卒業研究報告書

題目

コネクト４のゲーム AI 開発

指導教員

石水 隆　講師

報告者

18-1-037-0220

須貝　祥大

近畿大学理工学部情報学科

令和 4 年 1 月 24 日提出

概要

今日，人工知能の技術進歩が進む中で，ディープラーニングと呼ばれる手法が注目されている．ディープラーニングは囲碁や将棋，チェスなどのボードゲームでは，AI が着手を決定するために用いられており，ディープラーニングを用いたゲーム AI の中には，人間界のトッププロに匹敵する強さの物もある．つまり，囲碁や将棋などの選択肢がたくさんあるゲームにおいて，ディープラーニングが有効である．そこで本研究では，選択肢が少ないゲームにおいてディープラーニングが有効であるかどうかを検証する．

本研究では，コネクト 4 と呼ばれる重力付き四目並べというボードゲームを題材に python を用いてゲーム AI を開発する．今回は，モンテカルロ木探索を用いた自己対戦を行い，そのデータをもとにディープラーニングを行うプログラムを作成した．その後，作成したゲーム AI を既存の AI と対戦させることによって分かった，作成したゲーム AI の評価，考察，課題点について述べる．

目次

1. 序論

1.1 本研究の背景

近年，人工知能の技術が進歩していく中でディープラーニングと呼ばれる手法が注目されている．ディープラーニングとは人間がデータを編成して定義済みの数式にかけるのではなく，人間がデータに関する基本的なパラメータ設定のみを行い，その後は何層もの処理を用いたパターン認識を通じ，コンピュータに課題の解決方法を学習させるものである[14]．ディープラーニングは最近では囲碁や将棋などのボードゲームに使用されるものもあり，ディープラーニングを用いたゲーム AI の中にはプロに勝利する強さのものもある[7]．

1.2 コネクト４とは

コネクト４とは縦 6 マス，横 7 マスのボードと 2 色の石を用いた 2 人用のボードゲームである[10]．コネクト 4 のボードは垂直に立てられ，各プレイヤーはボードの上部から石を投入することができる．投入された石は，一番下のマスまたは，すでに石が置かれたマスの上のマスに置かれる．各プレイヤーは交互に石を投入していき，縦，横，斜めのいずれかの方向に自石を 4 つ並べることができたプレイヤーの勝利になる．コネクト 4 では双方最善手を打った場合，41 手で先手の勝ちになることが分かっている[3]．図 1 にコネクト 4 の最善手の手順を示す．コネクト 4 の既知の AI としてはα β法により着手選択する Connect 4 Solver がある[4]．また，フラッシュを用いたフリーゲームとして 4 目並べゲーム【Connect 4】が存在する[5]．また，コネクト 4 のバリエーションの一つとして筒型コネクト 4 がある．筒型コネクト 4 は，ゲーム盤が筒型をしており，左端の一列と右端の一列が隣接している．このため，筒型コネクト 4 では端を挟んで自石を 4 つ並べても勝利となる．筒型コネクト 4 については，横幅が 2 マスまたは 6 マスの場合だと先手は負ける事ができない，つまり先手がわざと負けようとしても負けられないことが示されている[9]．

33	34		15	38	39	30
32	28	41	5	37	21	22
29	23	40	4	36	13	19
26	20	31	3	35	12	18
25	17	27	2	24	11	14
16	9	10	1	7	8	6

図 1 コネクト 4 の最善手

1.3 本研究の目的

前述の通り，コネクト 4 には既知の AI として Connect 4 Solver が存在する．コネクト 4 はゲームの特性上，各手番の選択肢が最大でも 7 通りしかないため，$\alpha\ \beta$ 法でも短い時間で効率よく最適解を得ることができる．そこでコネクト 4 のように選択肢の少ないゲームでもディープラーニングは有効であるか検証するために，本研究では Python を用いてコネクト 4 の AI を作成し，ディープラーニングを行うことにより強い AI を目指す．そして，AI の作成後は他の AI と対戦させてその強さを検証する．

1.4 本報告書の構成

本報告書の構成は以下の通りである．まず第 2 章で，ディープラーニングとは何かというのを述べる．次に，第 3 章でディープラーニングを用いたゲーム AI について述べ，第 4 章でディープラーニングを用いて学習を行うために作成したプログラムについて述べる．その後，第 5 章で作成した AI についての評価，考察について述べ，第 6 章で結論を述べる．

2　ディープラーニングとは

ディープラーニングとは，脳の神経回路の仕組みを模したニューラルネットワークを多層に重ねることで，学習能力を高めた機械学習のうちの一つであり，「画像認識」，「音声認識」，「自然言語処理」などの分野で大きな戦果を挙げている[8]．

