KAIST Include 동아리 스터디

# AlphaGo와 AlphaGo Zero를 만들며 익히는 딥러닝 및 강화학습

Chris Ohk
utilForever@gmail.com

# Python으로 바둑 나타내기



- 바둑 규칙을 구현하는 Python 라이브러리를 만들려면…
  - 사람을 상대로 바둑을 둔 기보를 추적할 수 있어야 한다.
  - 두 바둑봇 간의 대국 기보를 추적할 수 있어야 한다.
  - 동일한 보드 위치에서 벌어질 수를 따질 수 있어야 한다.
  - 기존 경기 기록을 불러와서 훈련 데이터로 만들 수 있어야 한다.

# Python으로 바둑 나타내기



- 플레이어를 열거체로 표현 (gotypes.py)
  - 바둑은 흑돌과 백돌이 번갈아가며 수를 두며 경기를 진행
  - 플레이어(Player) 역시 흑(black)과 백(white)로 나타낼 수 있다.

```python
import enum

class Player(enum.Enum):
  black = 1
  white = 2

  @property
  def other(self):
    return Player.black if self == Player.white else Player.white
```

# Python으로 바둑 나타내기



- 바둑판의 점을 튜플로 나타내기 (gotypes.py)
  - 라이브러리 namedtuple을 사용하면 point[0]과 point[1] 대신 point.row와 point.col로 좌표에 접근할 수 있다.

```python
from collections import namedtuple

class Point(namedtuple('Point', 'row col')):
  def neighbors(self):
    return [
      Point(self.row - 1, self.col),
      Point(self.row + 1, self.col),
      Point(self.row, self.col - 1),
      Point(self.row, self.col + 1),
    ]
```

# Python으로 바둑 나타내기



- 수 설정: 돌 놓기, 차례 넘기기, 대국 포기 (goboard_slow.py)
  - 내 차례에 할 수 있는 행동들
    - 바둑판 위에 돌을 놓는다. → Move.play()
    - 차례를 넘긴다. → Move.pass_turn()
    - 대국을 포기한다. → Move.resign()
  - Player, Point, Move 클래스는 데이터 타입
    - 바둑판을 나타내기 위한 필수 클래스일 뿐 어떤 경기 규칙도 들어 있지 않다.
  - 앞으로 작업할 내용
    - Board 클래스를 구현해 돌을 놓고 따내는 규칙을 다룬다.
    - GameState 클래스를 구현해 바둑판의 모든 돌에 대해 누구 차례고 이전 상태는 어땠는지 파악한다.

# Python으로 바둑 나타내기



- 수 설정: 돌 놓기, 차례 넘기기, 대국 포기 (goboard_slow.py)

```python
import copy
from dlgo.gotypes import Player

class Move():
  def __init__(self, point=None, is_pass=False, is_resign=False):
    assert (point is not None) ^ is_pass ^ is_resign
    self.point = point
    self.is_play = (self.point is not None)
    self.is_pass = is_pass
    self.is_resign = is_resign

  @classmethod
  def play(cls, point):
    return Move(point=point)

  @classmethod
  def pass_turn(cls):
    return Move(is_pass=True)

  @classmethod
  def resign(cls):
    return Move(is_resign=True)
```

# 바둑판 구현하기

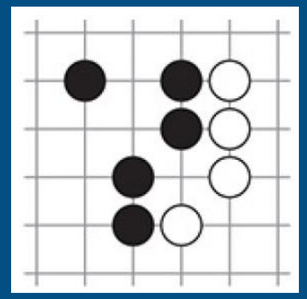


- 바둑판에서 돌을 따낼 것인지 확인하는 알고리즘을 생각해 보자.
  1. 활로가 남아 있는 이웃이 있는지 살펴봐야 한다.
  2. 이웃의 이웃 중에도 활로가 남아 있는 경우가 있는지 확인해야 한다.
  3. 이웃의 이웃의 이웃도 조사해야 하고, 그 이상도 살펴봐야 한다.
- 돌을 하나하나 살펴보는 방식은 계산 복잡도가 크다. 어떻게 해야 할까?
  빠르게 확인하려면 **연결된 돌들을 하나의 단위로 묶어서 생각**하면 된다.

