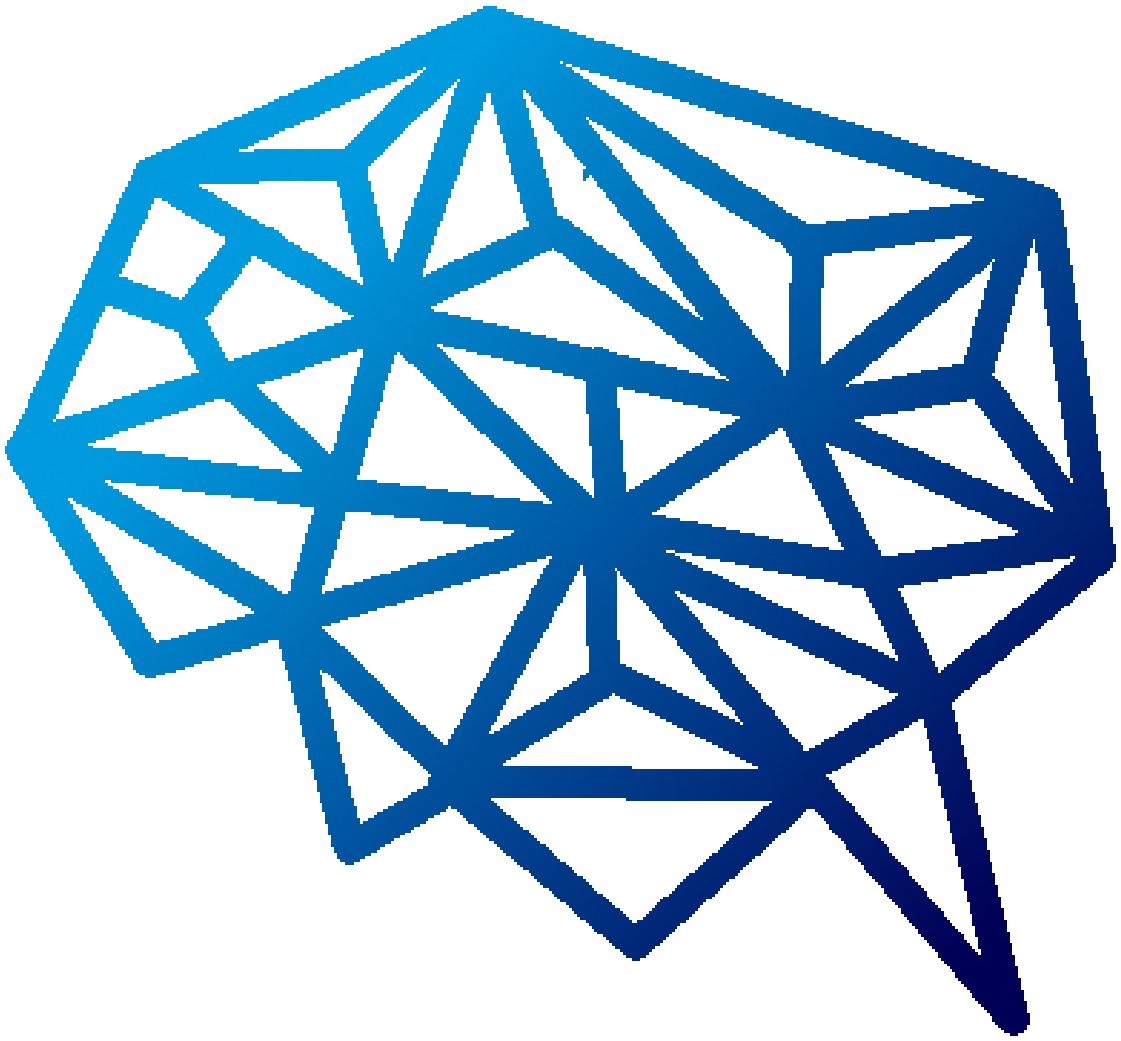


ALPHA GOMOKU

AN AI MADE FOR GOMOKU

2年4組　阿久津敦弘・鈴木悠太・土井翔

ALPHA GOMOKU

AN AI MADE FOR GOMOKU

阿久津敦弘・鈴木悠太・土井翔

1. 研究背景

A. なぜ AI をつくるのか

2015 年に Alpha Go という AI がプロの囲碁の世界チャンピオンに勝ったということを知り、その AI と対戦してみたくなった。そこで AI を自作する必要があったので本研究を始めた。

B. なぜ五目並べの AI をつくるのか

本当は囲碁の AI をつくろうとしたが、研究メンバー全員が囲碁のルールを知らなかったので断念した。メンバー全員が五目並べのルールを知っていたので、五目並べの AI をつくることにした。

2. 研究目的

本研究の目的は以下の 2 つである。

- 人間に勝ち越すことができる五目並べの AI をつくる。
- 実装した AI でアルゴリズムによる性能の差を明らかにする

3. 研究方法

A. 五目並べのルール

本研究では五目並べのルールは以下のように定義する。

- 二人で遊ぶ
- L x L のマスを用意する（最初は L=7 で固定していたが、MCTS を実装した AI が強すぎたので、途中から L=9 に変更した。αβ法と MCTS で実装したときの結果は L=7 のもの。また MCTS 以降に実装したものは L=9 のもの。）
- 二人のプレイヤーが交互に石が置いてないマスに自分の色の石を置く

関連する WEB ページ

Deep Mind

https://www.deepmind.com

Tensorflow

https://www.tensorflow.org

Keras

https://keras.io

- どちらかの色の石が、縦か横か斜めに連続して五つ並ぶとその色のプレイヤーの勝ちとする
- 全てのマスが埋まった場合は引き分けとする
- どちらかが勝つか引き分けになるとゲーム終了