3　ディープラーニングを用いたゲーム AI

ディープラーニングは，先に述べたとおり画像や音声等の分野で使用されているが，これだけにとどまらず，2016 年に韓国のプロ棋士に，ディープラーニングを用いたゲーム AI である「AlphaGo」が 4 勝 1 敗で勝利した[7]．「AlphaGo」はモンテカルロ木探索という探索法を使用して着手選択を行なっている．モンテカルロ木探索とは，通常の木探索にランダム要素を加味した評価関数を用いることで効率よく探索を行う事ができるアルゴリズムのことである[12]．また，「AlphaGo」はニューラルネットワークの一種であるデュアルネットワークというものを使用しており，次の一手予測を行うポリシーネットワークと，勝率を予測するバリューネットワークを統合したモデルを使用している[12]．図 2 に「AlphaGo」のデュアルネットワークの構成を示す．このデュアルネットワークは黒石，白石の位置の各 1 チャネル，黒石の 1~7 手前の位置が 7 チャネル，白石 1~7 手前の位置が 7 チャネル，手番が 1 チャネルの計 17 チャネルの入力層，第 1 層は 3x3 サイズ 256 種類のフィルタを持つ畳み込み層，第 2 層~第 39 層では 19 個の残差ブロック構造になっており，各残差ブロックは 3x3 サイズ 256 種類のフィルタを持つ畳み込み層からなる[12]．第 40 層から次の一手の予測確率を算出する部分と，勝率を予測する部分の 2 つに分岐している[12]．

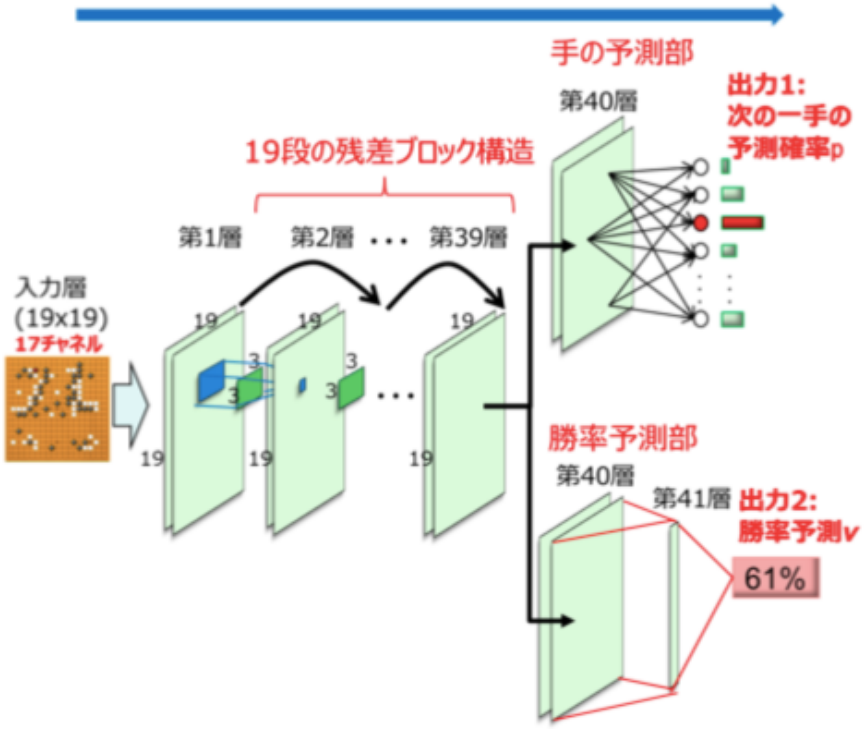

図 2 「Alpha Go」のデュアルネットワーク[12]

4 作成したプログラム

本研究では python を用いてディープラーニングにより着手選択をするコネクト 4 プログラムを作成した. 本研究で作成したプログラムは，モンテカルロ木探索を用いた自己対戦によりディープラーニングを行う AlphaZero[2][13]というプログラムを参考に AI を作成した.

4.1 AlphaZero

AlphaZero とは, 2018 年に DeepMind によって開発された人間の対局データや定石の知識を使わずに学習を行うボードゲーム AI で，AlphaGo の後継機種である[2]．AlphaZero は，自己対戦だけで人間の対局データを用いる AlphaGo を超える性能を実現することに成功した．また AlphaZero は，様々なボードゲーム向けに汎用化されており，チェスや将棋なども学習する事ができる.

