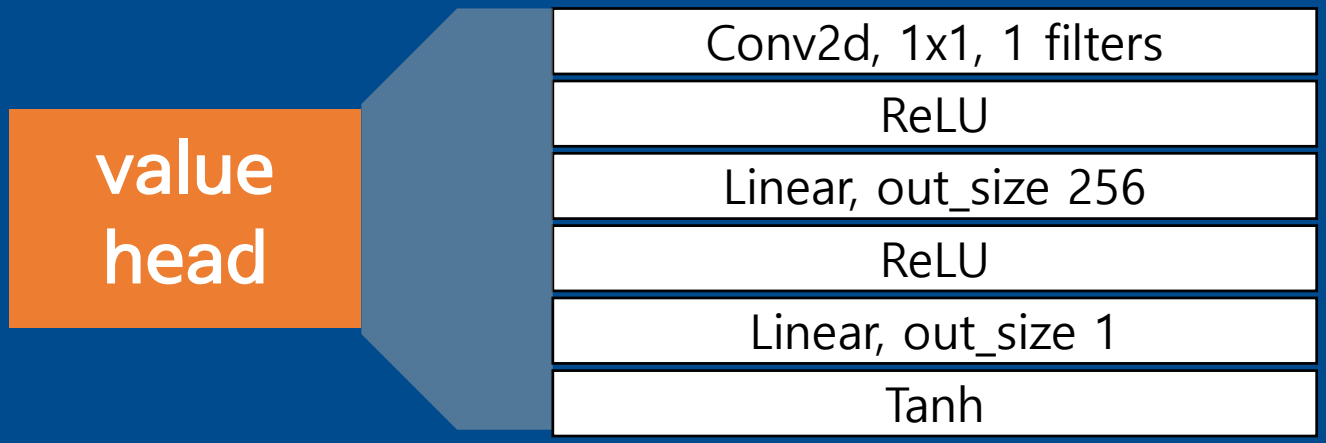
- 추출된 특징맵에서 정책을 구하는 부분.
- 판 위의 모든 점에 대한 정책인 크기 49인 벡터가 나온다.



신경망 구조 설계

Value Head

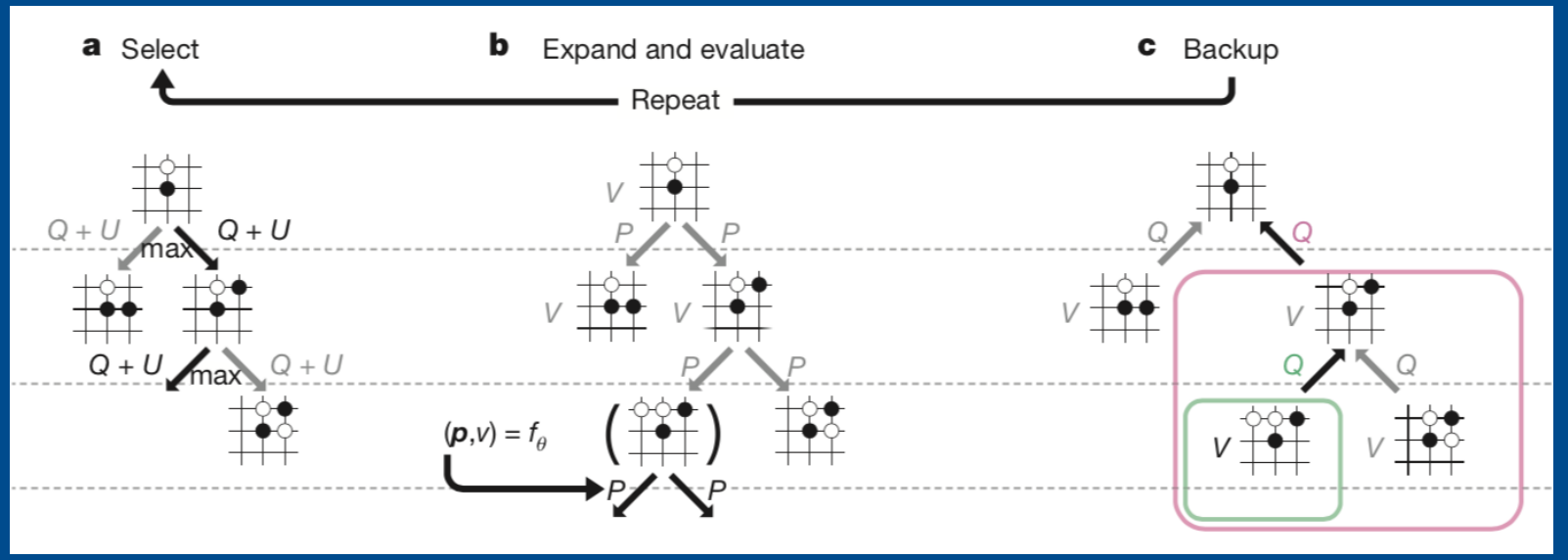
- 추출된 특징맵에서 가치를 구하는 부분.
- 크기가 1인 승률이 나온다.



