Connect6(육목) Agent Development Project

AlphaZero 기반의 Dual/Single-Head 강화학습

김현우(120250322), 조윤기(120250344)

Project Outline

AlphaZero 방법론 적용

범용적 게임 해결을 위해
고안된 AlphaZero
알고리즘(신경망 + MCTS)을
Connect6의 특수 규칙에
맞게 최적화하여 적용

Dual/Single-Head Architecture

신경망의 Policy Head의 설계 시
Dual-Head vs Single-Head 구조
비교 실험을 통해 최적의
Architecture 도출

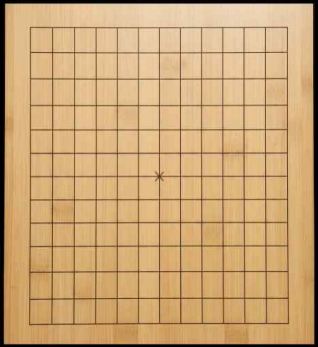
Self-Play 학습

인간의 기보 없이 오직 자가
대국(Self-Play)과 MCTS만으로
인간과의 경기에서 승리하는 것을
목표

Environment & Dataset

Environment

- **보드 크기:** 11x11 격자
- **승리 규칙:** 정확히 **6목** 달성 시 승리
  (7목 이상은 승리가 아니며 게임 계속 진행)
- **진행 방식:** 흑이 첫 1수 착수 후, 백과 흑이 번갈아 가며 **2수 씩** 착수
- **블로킹(Blocking):** 게임 시작 시 0/2/4개의 '착수 금지 구역'을 무작위로 생성



Dataset

- **자가 대국 (Self-Play):** Agent간 대국을 통해 정답 없이 학습 데이터 생성
- **데이터 증강 (Data Augmentation):** 바둑판의 기하학적 대칭성 활용
  - **회전(0°, 90°, 180°, 270°) 및 좌우 반전을 조합**
  - **1개의 대국 데이터를 8배로 증가** 시켜 데이터 효율성 극대화 및 overfitting 방지

State, Action, Reward

State

6개의 채널의 13 X 13 텐서로 구성

- Ch 1: 흑돌 위치
  흑돌 위치 = 1, 빈칸/백돌 = 0
- Ch 2: 백돌 위치
  백돌 위치 = 1, 빈칸/흑돌 = 0
- Ch 3: 블로킹 위치
  블로킹 자리 = 1, 나머지 = 0
- Ch 4: 현재 플레이어
  흑 차례 = 모두 1, 백 차례 = 모두 0
- Ch 5: 서브스텝
  첫 수 = 모두 0, 둘째 수 = 모두 1
- Ch 6: 마지막 수 위치
  마지막 수 위치 = 1, 나머지 = 0

Action

보드의 빈 위치에 돌을 놓는 것

- 11x11 격자를 Flatten한 169개 위치 중 하나를 선택(Discrete Space)
- 이미 돌이 있거나 블로킹된 곳은 Masking 처리하여 선택 불가