4.2 プログラムの仕様

本節は，本研究で作成したプログラムの仕様について述べる．self_play.py の SP_GAME_COUNT というパラメータでセルフプレイの実行回数, train_network.py の RN_EPOCHS というパラメータで学習データでの学習回数, evaluate_network.py の EN_GAME_COUNT というパラメータで学習データの評価を行う回数, train_cycle.py の for 文の数字で繰り返しの回数を設定する事ができ，train_cycle.py をターミナル上で実行することによって自己学習が開始する．学習の終了後は，human_play.py を実行することで，作成した AI と対戦する事ができる.

4.3 プログラムの詳細

本節では，本研究で作成したプログラムの詳細について述べる．表 1 に今回作成したプログラムの一覧を示す．また，付録に本研究で作成したプログラムのソースを示す.

表 1 本研究で作成したプログラム一覧

game.py	コネクト 4 のゲーム状態の処理を行うプログラム
dual_network.py	デュアルネットワークのプログラム
pv_mcts.py	モンテカルロ木探索を行うためのプログラム
self_play.py	ゲームのセルフプレイを行い学習データを作成するためのプログラム
train_network.py	学習データで学習を行うためのプログラム
evaluate_network.py	新しい学習データの評価を行うためのプログラム
train_cycle.py	セルフプレイ，学習，評価のサイクルを自動で行うためのプログラム
human_play.py	作成した AI と対戦するためのプログラム ゲーム UI の実装も行なっている

以下では各プログラムの詳細について述べる．

- game.py

game.py はコネクト 4 のゲーム状態の処理を行うためのプログラムである．このプログラムはゲームの勝敗や終了判定，合法手のリストの取得などを行なう．表 2 に game.py のメソッドを示す．

表 2　game.py のメソッド一覧

__init__(self,pieces=None,enemy_pieces=None)	盤面の初期化
piece_count(self,pieces)	石の数の取得
is_lose(self)	負けかどうか
is_draw(self)	引き分けかどうか
is_done(self)	ゲーム終了かどうか
next(self,action)	次の状態の取得
legal_actions(self)	合法手リストの取得
is_first_player(self)	先手かどうか
__str__(self)	ゲーム状態の文字列表示

- dual_network.py

dual_network.py はデュアルネットワークの実装を行うためのプログラムである．表 3 に dual_network.py のメソッドを示す．

表 3　dual_network.py のメソッド一覧

residual_block()	残差ブロックの作成
dual_network()	デュアルネットワークの作成

- pv_mct.py

pv_mct.py はモンテカルロ木探索を行うためのプログラムである．表 4 に pv_mct.py のメソッドを示す.

表 4　pv_mct.py のメソッド一覧

predict(model, state)	推論を行う
nodes_to_score(nodes)	ノードのリストを試行回数のリストに変換する
pv_mcts_scores(models, state, temperature)	モンテカルロ木探索のスコアの取得
pv_mcts_action(model, temperature=0)	モンテカルロ木探索で行動選択
boltzman(xs, temperature)	ボルツマン分布の計算を行う

- self_play.py

self_play.py はゲームのセルフプレイを行い，学習データを作成するためのプログラムである．今回は 1 回の学習でセルフプレイを 500 回行う．表 5 に self_play.py のメソッドを示す.

表 5　self_play.py のメソッド一覧

first_player_value(ended_state)	先手プレイヤーの価値を決める
write_date(history)	学習データを保存する
play(model)	1 ゲーム実行する
self_play()	セルフプレイを行う

- train_network.py

train_network.py は学習データで学習を行うためのプログラムである．self_play.py 等により与えられた学習データを用いて最新プレイヤーのモデルを作成する．表 6 に train_network.py のメソッドを示す.

表 6　train_network.py のメソッド一覧

load_data()	学習データを読み込む
train_network()	デュアルネットワークの学習を行う

- evaluate_network.py

evaluate_network.py は新しい学習データの評価を行うためのプログラムである. 最新プレイヤーのモデルとベストプレイヤーのモデルを対戦させて，ベストプレイヤーの更新を行うかどうかを決める．表 7 に evaluate_network.py のメソッドを示す.

表 7　evaluate_network.py のメソッド一覧

first_player_point(ended_state)	先手プレイヤーのポイントを決める
play(next_action)	1 ゲーム実行する
update_best_player()	ベストプレイヤーの交代を行う
evaluate_network()	最新プレイヤーの評価を行う

- train_cycle.py

train_cycle.py は学習サイクルを実行するためのプログラムである．このプログラムでは，self_play.py，train_network.py，evaluate_network.py を順に実行する．今回は学習サイクルを 15 回行った.