MCTS 변형

AlphaZero는 변형된 MCTS를 사용한다.

- PUCT(Polynomial Upper Confidence Trees) 알고리즘 사용
- Rollout 제거



MCTS 변형

PUCT 알고리즘

탐색할 수는 다음 수식에 의해 결정한다.

$$a_t = \text{argmax}_a(Q(s_t, a) + U(s_t, a))$$

이때 $Q(s_t, a)$는 평균 승률로 다음과 같이 정의된다.

$$Q(s_t, a) = \frac{W(s_t, a)}{N(s_t, a)}$$

MCTS 변형

PUCT 알고리즘

$U(s_t, a)$는 다음과 같이 정의된다.

$$U(s_t, a) = C(s_t)P(s_t, a)\frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)}$$

그리고 이전엔 상수였던 temperature가 다음과 같은 함수 $C(s_t)$로 바뀌었다.

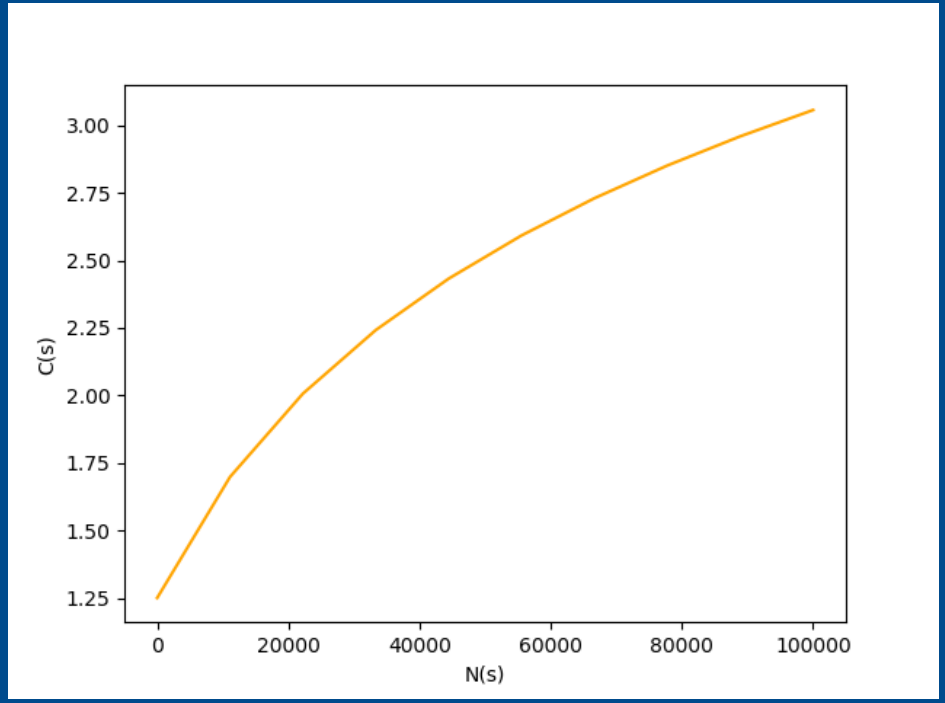
$$C(s_t) = \log_e\left(\frac{1 + N(s_t, a) + c_{base}}{c_{base}}\right) + c_{init}$$

MCTS - Select

$C(s)$는 방문 횟수가 클수록 값이 커진다.