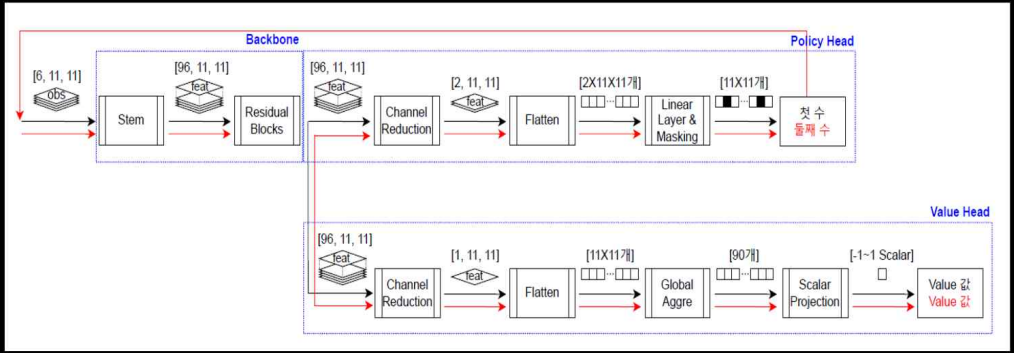
Reward

- 승리: +1, 패배: -1, 무승부: 0

Model Design

Neural Network Architecture

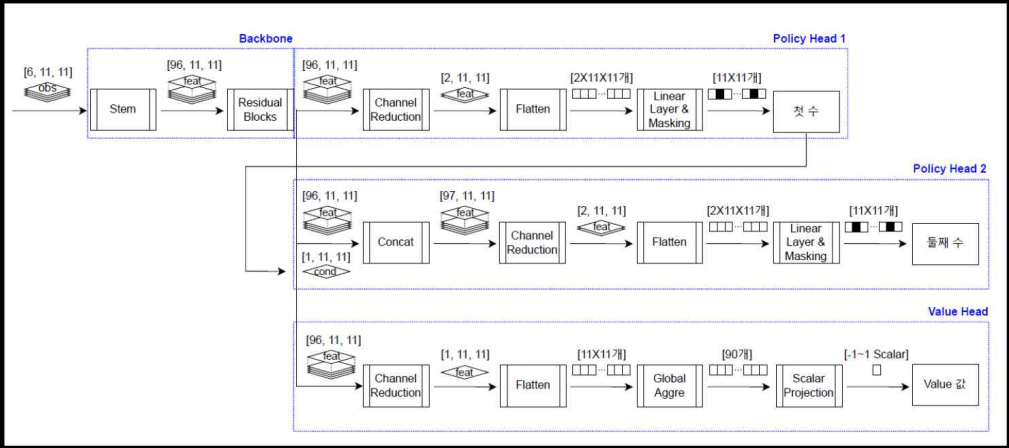
- Single-Head



Model Design

Neural Network Architecture

- Double-Head



Model Design Neural Network Architecture

1. Backbone (ResNet Encoder): 보드의 상태를 압축하여 Feature Map을 추출

가. 입력: (6, 11, 11) = (채널, W, H) = "obs"

나. Pipeline

1) Stem: Conv 3x3 (96 filters) → BatchNorm → ReLU / 출력: (96, 11, 11)

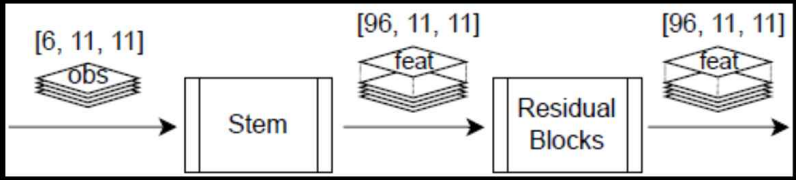
기대효과: 채널 증가를 통한 다양한 형세 판단 정보 분석 가능

2) Residual Blocks (12회 반복): Conv 3x3 (96) → BatchNorm → ReLU → Conv 3x3 (96) → BN → Skip Connection → ReLU / 출력: (96, 11, 11)

기대효과: 신경망의 효과와 Skip Connection으로 정보손실이 적은 Feature Map 추출 가능능

3) Stem + Residual Blocks = ResNet Encoder

다. 출력: (96, 11, 11) = (96개의 Feature map) = "feat"



Model Design Neural Network Architecture

2. Policy Head 1 (π1: 첫 번째 수) : Backbone의 출력을 받아 첫 번째 수 착수 위치 계산

* Single-Head에서는 Policy Head 1의 첫 번째 수 출력 값이 backbone의 입력값으로 다시 들어가 한번 더 통과(Policy Head 1을 두번 통과하여 두 수를 출력

가. 입력: Backbone 출력 = (96, 11, 11)

나. Pipeline

1) Channel Reduction: Conv 1x1 (2 filters) → BatchNorm → ReLU / 출력: (2, 11, 11)

기대효과: 채널 압축을 통하여 핵심 정보만 남기고 computation을 줄이고 overfitting 방지

2) Flatten: 출력값 flatten / 출력: 2 X 11 X 11 = 242개 노드

기대효과: 121개의 logit 계산을 위한 중간 단계

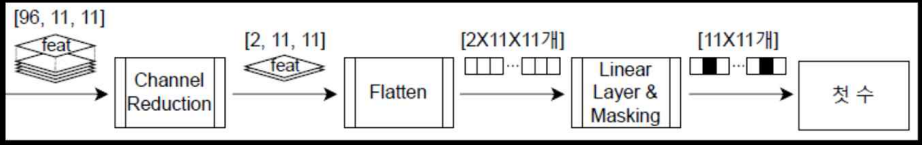
3) Linear Layer: FC Layer / 출력: 121개 노드

기대효과: 121개의 logit 계산을 통한 각 위치의 점수 산출

4) Masking: 이미 돌이 있거나 blocking 자리는 "-∞"로 변환(softmax시 0%) / 출력: 121개 노드

기대효과: 돌을 착수 할 수 없는 위치 구현

다. 출력: (121,) = 11x11 각 위치별 Logit 값(추천점수)



Model Design Neural Network Architecture

3. Policy Head 2 (π2: 두 번째 수) : Backbone출력에 첫 번째 수의 위치 정보(Condition)를 합쳐서 두 번째 착수 위치 계산

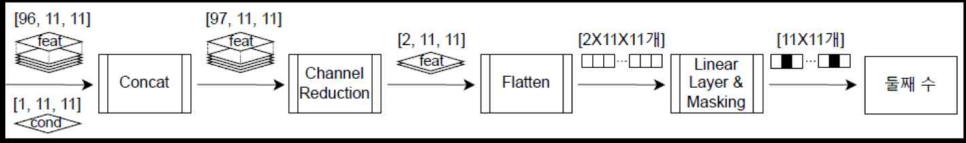
* Single-Head에서는 사용하지 않음.