- human_play.py

human_play.py 作成した AI と対戦するためのプログラムである．また，ゲーム UI の実装も行なっている．表 8 に human_play.py のメソッドを示す．また，図 3 に human_play.py を実行した時の様子を示す.

表 8　human_play.py のメソッド一覧

__init__(self,master=None,model=None)	ゲームの初期化を行う
turn_of_human(self,event)	人間のターンの処理を行う
turn_of_ai(self)	AI のターン処理を行う
draw_piece(self,index,first_player)	石の描画を行う
on_draw(self)	盤面と石の描画の更新を行う

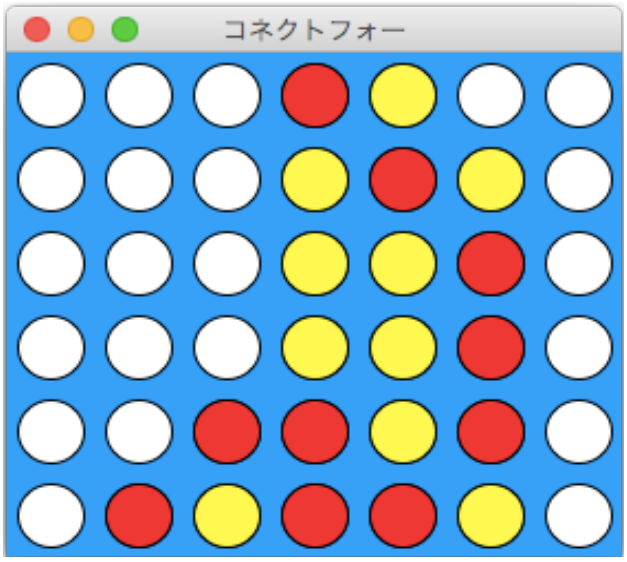

図 3　human_play.py の実行の様子

5　AI の性能評価および考察

今回作成した AI をフリーゲームのコネクト 4 [5]の CPU と 10 回対戦させたところ，勝率は 20%だった．この結果になった理由として考えられるのは， CPU はミスをせずに AI が先にミスをしてしまっていたので勝率が低くなってしまったという事である．また，Connect 4 Solver [4]と対戦させても勝つことができなかった．AI のミスの理由は，自己対戦で学習させていることにより，ミスがある AI 同士での対戦になってしまっていたことが考えられる．これの解決策として，初めから強い AI との対戦だと学習が進まない可能性が考えられるので，最初は自己対戦での学習をして，ある程度の強さになれば Connect 4 Solver[4]などの強い AI と対戦させる事が良いのではないかと考える．また以上の対戦結果から，選択肢の少ないゲームにおいてディープラーニングが多少は有効であるものの， α β 法などよりは優れていない事がわかった．

6　結論

本研究では，python を用いてコネクト 4 の作成した．今回作成した AI では着手選択をする際にミスが目立ってしまう結果となった. また, 選択肢の少ないゲームにおいてディープラーニングが多少は有効である事がわかった. 今後の課題としてあげられるのは，どのようにして AI のミスを減らすような学習をさせるかどうかという点である．

謝辞

本研究において丁寧にご指導くださいました石水隆教授に感謝いたします.

参考文献

[1] 中塚恭右：Python で作るコネクト４，近畿大学卒業研究報告書 (2020)

[2] 強化学習 Alpha Zero 18(コネクト4) https://ailog.site/2019/10/12/zero10/

[3] Victor Allis: A Knowledge-based Approach of Connect-Four, The Game is Solved: White Wins, Master Thesis, Department of Mathematics and Computer Science Vrije Universiteit (1988) http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf

[4] Pasal Pons：Connect 4 Solver (2015) https://connect4.gamesolver.org/

[5] 4 目並べゲーム[Connect4]，ひといきゲーム http://hitoikigame.com/blog-entry-5637.html

[6] 中村紘也：ニューラルネットワークとは｜仕組み・学習手法活用事例・ディープラーニングとの違い，ledge.ai，(20202/10)https://ledge.ai/neural-network/

[7] 小寺貴之：「グーグル囲碁 AI「４勝１敗」は人類の敗北か，プロ棋士から見た勝負の評価」ニュースイッチ，2016/3/6 日刊工業新聞，(2016) https://newswitch.jp/p/3971

[8] 柏川 元希：ディープ・ラーニングとは|意味・AI，機械学習との違い・仕組み・学習方法から応用例まで，ledge.ai，(2020/02/10) https://ledge.ai/deep-learning/