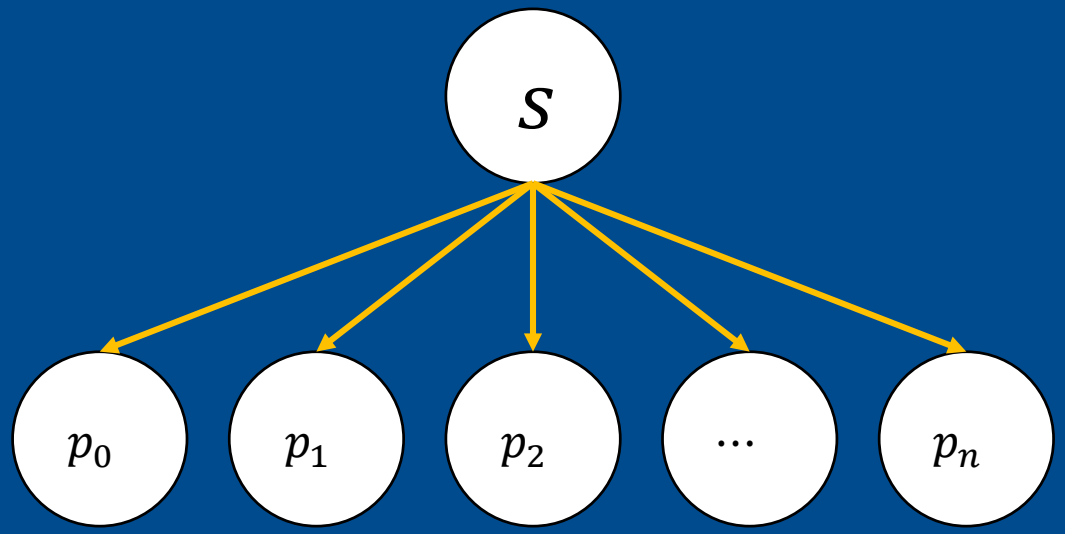
→ 해당 노드를 많이 방문할수록 다른 자식 노드도 탐색할 수 있게 하기 위함.

하지만, 빠르게 진행되는 게임에선, 상수일 필요가 있다.



MCTS - Expand

신경망에서 나온 값 $f_\theta(s) = (\boldsymbol{p}, v)$ 중 $\boldsymbol{p}$를 바탕으로 다음 수를 예측한다.



MCTS - Backup

Alpha Zero에선 Rollout을 제거하고, 대신 승률 추정만 한다.

따라서 Backup 단계에서 각 노드의 값은 다음과 같이 변한다.

$$N(s_t, a) = N(s_t, a) + 1$$
$$W(s_t, a) = W(s_t, a) + v$$

v : 신경망에서 나온 값

MCTS - Play

특정 조건을 만족하면 시뮬레이션을 멈추고 착수 위치를 결정 한다.

착수 위치 결정엔 다양한 방법이 존재한다.

- 가장 많이 방문한 node를 선택
- 가장 승률이 높은 node를 선택
- 여러 수치를 적절히 결합해 node를 선택

→ AlphaZero에선 가장 많이 방문한 node를 선택한다.

학습 데이터 만들기

학습 데이터는 상태(s), 정책(π), 게임 결과(z)의 쌍으로 구성된다.

정책은 MCTS에서 선택한 수를 최적의 수로 가정하여 학습한다.

→ 정책 $\pi(s,a) = \frac{N(s,a)}{\sum_b N(s,b)}$

가치는 게임의 결과를 예측하도록 학습한다.

→ $z = \begin{cases} 1 & (if\ win) \\ 0 & (if\ draw) \\ -1 & (if\ lose) \end{cases}$

학습 데이터 만들기

강화학습에서 탐험은 매우 중요하고, 현재 모델에 과적합 되는 걸 막기 위해 다음과 같이 noise를 섞는다.

$$P(s,a) = (1-\epsilon)p_a + \epsilon\eta_a \quad \text{where } \eta \sim \text{Dir}(\alpha)$$

$\text{Dir}(\alpha)$는 python에서 다음과 같이 구할 수 있다.