가. 입력: Backbone 출력 (96, 11, 11) + Condition (1, 11, 11)

나. Pipeline

1) Concatenation: 96 + 1 = 97 채널 / 출력: (97, 11, 11)
   기대효과: "다양한 형세 판단"을 위한 채널(96)과 "직전 착수 위치"를 표시한 채널(1)을 한 데이터로 구현
2) Channel Reduction: Conv 1x1 (2 filters) → BatchNorm → ReLU / 출력: (2, 11, 11)
   기대효과: 채널 압축을 통하여 핵심 정보만 남기고 computation을 줄이고 overfitting 방지
3) Flatten: 출력값 flatten / 출력: 2 X 11 X 11 = 242개 노드
   기대효과: 121개의 logit 계산을 위한 중간 단계
4) Linear Layer: FC Layer / 출력: 121개 노드
   기대효과: 121개의 logit 계산을 통한 각 위치의 점수 산출
5) Masking: 이미 돌이 있거나 blocking 자리는 "-∞"로 변환(softmax시 0%) / 출력: 121개 노드
   기대효과: 돌을 착수 할 수 없는 위치 구현

다. 출력: (121,) = 11x11 각 위치별 Logit 값(추천점수)



Model Design Neural Network Architecture

4. Value Head (v: 승률 예측) : 현재 국면이 나에게 얼마나 유리한지를 판단

가. 입력: Backbone 출력 = (96, 11, 11)

나. Pipeline

1) Channel Reduction: Conv 1x1 (1 filters) → BatchNorm → ReLU / 출력: (1, 11, 11)

기대효과: 채널 압축을 통하여 핵심 정보만 남기고 computation을 줄이고 overfitting 방지

2) Flatten: 출력값 flatten / 출력: 1 X 11 X 11 = 121개 노드

기대효과: 121개의 logit 계산을 위한 중간 단계

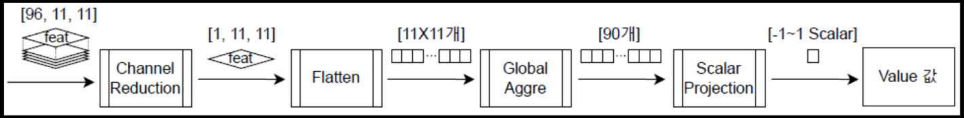
3) Global Aggregation: FC Layer(90 노드) → ReLU / 출력: 90개 노드

기대효과: 급격한 압축으로 인한 정보손실을 방지하고 90개의 위치의 정보에 대한 전역적인 판단(고차원 추론)

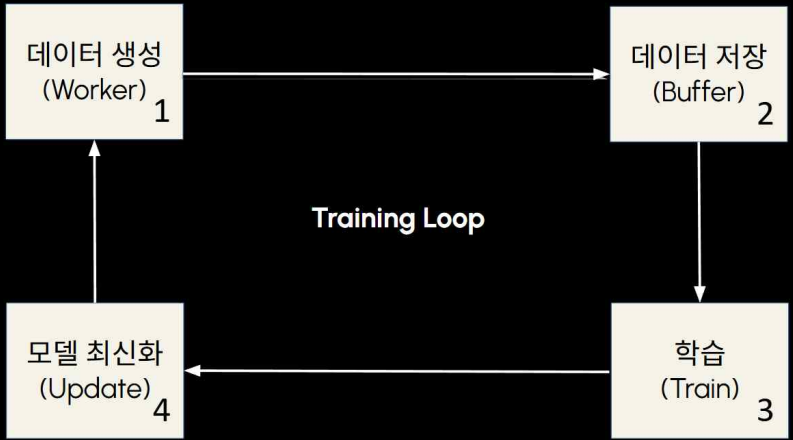
4) Scalar Projection: FC Layer(1 노드) → Tanh / 출력: -1~1 범위의 스칼라 값

기대효과: 90개의 판단근거를 종합하여 하나의 logit으로 만들고 Activation Function(Tanh)을 통해 점수(-1 ~ 1)화

다. 출력: 스칼라 값 = 범위 [-1, 1] 사이의 스칼라 값



Training Pipeline



Training Pipeline

데이터 생성
(Worker)
1

1. 목표: Agent가 Self-Play를 통하여 충분한 데이터 생성
2. 동작원리

가. Simulation: 현재 보드 상태(obs)에서 200회의 MCTS simulation을 수행하며 매 simulation마다 신경망(Model)을 호출하여 미래의 형세를 평가