# 바둑에서 연결 추적하기 : 이음



- 이음을 set으로 인코딩 (goboard_slow.py)
  - 같은 색 돌의 연결된 그룹을 이음수 또는 이음이라고 한다.
  - num_liberties()를 호출해 활로 수를 알 수 있다.
  - add_liberty()와 remove_liberty()를 사용해 이음의 활로를 추가하고 제거한다.
  - 선수가 돌을 놓아 두 개의 그룹을 연결한 경우 merged_with()를 호출한다.



# 바둑에서 연결 추적하기 : 이음



- 이음을 set으로 인코딩 (goboard_slow.py)

```python
class GoString():
  def __init__(self, color, stones, liberties):
    self.color = color
    self.stones = set(stones)
    self.liberties = set(liberties)

  def remove_liberty(self, point):
    self.liberties.remove(point)

  def.add_liberty(self, point):
    self.liberties.add(point)

  def merged_with(self, go_string):
      assert go_string.color == self.color
      combined_stones = self.stones | go_string.stones
        return GoString(
          self.color, combined_stones,
          (self.liberties | go_string.liberties) - combined_stones)

  @property
  def num_liberties(self):
    return len(self.liberties)

  def __eq__(self, other):
    return isinstance(other, GoString) and \
      self.color == other.color and \
      self.stones == other.stones and \
      self.liberties = other.liberties
```

# 바둑판에 돌 놓기와 따내기



- 돌을 놓는 알고리즘
  - 같은 색의 이음을 연결한다.
  - 상대방 색 돌의 근접한 이음의 활로 수를 낮춘다.
  - 상대방 색 돌의 이음의 활로가 0이라면 이를 제거한다.
- 또한 새로 만들어진 이음의 활로가 0이면 이 수를 둘 수 없게 해야 한다.

# 바둑판에 돌 놓기와 따내기



- 바둑판 Board 인스턴스 생성 (goboard_slow.py)

```
class Board():
  def __init__(self, num_rows, num_cols):
    self.num_rows = num_rows
    self.num_cols = num_cols
    self._grid = {}
```

# 바둑판에 돌 놓기와 따내기



- 활로 파악용 이웃 점 확인 (goboard_slow.py)

```
def place_stone(self, player, point):
  assert self.is_on_grid(point)
  assert self._grid.get(point) is None
  adjacent_same_color = []
  adjacent_opposite_color = []
  liberties = []
  for neighbor in point.neighbors():
    if not self.is_on_grid(neighbor):
      continue
    neighbor_string = self._grid.get(neighbor)
    if neighbor_string is None:
      liberties.append(neighbor)
    elif neighbor_string.color == player:
      if neighbor_string not in adjacent_same_color:
        adjacent_same_color.append(neighbor_string)
    else:
      if neighbor_string not in adjacent_opposite_color:
        adjacent_opposite_color.append(neighbor_string)
  new_string = GoString(player, [point], liberties)
```

# 바둑판에 돌 놓기와 따내기



- 돌 놓기와 따내기 유틸리티 메소드 (goboard_slow.py)

```
def is_on_grid(self, point):
  return 1 <= point.row <= self.num_rows and \
    1 <= point.col <= self.num_cols

def get(self, point):
  string = self._grid.get(point)
  if string is None:
    return None
  return string.color

def get_go_string(self, point):
  string = self._grid.get(point)
  if string is None:
    return None
  return string
```

# 바둑판에 돌 놓기와 따내기



- place_stone() 정의 (goboard_slow.py)

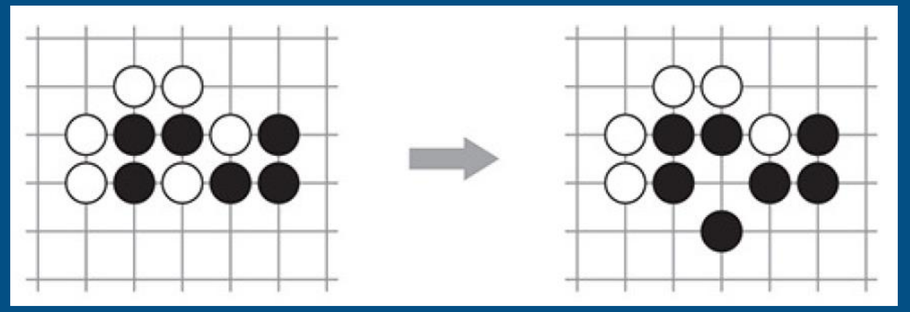
```
    for same_color_string in adjacent_same_color:
      new_string = new_string.merged_with(same_color_string)
    for new_string_point in new_string.stones:
      self._grid[new_string_point] = new_string
    for other_color_string in adjacent_opposite_color:
      other_color_string.remove_liberty(point)
    for other_color_string in adjacent_opposite_color:
        if other_color_string.num_liberties == 0:
          self._remove_string(other_color_string)
```