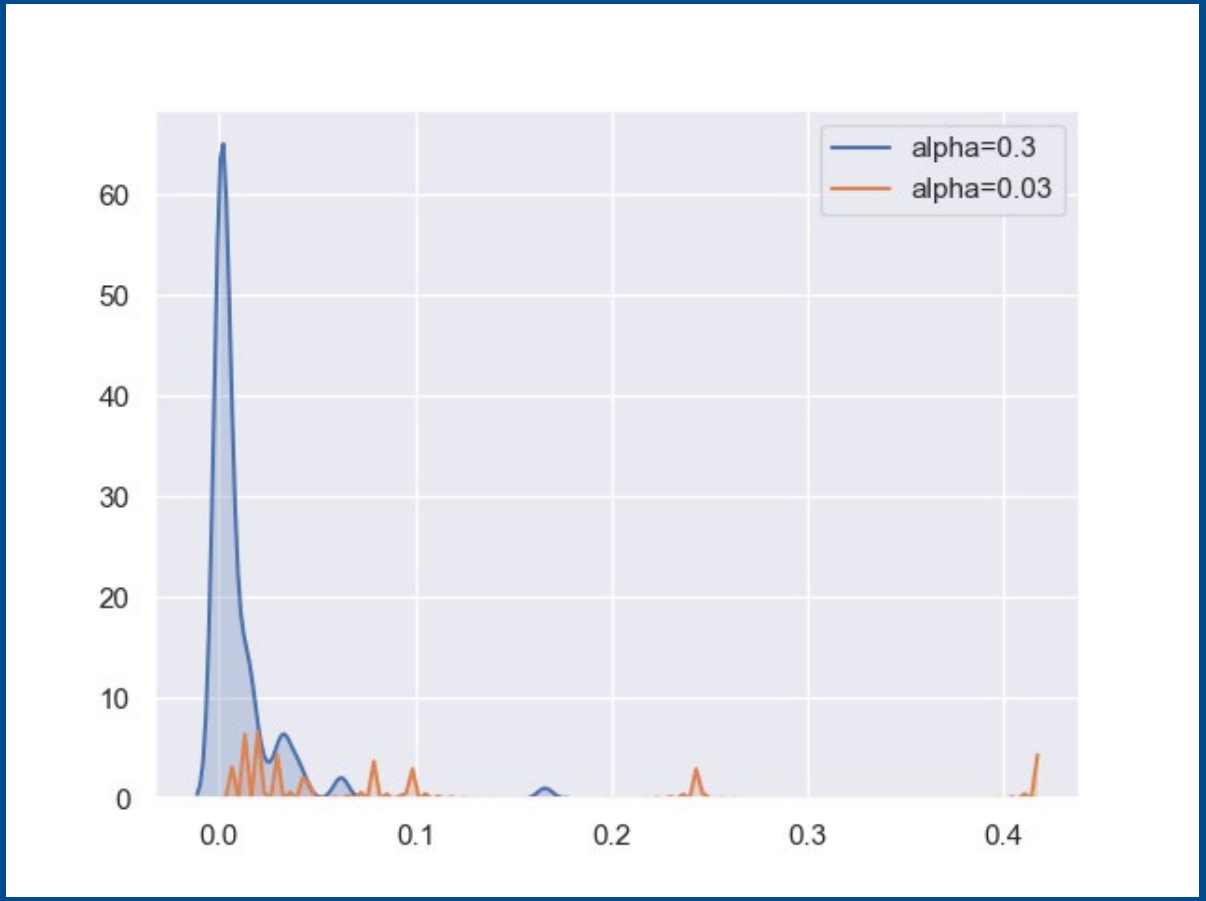
```
import numpy

alpha = 0.03
noise = numpy.random.dirichlet([alpha] * 256)
```