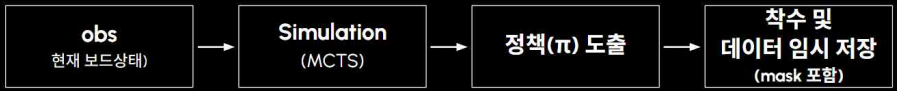
* 착수 위치(노드)마다 N(방문횟수)과 W(가치, v값의 누적 계산)를 업데이트하며, 다음 노드 선택시에는 PUCT 공식을 통하여 선정

나. 정책 도출: 시뮬레이션이 끝나면, 각 착수 지점의 방문 횟수(N)를 확률 분포(π)로 변환 하며 이것이 해당 국면의 '정답지' 역할

다. 착수 및 임시 저장: 확률에 따라 실제 돌을 착수하고, (obs, π, mask) 쌍을 메모리에 임시 저장

* 아직 승패에 대한 보상(z)는 모르는 상태이기 때문에 임시 저장 후 승패 결정에 따른 보상(z)를 추가하여 최종 저장장

라. 흐름도



Training Pipeline

1. 목표: 게임 종료에 따른 데이터 저장 및 증강
2. 동작원리

가. Backfilling: 게임이 종료 후 승패가 결정되면, 그동안 임시 저장했던 모든 수에 대해 "누가 둔 수인가?"를 따져 z값으로 반영(승: +1, 패: -1, 무승부: 0)

나. Augmentation: 바둑판의 대칭성을 활용하여, 1개의 원본 데이터를 회전(0, 90, 180, 270도) 및 좌우 반전을 통해 8개의 데이터로 증강(데이터 효용 및 일반화 성능 향상)

다. buffer 저장: 데이터를 Replay Buffer에(최대 500,000개)에 저장하며 용량이 가득 차면 가장 오래된 데이터부터 삭제

Data 저장 형태: {obs, : (6,11,11) = 보드 상태 관측
π, : (121,) = MCTS 방문 빈도 분포 [0, 1]
z, : 스칼라(+1, 0, -1) = 승, 무, 패
mask : (121,) = 각 위치에 돌을 놓을 수 있는지 여부 [True/False]
}

라. 흐름도

Training Pipeline

학습 (Train) 3 | 모델 최신화 (Update) 4

1. 목표: Self-play 종료 후 Replay Buffer의 데이터를 Random Sampling하여 신경망 학습
2. 동작원리

가. Random Sampling: Replay Buffer에서 512개(Batch Size)의 데이터를 Random Sampling

* 최근에 축적된 데이터 뿐만 아니라 초기 데이터도 섞여 있어 overfitting 방지

나. Prediction: Sampling한 데이터를 신경망의 입력값으로 활용하여 예측 값(P, v) 출력

* P: 착수 위치 확률 분포, v: 승리 확률(-1 ~ 1)

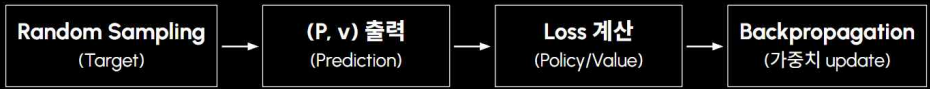
다. Loss 계산

1) Policy loss: π (target) - P (prediction) / Cross-Entropy loss

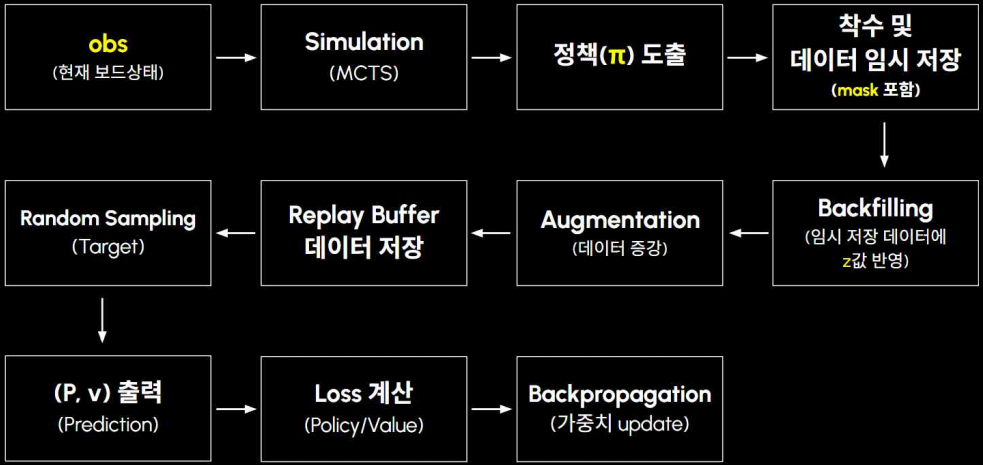
2) Value loss: z (target) - v (prediction) / MSE loss