[9] Yoshiaki Yamaguchi, Todd W. Neller, First Player's Cannot-Lose Strategies for Cylinder-Infinite-Connect-Four with Widths 2 and 6, Advances in Computer Games, pp.113-121 (2015) http://cs.gettysburg.edu/~tneller/papers/acg2015.pdf

[10] James Dow Allen, The Complete Book of Connect 4, Puzzle Wright Press (2010)

[11] James Dow Allen, Expert Play in Connect-Four, (1990), http://tromp.github.io/c4.html

[12] 大槻知史：アルファ碁からアルファ碁ゼロへ，(2017) http://home.q00.itscom.net/otsuki/alphaZero.pdf

[13] syu-hei：Reversi-AlphaZero(2020) https://github.com/syu-hei/Reversi-AlphaZero

[14] ディープ・ラーニングとは，SAS，SAS Institute Japan 株式会社，https://www.sas.com/ja_jp/insights/analytics/deep-learning.html#dlbreakthroughs

ソースプログラム

本研究で作成したソースファイルの一部を以下に示す.

- game.py

```
import random
import math

class State:
    #初期化する
def __init__(self, pieces=None, enemy_pieces=None):
#石の配置
    self.pieces = pieces if pieces != None else [0] * 42
    self.enemy_pieces = enemy_pieces if enemy_pieces != None else [0] * 42
#石の数を取得する
def piece_count(self, pieces):
    count = 0
    for i in pieces:
        if i == 1:
            count +=  1
    return count
#負けかどうか判別する
def is_lose(self):
    def is_comp(x, y, dx, dy):
        for k in range(4):
            if y < 0 or 5 < y or x < 0 or 6 < x or ¥
                self.enemy_pieces[x+y*7] == 0:
                return False
            x, y = x+dx, y+dy
        return True

    for j in range(6):
        for i in range(7):
            if is_comp(i, j, 1, 0) or is_comp(i, j, 0, 1) or ¥
                is_comp(i, j, 1, -1) or is_comp(i, j, 1, 1):
                return True
    return False
#引き分けかどうか判別する
def is_draw(self):
    return self.piece_count(self.pieces) + self.piece_count(self.enemy_pieces) == 42
```

```
#ゲーム終了かどうか判別する
def is_done(self):
    return self.is_lose() or self.is_draw()
#次の状態を取得する
def next(self, action):
    pieces = self.pieces.copy()
    for j in range(5,-1,-1):
        if self.pieces[action+j*7] == 0 and self.enemy_pieces[action+j*7] == 0:
            pieces[action+j*7] = 1
            break
    return State(self.enemy_pieces, pieces)
#合法手のリストを取得する
def legal_actions(self):
    actions = []
    for i in range(7):
        if self.pieces[i] == 0 and self.enemy_pieces[i] == 0:
            actions.append(i)
    return actions
#先手かどうか
def is_first_player(self):
    return self.piece_count(self.pieces) == self.piece_count(self.enemy_pieces)
#文字列の表示
def __str__(self):
    ox = ('o', 'x') if self.is_first_player() else ('x', 'o')
    str = ''
    for i in range(42):
        if self.pieces[i] == 1:
            str += ox[0]
        elif self.enemy_pieces[i] == 1:
            str += ox[1]
        else:
            str += '-'
        if i % 7 == 6:
            str += '¥n'
    return str
```

- dual_net_work.py

```
from tensorflow.keras.layers import Activation, Add, BatchNormalization, Conv2D,
Dense, GlobalAveragePooling2D, Input
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
from tensorflow.keras import backend as K
import os
#パラメータの準備
DN_FILTERS  = 128
DN_RESIDUAL_NUM =  16
DN_INPUT_SHAPE = (7, 6, 2)
DN_OUTPUT_SIZE = 7
   #畳み込み層の作成
def conv(filters):
    return Conv2D(filters, 3, padding='same', use_bias=False,
                kernel_initializer='he_normal', kernel_regularizer=l2(0.0005))
    #残差ブロックの作成
def residual_block():
    def f(x):
        sc = x
        x = conv(DN_FILTERS)(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = conv(DN_FILTERS)(x)
        x = BatchNormalization()(x)
        x = Add()([x, sc])
        x = Activation('relu')(x)
        return x
    return f
     #デュアルネットワークの作成
def dual_network():
#モデルがあれば無処理
    if os.path.exists('./model/best.h5'):
        return
   #入力層
    input = Input(shape=DN_INPUT_SHAPE)

#畳み込み層
    x = conv(DN_FILTERS)(input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
```

```
#残差ブロックx16
    for i in range(DN_RESIDUAL_NUM):
      x = residual_block()(x)

#プーリング層
    x = GlobalAveragePooling2D()(x)

#ポリシー出力
    p = Dense(DN_OUTPUT_SIZE, kernel_regularizer=l2(0.0005),
        activation='softmax', name='pi')(x)

#バリュー出力
    v = Dense(1, kernel_regularizer=l2(0.0005))(x)
    v = Activation('tanh', name='v')(v)

#モデルの作成
    model = Model(inputs=input, outputs=[p,v])

#モデルの保存
    os.makedirs('./model/', exist_ok=True)
    model.save('./model/best.h5')

#モデルの破棄
    K.clear_session()
    del model
```