# 바둑판에 돌 놓기와 따내기



- place_stone() 정의 (goboard_slow.py)

```
def _remove_string(self, string):
  for point in string.stones:
    for neighbor in point.neighbors():
      neighbor_string = self._grid.get(neighbor)
      if neighbor_string is None:
        continue
      if neighbor_string is not string:
        neighbor_string.add_liberty(point)
  self._grid[point] = None
```



# 대국 현황 기록과 반칙수 확인



- 지금까지 Board 클래스에 돌을 놓고 따내는 규칙을 구현했다.
- 이제 GameState 클래스에서 대국 현황을 파악해
  수를 추가하는 기능을 구현해 보자.
  - 대국 현황은 현재 판 상태, 다음 선수, 이전 상태, 직전 수 등을 포함한다.

# 대국 현황 기록과 반칙수 확인



- 바둑 게임 현황 인코딩 (goboard_slow.py)

```python
class GameState():
  def __init__(self, board, next_player, previous, move):
    self.board = board
    self.next_player = next_player
    self.previous_state = previous
    self.last_move = move

  def apply_move(self, move):
    if move.is_play:
      next_board = copy.deepcopy(self.board)
      next_board.place_stone(self.next_player, move.point)
    else:
      next_board = self.board
    return GameState(next_board, self.next_player.other, self, move)

  @classmethod
  def new_game(cls, board_size):
    if isinstance(board_size, int):
      board_size = (board_size, board_size)
    board = Board(*board_size)
    return GameState(board, Player.black, None, None)
```

# 대국 현황 기록과 반칙수 확인



- 대국 종료 판단 (goboard_slow.py)

```python
def is_over(self):
  if self.last_move is None:
    return False
  if self.last_move.is_resign:
    return True
  second_last_move = self.previous_state.last_move
  if second_last_move is None:
    return False
  return self.last_move.is_pass and second_last_move.is_pass
```

# 대국 현황 기록과 반칙수 확인

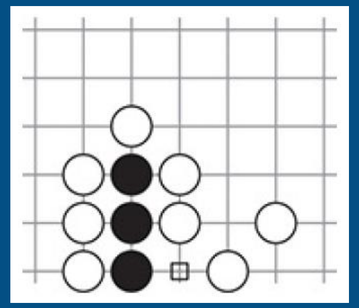


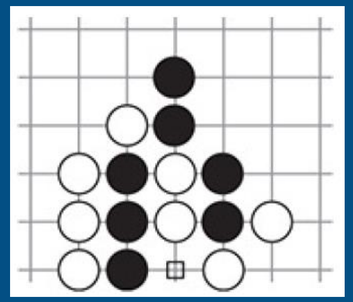
- 이제 apply_move()를 사용해 현재 경기 상태에 수를 추가하는 기능을 구현한다. 이때 이 수가 제대로 된 수인지 확인하는 코드도 작성해야 한다.
  - 두고자 하는 점이 비었는지 확인
  - 자충수가 아닌지 확인
  - 이 수가 바둑 규칙에 위반되는 것은 아닌지 확인

# 자충수



- 이음에 활로가 하나밖에 남지 않았을 때 그 활로에 돌을 놓는 수를 **자충수(self-capture)**라고 한다.