라. Backpropagation: 총 Loss(Policy loss + Value loss)를 줄이는 방향으로 가중치 Update

마. 흐름도



Training Pipeline (전체 흐름도)



Main hyperparameter

Model Architecture

Hyperparameter	설명	값	비고
in_channels	입력 채널 수	6	관측 정보량
channels	Feature map 채널	96	형세파악
blocks	Residual block 개수	12	모델 표현력
board_size	바둑판 크기	11	13 x 13 size

Main hyperparameter

Train

Hyperparameter	설명	값	비고
learning rate	학습률	0.001 → 0.0001	수렴 속도, 안정성
batch size	배치 크기	512	학습안정성, 메모리
optimizer	최적화 알고리즘	Adam W	수렴 특성
num epochs	에폭 수	200	학습량 조절

Main hyperparameter

MCTS

Hyperparameter	설명	값	비고
num_simulations	시뮬레이션 횟수	200	탐색 품질, 계산량
c_puct	탐색 상수 (UCB)	1.5	Exploration vs Exploitation
temperature	행동 선택 정도	1.0 → 0.1	초반 exploration, 후반 exploitation
dirichlet alpha	루트 노이즈	0.3	탐색 다양성
dirichlet eps	노이즈 비율	0.25	탐색 vs 정책

Main hyperparameter

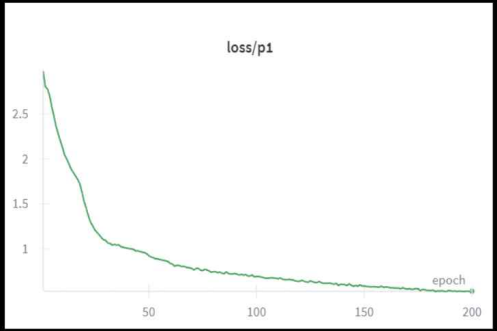
- Self-Play

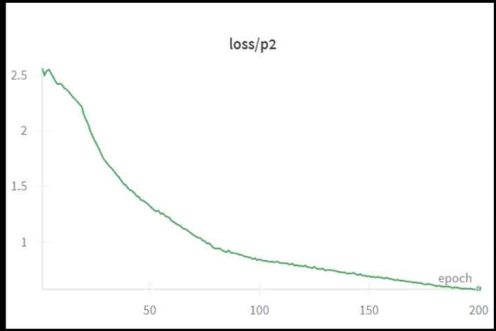
Hyperparameter	설명	값	비고
games_per_iter	반복당 게임 수	60	데이터 수집량
selfplay_workers	병렬 워커 수	12	Self-play 속도
buffer_cap	리플레이 버퍼 크기	500,000	메모리, 데이터 다양성
min buffer	학습시작 최소 데이터 수	4,000	적정학습 시작 최소 샘플

Experiment Environment & Evaluation Metric

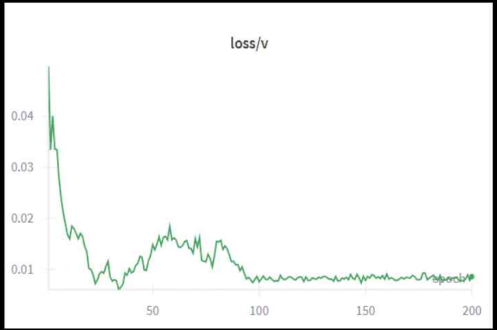
- **Experiment Environment**
  1. **Framework: Python 3.8+, PyTorch**
  2. **Computing Resources**
     - 가. NVIDIA RTX 3090 2대: 신경망 추론 및 학습
     - 나. CPU: 다중 코어를 활용한 16개의 Self-Play Workers 병렬 구동
- **Evaluation Metric**
  1. **Policy/Value Loss**
     - 가. 학습 데이터에 대한 모델의 loss 감소 추이 확인
     - 나. Policy Loss: MCTS가 찾은 수(π)와 모델의 예측(P) 간의 Cross-Entropy Loss
     - 다. Value Loss: 실제 결과(z)와 예측(v) 간의 MSE Loss
  2. **Win rate**
     - 가. 일정주기(20 Epoch)마다 현재 모델과 이전 베스트 모델 간의 100판의 평가 대국을 수행
     - 나. 새로운 모델의 승률이 55% 이상일 경우, 새로운 베스트 모델로 선택 후 가중치 저장
  3. **Explained Variance(EV)**
     - 가. Value Head가 예측한 승률(v)이 실제 게임 결과(z)를 얼마나 잘 예측하였는지를 나타내는 지표
     - 나. $EV = 1 - Var(z - v)/Var(z) \rightarrow 1.0$에 가까울 수록 승패를 잘 예측