학습 데이터 만들기

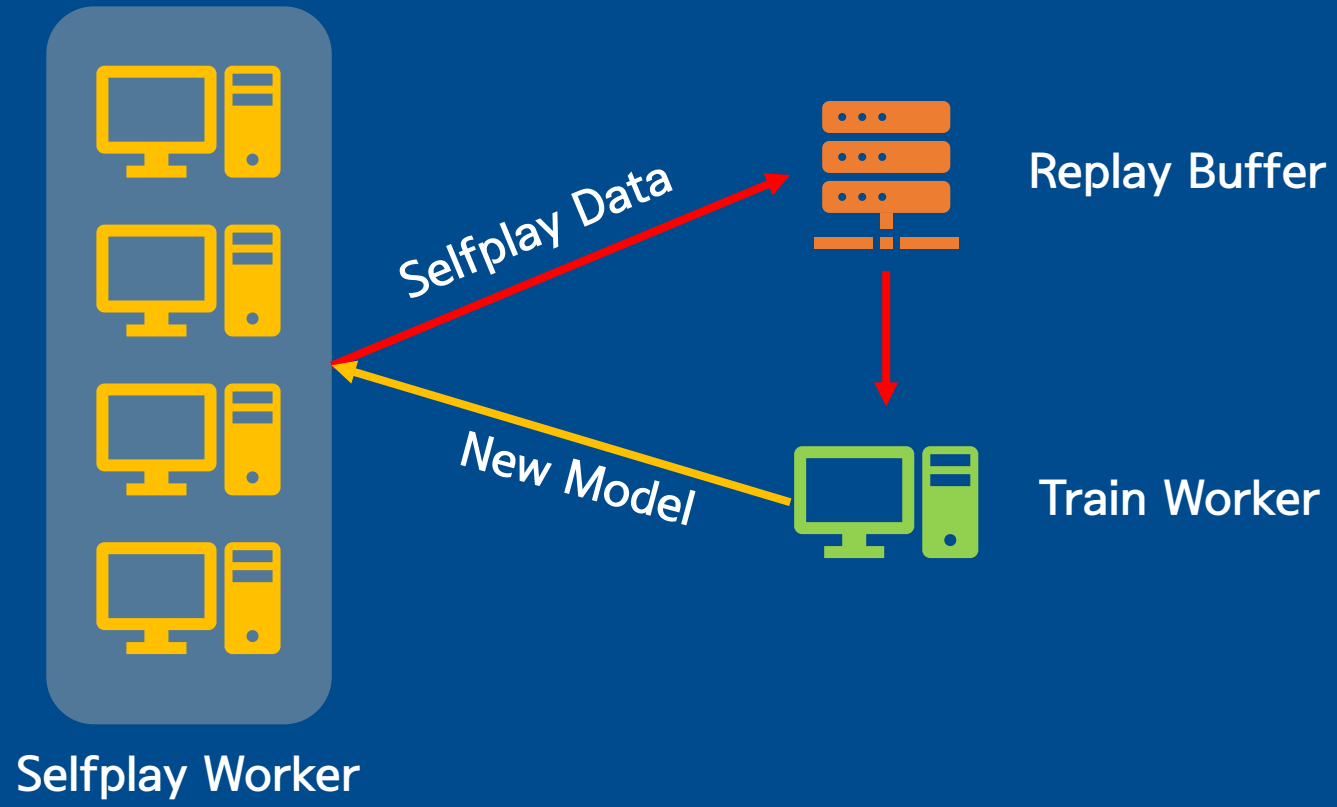
α가 작을수록 더 넓은 분포를 갖는다.

→ 탐색 범위가 넓어진다



학습 시키기

만들어진 학습 데이터는 Replay Buffer에 저장된다.



학습 시키기

- Selfplay Worker
  - 학습데이터를 만든다.
  - Train Worker에서 학습된 모델을 받아 학습 데이터를 만든다.
- Replay Buffer
  - Selfplay Worker에서 만들어진 데이터를 저장하는 곳이다.
- Train Worker
  - Replay Buffer에서 데이터를 가져와 새로운 신경망을 학습한다.

학습 시키기

신경망을 다음의 loss function으로 최적화한다.

$$L = (z - v)^2 - \boldsymbol{\pi}^\top \log \boldsymbol{p} + c\|\theta\|^2$$

- $(z - v)^2$: 신경망의 승률 추정치 v와 게임의 결과 z 사이의 MSE(Mean Square Error)
- $-\boldsymbol{\pi}^\top \log \boldsymbol{p}$: 신경망의 정책 $\boldsymbol{p}$와 MCTS의 정책 $\boldsymbol{\pi}$ 사이의 Cross Entropy
- $c\|\theta\|^2$: overfitting을 방지하기 위한 Weight Decay

학습 시키기

학습은 다음의 순서대로 진행된다.

1. 자가대국을 통해 데이터를 만들고 Replay Buffer에 넣는다.
2. 자가대국 데이터를 Replay Buffer에서 가져온다.
3. Loss를 계산해 기울기를 계산한다.
4. 계산된 기울기를 바탕으로 신경망을 학습시킨다.
5. 새로운 신경망으로 자가대국을 한다.

multiprocessing

CPU에는 연산을 하는 부품인 코어가 여러 개 있다.

오랜 시간이 걸리는 작업(self-play 등)을 여러 개의 코어가
협업하도록 만들어 작업 효율을 높일 수 있다.

파이썬에서는 multiprocessing 패키지를 이용하면 된다.

mp.set_start_method

mp.set_start_method('spawn')

자식 프로세스를 만들 때, 새로운 파이썬 인터프리터를 실행시킨다.

윈도우, macOS에서의 기본값이므로, 리눅스에서 이걸 호출하면
운영체제와 상관없이 동일한 동작이 보장된다.

mp.Queue


```
import time
def worker(queue):
  # 10초 대기후 queue에 1을 넣는다.
  time.sleep(10)
  queue.put(1)

manager = mp.Manager()
queue = manager.Queue()
p = mp.Process(target=worker, args=(queue,))
p.daemon = True
p.start()

# put이 10초 뒤에 호출되므로, 이 부분 역시 10초 뒤에 호출된다.
print(queue.get())
```

감사합니다

http://github.com/utilForever

질문 환영합니다!