# 자충수



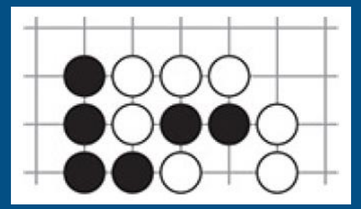
- 자충수 규칙을 적용한 GameState 정의 (goboard_slow.py)

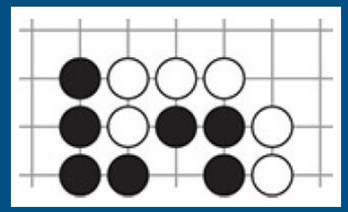
```python
def is_move_self_capture(self, player, move):
  if not move.is_play:
    return False
  next_board = copy.deepcopy(self.board)
  next_board.place_stone(player, move.point)
  new_string = next_board.get_go_string(move.point)
  return new_string.num_liberties == 0
```

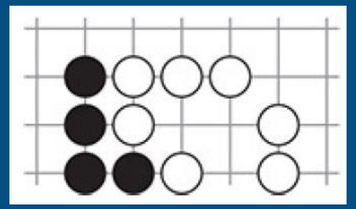
# 패



- 패는 어떤 수로 인해 판이 다시 이전 상태가 되었는지에 대한 것
  하지만 **동형반복(Situational Superko)** 상황이 아니라면
  **환격(Snapback)**과 같은 수로 인해 패가 적용되지 않을 때가 있다.







# 패



- 현재 게임 상태가 패 규칙을 위반하는가? (goboard_slow.py)

```python
@property
def situation(self):
  return (self.next_player, self.board)

def does_move_violate_ko(self, player, move):
  if not move.is_play:
    return False
  next_board = copy.deepcopy(self.board)
  next_board.place_stone(player, move.point)
  next_situation = (player.other, next_board)
  past_state = self.previous_state
  while past_state is not None:
    if past_state.situation == next_situation:
      return True
    past_state = past_state.previous_state
  return False
```

# 패



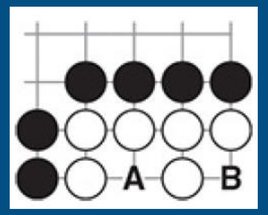
- 주어진 게임 상황에서 이 수는 유효한가? (goboard_slow.py)

```python
def is_valid_move(self, move):
  if self.is_over():
    return False
  if move.is_pass or move.is_resign:
    return True
  return (
    self.board.get(move.point) is None and
    not self.is_move_self_capture(self.next_player, move) and
    not self.does_move_violate_ko(self.next_player, move))
```

# 게임 종료



- 컴퓨터로 바둑을 두는 기본 개념은 자체 대국이다.
  - 처음에는 매우 약한 바둑봇으로 시작하지만 계속 바둑을 두면서 그 결과를 활용해 점차 강한 봇을 만들게 된다.
  - 이 기법의 장점을 활용하려면 자체 경기를 끝까지 치러야 한다. 하지만 대국은 보통 어느 한쪽이 더 두어봐야 승산이 없는 경우 끝난다. 만약 봇이 둘 수 있는 데까지 수를 두도록 놔두면 결국 활로를 막아버리고 돌을 다 뺏길 것이다.
  - 봇이 합리적인 방식으로 게임을 끝내는 데 도움이 되는 몇 가지 임시방편
    - 같은 색의 돌로 완전히 둘러싸인 곳에는 돌을 놓지 말라.
    - 활로가 하나뿐인 곳에는 돌을 놓지 말라.
    - 활로가 하나인 다른 돌은 일단 잡아라.



# 게임 종료



- 이 위치가 집일까? (helpers.py)
  - 게임 끝에 점수를 세는 방법은 바둑 대회와 바둑 협회마다 미묘하게 다르게 적용한다.
  - 여기서는 **중국식 셈법**이라고도 부르는 AGA의 **영역 셈법** 규칙을 따르도록 봇을 만든다.
  - 일반적인 대국에서는 **일본식 셈법**을 더 많이 사용하지만 AGA 규칙이 컴퓨터로 만들기에 조금 더 쉽고 규칙의 차이가 게임 결과에 미치는 영향은 거의 없다.

```python
from dlgo.gotypes import Point

def is_point_an_eye(board, point, color):
    if board.get(point) is not None:
        return False
    for neighbor in point.neighbors():
        if board.is_on_grid(neighbor):
            neighbor_color = board.get(neighbor)
            if neighbor_color != color:
                return False

    friendly_corners = 0
    off_board_corners = 0
    corners = [
        Point(point.row - 1, point.col - 1),
        Point(point.row - 1, point.col + 1),
        Point(point.row + 1, point.col - 1),
        Point(point.row + 1, point.col + 1),
    ]
    for corner in corners:
        if board.is_on_grid(corner):
            corner_color = board.get(corner)
            if corner_color == color:
                friendly_corner += 1
        else:
            off_board_corners += 1
    if off_board_corners > 0:
        return off_board_corners + friendly_corners == 4
    return friendly_corners >= 3
```