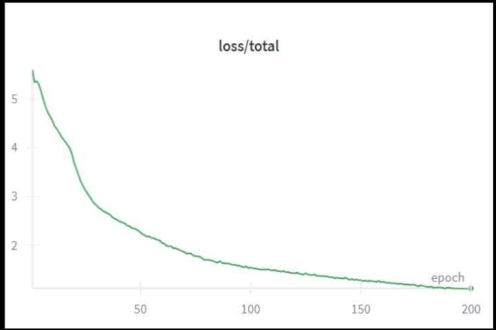
Experiment Results (Dual-Head)



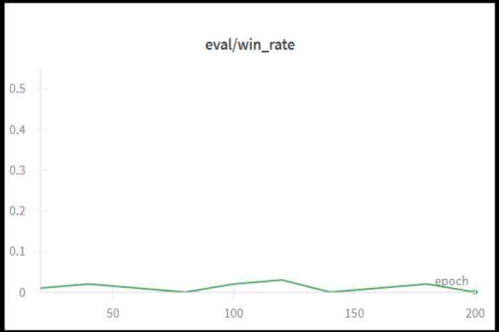


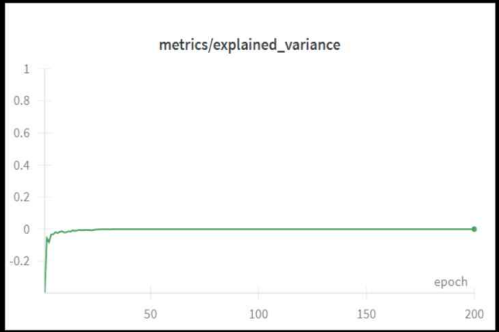
Experiment Results (Dual-Head)





Experiment Results (Dual-Head)





Experiment Results (Dual-Head)

Loss

- Epoch 0 → Epoch 200에 따른 Loss 변화
  - p1 loss: 2.8 → 0.53
  - p2 loss: 2.5 → 0.58
  - v_loss: 0.05 → 0.009

==========

- 해석
  - 모델이 MCTS target 분포를 성공적으로 학습
  - P loss는 수렴이 안정적 이지만,
  - v loss는 불안정 및 극단적 수렴(overfitting)
  - 학습 메커니즘 자체는 정상 작동

Win Rate

- Epoch 0 → Epoch 200에 따른 Win Rate 변화

Epoch:	20	60	100	140	180	200
Win Rate:	0	0	0	0	0	0

==========

- 해석
  - 새 모델이 이전 모델을 단 한 번도 이기지 못함
  - 200 epoch 동안 성능 향상이 없음
  - Loss 감소에도 불구하고 실제 플레이 능력은 정체

Explained Variance

- Epoch 0 → Epoch 200에 따른 EV 변화

Epoch:	0	50	100	150	200
EV:	-0.2	0.01	0.02	0.0	0.0

1.0에 가까울 수록 승/패 예측 성능 좋음

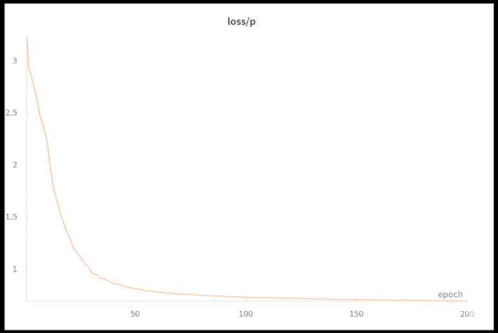
==========

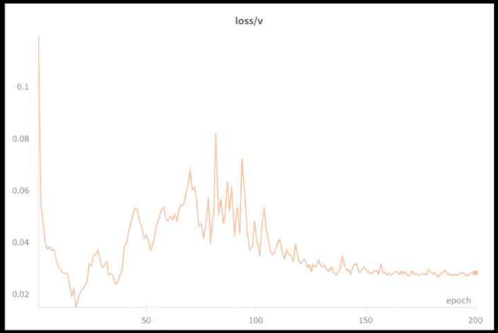
- 해석
  - Value head가 모든 상황에서 동일한 값(≈0.0)
  - 게임 상황에 대한 이해력 부족
  - MCTS가 value 정보를 활용하지 못함