- pv_mcts.py

```
from game import State
from dual_network import DN_INPUT_SHAPE
from math import sqrt
from tensorflow.keras.models import load_model
from pathlib import Path
import numpy as np

#パラメータの準備
PV_EVALUATE_COUNT = 50

#推論
def predict(model, state):
```

```
#推論のための入力データシェイプの変換
    a, b, c = DN_INPUT_SHAPE
    x = np.array([state.pieces, state.enemy_pieces])
    x = x.reshape(c, a, b).transpose(1, 2, 0).reshape(1, a, b, c)
    #推論
    y = model.predict(x, batch_size=1)
    #方策の取得
    policies = y[0][0][list(state.legal_actions())]
    policies /= sum(policies) if sum(policies) else 1
    #価値の取得
    value = y[1][0][0]
    return policies, value
#ノードのリストを試行回数のリストに変換
def nodes_to_scores(nodes):
    scores = []
    for c in nodes:
        scores.append(c.n)
    return scores

#モンテカルロ木探索のスコアの取得
def pv_mcts_scores(model, state, temperature):
    class Node:
#ノードの初期化
        def __init__(self, state, p):
            self.state = state
            self.p = p
            self.w = 0
            self.n = 0
            self.child_nodes = None
        #局面の価値の計算
        def evaluate(self):
            if self.state.is_done():
                value = -1 if self.state.is_lose() else 0

                self.w += value
                self.n += 1
                return value
            #子ノードが存在しないとき
            if not self.child_nodes:
```

```
#ニューラルネットワークの推論で方策と価値を取得
            policies, value = predict(model, self.state)

            self.w += value
            self.n += 1

            self.child_nodes = []
            for action, policy in zip(self.state.legal_actions(), policies):
                self.child_nodes.append(Node(self.state.next(action), policy))
            return value
        #子ノードが存在するとき
        else:
            value = -self.next_child_node().evaluate()

            self.w += value
            self.n += 1
            return value
    #アーク評価値が最大の子ノードを取得
    def next_child_node(self):
        C_PUCT = 1.0
        t = sum(nodes_to_scores(self.child_nodes))
        pucb_values = []
        for child_node in self.child_nodes:
            pucb_values.append((-child_node.w / child_node.n if child_node.n else
0.0) +
                               C_PUCT * child_node.p * sqrt(t) / (1 + child_node.n))
        #アーク評価値が細田の子ノードを返す
        return self.child_nodes[np.argmax(pucb_values)]

    root_node = Node(state, 0)
    #複数回評価の実行
    for _ in range(PV_EVALUATE_COUNT):
        root_node.evaluate()
    #合法手の確率分布
    scores = nodes_to_scores(root_node.child_nodes)
    if temperature == 0:
        action = np.argmax(scores)
        scores = np.zeros(len(scores))
        scores[action] = 1
```

```
    else:
        scores = boltzman(scores, temperature)
    return scores
#モンテカルロ木探索で行動選択
def pv_mcts_action(model, temperature=0):
    def pv_mcts_action(state):
        scores = pv_mcts_scores(model, state, temperature)
        return np.random.choice(state.legal_actions(), p=scores)
    return pv_mcts_action
#ポルツマン分布
def boltzman(xs, temperature):
    xs = [x ** (1 / temperature) for x in xs]
    return [x / sum(xs) for x in xs]
```