# 첫번째 봇 만들기



- 바둑판 구현과 게임 상태 코드화를 끝냈으니 이제 바둑을 두는 첫번째 봇을 만들 수 있다. 우선 모든 봇이 사용할 인터페이스를 정의한다.
- 바둑 에이전트의 핵심 인터페이스 (base.py)

```python
class Agent:
  def __init__(self):
    pass

  def select_move(self, game_state):
    raise NotImplementedError()
```

# 첫번째 봇 만들기



- 무작위 바둑봇 (naive.py)

```python
import random
from dlgo.agent.base import Agent
from dlgo.agent.helpers import is_point_an_eye
from dlgo.gobard_slow import Move
from dlgo.gotypes import Point

class RandomBot(Agent):
  def select_move(self, game_state):
    candidates = []
    for r in range(1, game_state.board.num_rows + 1):
      for c in range(1, game_state.board_num_cols + 1):
        candidate = Point(row=r, col=c)
        if game_state.is_valid_move(Move.play(candidate)) and \
          not is_point_an_eye(game_state.board, candidate, game_state.next_player):
          candidates.append(candidate)
    if not candidates:
      return Move.pass_turn()
    return Move.play(random.choice(candidates))
```

# 첫번째 봇 만들기



- 봇 대 봇 경기의 유틸리티 함수 (utils.py)

```python
from dlgo import gotypes

COLS = 'ABCDEFGHIJKLMNOPQRST'
STONE_TO_CHAR = {
  None: ' . ',
  gotypes.Player.black: ' x ',
  gotypes.Player.white: ' o ',
}

def print_move(player, move):
  if move.is_pass:
    move_str = 'passes'
  elif move.is_resign:
    move_str = 'resigns'
  else:
    move_str = '%s%d' % (COLS[move.point.col - 1], move.point.row)
  print('%s %s' % (player, move_str))

def print_board(board):
  for row in range(board.num_rows, 0, -1):
    bump = " " if row <= 9 else ""
    line = []
    for col in range(1, board.num_cols + 1):
      stone = board.get(gotypes.Point(row=row, col=col))
      line.append(STONE_TO_CHAR[stone])
    print('%s%d %s ' % (bump, row, ''.join(line)))
  print('    ' + '  '.join(COLS[:board.num_cols]))
```

# 첫번째 봇 만들기



- 봇이 자체 대국을 치르는 스크립트 (bot_v_bot.py)

```python
from dlgo import agent
from dlgo import goboard
from dlgo import gotypes
from dlgo.utils import print_board, print_move
import time

def main():
  board_size = 9
  game = goboard.GameState.new_game(board_size)
  bots = {
    gotypes.Player.black: agent.naive.RandomBot(),
    gotypes.Player.white: agent.naive.RandomBot(),
  }
  while not game.is_over():
    time.sleep(0.3)

    print(chr(27) + "[2J")
    print_board(game.board)
    bot_move = bots[game.next_player].select_move(game)
    print_move(game.next_player, bot_move)
    game = game.apply_move(bot_move)

if __name__ == '__main__':
  main()
```

# 과제 #1



- 대국 속도를 향상시키는 방법 : 조브리스트 해싱(Zobrist Hashing)
  - 조브리스트 해싱을 사용해 바둑판을 해시값으로 저장해 보자.
- 봇과 대국하기
  - 플레이어와 앞서 만든 RandomBot이 대결할 수 있는 스크립트를 만들어 보자.
- 과제 내용 업로드 : 4월 10일 (토) 14:00
- 과제 마감 : 4월 23일 (금) 24:00

감사합니다!

스터디 듣느라 고생 많았습니다.