종합분석

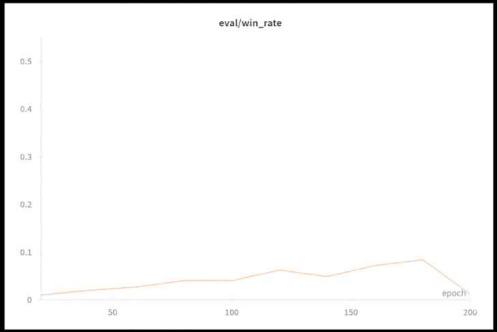
- "Value Head 성능 저하 → MCTS 탐색 품질 저하 → 저품질의 Policy Target 생성 및 학습 → 성능 개선 없음"의 악순환 구조
- 실제 게임 결과가 평균적으로 0에 가깝기 때문에 MSE loss는 낮지만 실제로는 의미없는 예측을 하고 있음.
- 모델 구조상 `p2`는 첫 수 정보를 `concat`으로 받지만 실제로는 이를 무시하고 평균적인 두 번째 수만 학습한다고 판단
  즉, 첫 수에 따라 두 번째 수가 적절하게 변하지 않아 연관성 학습 실패
- Single-Head 구조로의 전환 필요

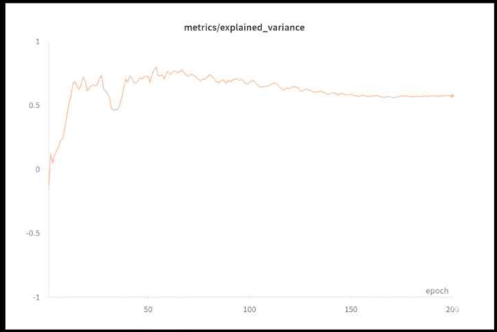
Experiment Results (Single-Head)





Experiment Results (Single-Head)





Experiment Results (Single-Head)

Loss

Epoch 0 → Epoch 200에 따른 Loss 변화
- p loss: 3.3 → 0.51
- v loss: 0.12 → 0.03

==========

해석
- 학습 초중반 v loss가 불안정하지만 후반으로 갈 수록 안정 되며 dual-head에 비해 극단적 값(0.009)으로 수렴되지 않음.
- 학습 메커니즘 정상 작동

Win Rate

Epoch 0 → Epoch 200에 따른 Win Rate 변화

Epoch:	20	60	100	140	180	200
Win Rate:	0.01	0.03	0.04	0.05	0.08	0.01

==========

해석
- Dual-Head에 비해 적은 수치지만 개선
- 하지만 0.55이상 승률이 나와야 유의미하다고 볼 수 있기 때문에 성능이 개선되고 있다고 볼 수 없음.

Explained Variance

Epoch 0 → Epoch 200에 따른 EV 변화

Epoch:	0	50	100	150	200
EV:	-0.2	0.7	0.65	0.6	0.6

1.0에 가까울 수록 승/패 예측 성능 좋음

==========

해석
- 수치는 개선되었지만 self-play draw 비율이 개선되지 못하였음(약 90%)
- 즉, 승패를 가리지 못하고 무의미한 수만 두다가 끝나버리는 무승부를 맞추는 것을 잘 맞추고 있다는 착각하고 있을 것이라고 판단.

종합분석

- 평가지표가 Dual-Head에 비해 개선된 것은 사실이지만 유의미한 성능의 도약이라고는 볼 수 없음.
- 다양한 파라미터를 조절하여 실험해 본 결과 MCTS simulation 횟수가 성능에 큰 영향을 끼치는 것을 알 수 있었음.
- 프로젝트에서는 정해진 기간으로 인하여 simulation 횟수를 200회(1 epoch 당 약 25분 소요) 까지 밖에 늘리지 못하였지만, 600~800회까지 늘려 다양한 수를 학습하게 한다면 충분히 좋은 성능이 나올 것이라고 판단됨.
- 프로젝트 종료 후에도 simulation 횟수를 높여 실험을 이어나갈 예정

Results & Discussion

Model Architecture 비교 분석

육목(Connect6) Agent 개발의 핵심은 "한 턴에 두 수를 착수한다"는 규칙을 신경망에 어떻게 모델링할 것인가였으며,
본 프로젝트에서는 이를 해결하기 위해 두 가지 접근 방식을 아래와 같이 비교 실험하였음.

1. 가정

가. 실험 설계 단계에서 첫 번째 수와 두 번째 수 사이의 전략적 연계성이 중요할 것이라 판단

나. 첫 번째 수를 조건(Condition)으로 받아 두 번째 수를 결정하는 Dual-Head 구조가 더 우수할 것으로 가정

2. 실험 결과

가. 실제 학습 결과, Single-Head 모델이 평가지표 면에서 Dual-Head 모델보다 다소 우수한 성능을 보임.