- self_play.py

```
from game import State
from pv_mcts import pv_mcts_scores
from dual_network import DN_OUTPUT_SIZE
from datetime import datetime
from tensorflow.keras.models import load_model
from tensorflow.keras import backend as K
from pathlib import Path
import numpy as np
import pickle
import os
#パラメータの準備
SP_GAME_COUNT = 500#セルフプレイの回数
SP_TEMPERATURE = 1.0
#先手プレイヤーの価値
def first_player_value(ended_state):
    if ended_state.is_lose():
        return -1 if ended_state.is_first_player() else 1
    return 0
#学習データを保存する
def write_data(history):
    now = datetime.now()
    os.makedirs('./data/', exist_ok=True)
    path = './data/{:04}{:02}{:02}{:02}{:02}{:02}.history'.format(now.year,
now.month, now.day, now.hour, now.minute, now.second)
```

```
    with open(path, mode='wb') as f:
        pickle.dump(history, f)
#1ゲーム実行する
def play(model):
    history = []

    state = State()

    while True:
#ゲーム終了時
        if state.is_done():
            break
    #合法手の確率分布を取得する
        scores = pv_mcts_scores(model, state, SP_TEMPERATURE)
        #学習データに方策を追加する
        policies = [0] * DN_OUTPUT_SIZE
        for action, policy in zip(state.legal_actions(), scores):
            policies[action] = policy
        history.append([[state.pieces, state.enemy_pieces], policies, None])

        action = np.random.choice(state.legal_actions(), p=scores)

        state = state.next(action)
    #学習データに価値を追加する
    value = first_player_value(state)
    for i in range(len(history)):
        history[i][2] = value
        value = -value
    return history
#セルフプレイをする
def self_play():
    history = []
   # ベストプレイヤーのモデルを読み込む
    model = load_model('./model/best.h5')
    #指定した回数ゲームを実行する
    for i in range(SP_GAME_COUNT):
        h = play(model)
        history.extend(h)
```

```
#ゲーム回数を出力する
        print('¥rSelfPlay {}/{}'.format(i+1, SP_GAME_COUNT), end='')
    print('')
#学習データを保存する
    write_data(history)
    #モデルを破棄する
    K.clear_session()
    del model
```

- train_network.py

```
from dual_network import DN_INPUT_SHAPE
from tensorflow.keras.callbacks import LearningRateScheduler, LambdaCallback
from tensorflow.keras.models import load_model
from tensorflow.keras import backend as K
from pathlib import Path
import numpy as np
import pickle
#パラメータの準備
RN_EPOCHS = 100 #学習回数を決める
#学習データを読み込む
def load_data():
    history_path = sorted(Path('./data').glob('*.history'))[-1]
    with history_path.open(mode='rb') as f:
        return pickle.load(f)
#デュアルネットワークの学習
def train_network():
    history = load_data()
    xs, y_policies, y_values = zip(*history)

    a, b, c = DN_INPUT_SHAPE
    xs = np.array(xs)
    xs = xs.reshape(len(xs), c, a, b).transpose(0, 2, 3, 1)
    y_policies = np.array(y_policies)
    y_values = np.array(y_values)
    #ベストプレイヤーのモデルの読み込み
    model = load_model('./model/best.h5')
    #モデルをコンパイルする
    model.compile(loss=['categorical_crossentropy', 'mse'], optimizer='adam')
```

```
#学習率
    def step_decay(epoch):
        x = 0.001
        if epoch >= 50: x = 0.0005
        if epoch >= 80: x = 0.00025
        return x
    lr_decay = LearningRateScheduler(step_decay)
    #出力
    print_callback = LambdaCallback(
           on_epoch_begin=lambda epoch,logs:
               print('¥rTrain {}/{}'.format(epoch + 1,RN_EPOCHS), end=''))
      #指定した回数分学習を実行する
    model.fit(xs, [y_policies, y_values], batch_size=128, epochs=RN_EPOCHS,
            verbose=0, callbacks=[lr_decay, print_callback])
    print('')
#最新プレイヤーのモデルの保存
    model.save('./model/latest.h5')
#モデルの破棄
    K.clear_session()
    del model
```