나. 실제 GUI 구현을 통한 대국(정성적 평가)에서도 Single-Head 모델의 성능이 우수

3. 원인 분석

가. 단순 결합(Concatenation)의 한계

1) Dual-Head 모델은 Backbone의 Feature Map에 첫 수의 위치 정보(One-hot)를 단순 결합(Concat)하여 두번 째 수를 추론하도록 설계되었음.

2) 그러나 실험 결과, 이 방식만으로는 두 수 간의 복잡한 공간적·전략적 상관관계를 해석해내지 못하였음.

나. 오차 전파 (Error Propagation)

1) 학습 초기 첫 번째 수의 질이 좋지 않을 때, 이 값이 그대로 두 번째 수를 위한 입력으로 들어감.

2) 이러한 입력값이 두 번째 헤드의 학습까지 저해하는 악순환이 발생

4. 결론

가. 반면, Single-Head 모델은 상태 전이를 서브스텝(Substep) 단위로 쪼개어 입력값으로 활용

나. 이는 "변화된 상태에서의 최적해"를 독립적으로 찾게 함으로써 학습 안정성 확보를 가능하게 하여 더 우수한 성능

Results & Discussion

핵심 Hyperparameter의 영향

Model Architecture, Train, MCTS, Self-play의 각종 Hyperparameter는 AlphaZero style에 적합하다고 검증된 값을 준용하고자 하였음. 하지만 AlphaZero 알고리즘은 MCTS 기반이기 때문에 상당한 연산량이 수반됨. 제한된 프로젝트 기간내 정해진 GPU로 인하여 "성능-추론속도" 간의 Trade-off 관계에 영향을 끼치는 Hyperparameter의 선별 및 조정이 필요하였음.

1. MCTS simulation 횟수

가. "성능-추론속도"의 Trade-off 관계에 가장 직접적인 영향을 끼치는 Hyperparameter

나. 너무 낮으면, 속도는 빠르지만 MCTS가 신경망의 잘못된 직관(P)을 충분히 보정하지 못해 저품질의 정책(π)을 생성

다. 너무 높으면, 성능 향상 폭 대비 연산 비용이 과다하여 학습속도 제한

라. 반복적인 실험을 통하여 200회가 최적이라고 판단.(하지만 성능을 위해서는 600~800회 까지 증가 필요)

2. Data Augmentation 개수

가. Self-play 데이터 생성 비용이 매우 높은 환경이므로, 대칭성을 활용한 8배 데이터 증강은 제한된 시간내 고품질의 데이터를 확보하기 위한 방안으로 도입.

나. 학습 초반 저품질의 데이터 또한 증강 되어 학습의 불안정이 초래될 것으로 예상되었지만 이는 MCTS를 통한 수많은 데이터 축적으로 극복되었음.

다. 즉, 데이터 증가는 모델의 Overfitting을 방지하고 돌의 형세를 학습하게 하는데 큰 역할을 했다고 판단하였음.

Results & Discussion

휴리스틱 도입의 효과와 시사점

학습 초기에 승리에 도움이 되지 않는 위치에 착수하는 경우가 대부분임을 확인하였으며, 초기 성능의 신속한 향상을 위하여 어느정도의 휴리스틱 도입이 도움이 될 것이라고 가정하였음.

1. Proximity Prior

가. 목적: 마지막 착수 주변에 더 높은 가중치를 부여하여 형세가 응집 될 수 있도록 착수를 유도

나. 문제점: 상대의 마지막 착수주변에 돌을 두는 경향이 수비 편향 문제로 이어져 성능 저하

2. Threat Detection

가. "연속된 돌이 4개 나오면 양쪽을 막는다.", "연속된 돌이 3개 나오면 최소한 한쪽은 막는다." 등 상대방의 즉시 승리로 이어지는 수를 막기위한 "착수 규칙" 구현

나. 문제점: 다양한 전략을 학습하기 보다는 "착수 규칙"을 구현하기 위한 착수에 집중하여 창의적 수 제한 판단

결과적으로는 "어설픈 인간의 지식을 투입하지 않는다."라는 AlphaZero의 지향점을 준용하고자 도입한 휴리스틱을 비활성화 하였음. 학습 초기의 체크포인트를 통하여 GUI를 구현해 정성적 평가를 해본 결과, 모델 초기부터 돌이 잘 응집되어 경기가 이어지는 것 처럼 보였음. 하지만 학습이 중후반부로 이어질 수록 휴리스틱이 오히려 장애물이 되어 창의적인 수가 제한됨을 확인.