- evaluate_network.py

```
from game import State
from pv_mcts import pv_mcts_action
from tensorflow.keras.models import load_model
from tensorflow.keras import backend as K
from pathlib import Path
from shutil import copy
import numpy as np
#パラメータの準備
EN_GAME_COUNT = 10 #評価するためのゲーム数
EN_TEMPERATURE = 1.0
#先手プレイヤーのポイント
def first_player_point(ended_state):
    if ended_state.is_lose():
#先手勝ち1点,先手負け0点,引き分け0.5点
        return 0 if ended_state.is_first_player() else 1
    return 0.5
```

```
#１ゲーム実行する
def play(next_actions):
    state = State()
    #ゲーム終了まで
    while True:
#ゲーム終了時
        if state.is_done():
            break;
    #行動を取得する
        next_action = next_actions[0] if state.is_first_player() else next_actions[1]
        action = next_action(state)
        #次の状態を取得する
        state = state.next(action)
#先手プレイヤーのポイントを返す
    return first_player_point(state)
#ベストプレイヤーを交代する
def update_best_player():
    copy('./model/latest.h5', './model/best.h5')
    print('Change BestPlayer')
#ネットワークを評価する
def evaluate_network():
#最新プレイヤーを読み込む
    model0 = load_model('./model/latest.h5')

#ベストプレイヤーを読み込む
    model1 = load_model('./model/best.h5')

#PV MCTSで行動選択を行う関数の生成
    next_action0 = pv_mcts_action(model0, EN_TEMPERATURE)
    next_action1 = pv_mcts_action(model1, EN_TEMPERATURE)
    next_actions = (next_action0, next_action1)
    #指定した回数の対戦を繰り返す
    total_point = 0
    for i in range(EN_GAME_COUNT):
        if i % 2 == 0:
            total_point += play(next_actions)
        else:
            total_point += 1 - play(list(reversed(next_actions)))
```

```
#出力
        print('¥rEvaluate {}/{}'.format(i + 1, EN_GAME_COUNT), end='')
    print('')
#平均ポイントを計算する
    average_point = total_point / EN_GAME_COUNT
    print('AveragePoint', average_point)

#モデルの破棄
    K.clear_session()
    del model0
    del model1

#ベストプレイヤーの交代
    if average_point > 0.5:
        update_best_player()
        return True
    else:
        return False
```

- train_cycle.py

```
from dual_network import dual_network
from self_play import self_play
from train_network import train_network
from evaluate_network import evaluate_network

#デュアルネットワークの作成
dual_network()
#15サイクル実行する
for i in range(15):
    print('Train',i,'====================')
#セルフプレイ
    self_play()

#パラメータ更新
    train_network()

#新パラメータ評価部
    evaluate_network()
```

- human_play.py

```
from game import State
from pv_mcts import pv_mcts_action
from tensorflow.keras.models import load_model
from pathlib import Path
from threading import Thread
import tkinter as tk
#ベストプレイヤーのモデルの読み込み
model = load_model('./model/best.h5')
#ゲームUIの定義
class GameUI(tk.Frame):
#初期化を行う
 def __init__(self, master=None, model=None):
    tk.Frame.__init__(self, master)
    self.master.title('コネクトフォー')
    #ゲーム状態を生成する
    self.state = State()
    #PV MCTSで行動選択を行う関数を生成する
    self.next_action = pv_mcts_action(model, 0.0)
    #キャンバスを生成する
    self.c = tk.Canvas(self, width = 280, height = 240, highlightthickness = 0)
    self.c.bind('<Button-1>', self.turn_of_human)
    self.c.pack()

    #描画の更新を行う
    self.on_draw()
#人間のターン
def turn_of_human(self, event):
    if self.state.is_done():
        self.state = State()
        self.on_draw()
        return
    #先手でない時
    if not self.state.is_first_player():
        return
    #クリック位置を行動に変換
    x = int(event.x/40)
    if x < 0 or 6 < x:
        return
```

```
    action = x
#合法手でない場合
    if not (action in self.state.legal_actions()):
        return
    #次の状態の取得
    self.state = self.state.next(action)
    self.on_draw()
    #AIのターンに移る
    self.master.after(1, self.turn_of_ai)

#AIのターン
def turn_of_ai(self):
    if self.state.is_done():
        return
    #行動の取得
    action = self.next_action(self.state)
#次の状態を取得する
    self.state = self.state.next(action)
    self.on_draw()
#石の描画を行う
def draw_piece(self, index, first_player):
    x = (index%7)*40+5
    y = int(index/7)*40+5
    if first_player:
        self.c.create_oval(x, y, x+30, y+30, width = 1.0, fill = '#FF0000')
    else:
        self.c.create_oval(x, y, x+30, y+30, width = 1.0, fill = '#FFFF00')
#描画の更新を行う
def on_draw(self):
    self.c.delete('all')
    self.c.create_rectangle(0, 0, 280, 240, width = 0.0, fill = '#00A0FF')
    for i in range(42):
        x = (i%7)*40+5
        y = int(i/7)*40+5
        self.c.create_oval(x, y, x+30, y+30, width = 1.0, fill = '#FFFFFF')

    for i in range(42):
        if self.state.pieces[i] == 1:
            self.draw_piece(i, self.state.is_first_player())
```

```
        if self.state.enemy_pieces[i] == 1:
            self.draw_piece(i, not self.state.is_first_player())
#ゲームUIを実行する
f = GameUI(model=model)
f.pack()
f.mainloop()
```