B. 開発環境

本研究で作成したプログラムは以下を使用し、作成した。

- Visual Studio Code (for Editor)
- Chrome Dev Tools (for Debug Tools)
- Node.js (for test)
- Tensorflow.js (library)

C. 実行環境

本研究は以下の環境下で実行した。

- Chrome browser
- Mozzila Fire Fox

4. MINIMAX

A. 戦略

I. ゲーム木

ゲームで次にどのような局面になるかを木にしたものを**ゲーム木**という。一般に木の節目を**ノード(Node)**、枝を**ブランチ(Branch)**、根を**ルート(Root)**、末端を**リーフ(Leaf)**という。

II. MINIMAX 法

私たちは以下のように探索すれば強い AI ができると考えた。

- ある局面が自分にとってどれぐらい有利かを数値化する評価関数をつくる
- ノードが自分のターンのときは子ノードから評価値が最大のものを選択する
- ノードが相手のターンのときは子ノードから評価値が最小のものを選択する

http://www.es-cube.net/es-cube/reversi/sample によると、この戦略を **Minimax 法**という。

III. AB法

Minimax 法の探索は無駄が多い。http://www.es-cube.net/es-cube/reversi/sample にはこれを解決する方法として、**αβ法**が挙げられている。これは以下のように探索する。

- 変数 a,b を用意する（a=-∞,b=∞を代入）
- ノードが自分のターンのときは子ノードから評価値が最大のものを選択し、a に評価値と a のうち大きいほうを代入する
- ノードが相手のターンのときは子ノードから評価値が最小のものを選択し、b に評価値と b のうち小さいほうを代入する

Minimax 法の出力とαβ法の出力は一致するので、本研究ではαβ法を実装する。

B. 結果

AI は 10 戦中 2 回勝利し、8 回負けた。処理時間はバラつき平均 10000ms であったが、30000ms かかるときもあった。

C. 考察

非常に弱かった。更にやや処理時間が長かった。評価関数によって戦術が大きく変化した。AI をつくっている人が五目並べの必勝法を知らないと強い AI をつくれないことが分かった。したがって人がつくる評価関数を必要としない AI をつくる必要がある。

5. MCTS

A. 戦略

I. 強化学習

ある環境内である行動を選択したときの報酬から最適な方策を学習するものを**強化学習**という。強化学習する AI は人がつくる評価関数を必要としないため、強い AI をつくることができると考えた。

II. モンテカルロ法

一般に、乱数を用いて課題を解決するアプローチを**モンテカルロ法**という。

III. MCTS

モンテカルロ法を木の探索に応用したものを **MCTS** という。具体的にはランダムに子ノードから選択することを繰り返し（playout）、その報酬の平均から次の手を選択する。MCTS は Alpha Go に使われた技術の中の一つである。

B. 結果

各ノードに対し 500 回探索した AI はマスが 7 x 7 のとき 10 回中 8 回勝利し 1 回も負けなかった。またマスが 9 x 9 のときは 10 回中 4 回勝利し、2 回負けた。処理時間はマスが 7 x 7 のときは平均 10000ms 程度で 9 x 9 のときは平均 60000ms 程度であった。

C. 考察

7 x 7 のときはかなり強かった。ただ 9 x 9 のときは処理時間が非常に長く、あまり強くなかった。探索した結果を 1 度しか利用していないので無駄が多いと考えられる。よって探索結果を利用できるようにするべきである。

6. UCT

A. 戦略

I. UCB1

以下の問題を**多腕バンディット問題**という。

> 複数のアームを持つ「スロットマシン」がある。アームごとにコインが出る確率 p が決まっているが、p は未知数である。このとき、決められた回数で多くの当たりを引くにはどのようにアームを選択するのがよいか。

この問題では、**知識の利用**と**探索**のバランスが課題となる。MCTS のように探索だけであると、最もコインが出るアームを選択する回数が減ってしまう。これに対し以下の戦略を **UCB1** という。

- 探索していない手があるときその手を探索する
- 全て探索したときは、報酬の期待値とその値の不確かさ（総試行回数を N、対象のノードの試行回数を n としたとき不確かさは $\sqrt{\frac{2\log N}{n}}$ とする）の和が最大の子ノードを選択する

II. UCT

UCB1 を MCTS に応用したものを **UCT** という。UCT では UCB1 で全体で一定回数探索し、最後に訪問回数が最大のノードを選択する。

B. 結果

総シュミレーション時間が 10000ms のとき、AI は 50 回中 26 回勝ち、22 回負けた。処理時間に制限を付けたため処理時間は全て 10000ms になった。

C. 考察

MCTS に比べかなり強かった。また処理時間が短い。ただ先手なのに関わらず負けているので、まだまだ改良の余地がある。囲碁の世界チャンピオンに勝った AI を参考に実装すれば強くなるのではないかと考えた。

7. ALPHAZERO

A. 戦略

I. 価値反復法

価値反復法とは、ある行動をとったときに次の状態価値と今の状態価値の差分を計算し、その差分だけ今の状態価値を変化させる手法である。

II. ベルマン方程式

未来の報酬の総和は確定していないため、状態 s と方策 a を固定する。

このとき、以下の式

行動価値関数（Q 関数）　$Q(s_t, a_{t+1}) = R_{t+1} + \gamma * Q(s_{t+1}, a_{t+1})$

状態価値関数（V 関数）　$V(s_t, a_t) = R_{t+1} + \gamma * V(s_{t+1}, a_{t+1})$

をまとめて**ベルマン方程式**という。ベルマン方程式は「次の状態が、今の状態と選択した行動によって決定する。」というシステム（マルコフ決定過程）の中で使うことができる。

III. Q 学習

Q 学習は価値反復法の一種である。Q 学習は状態価値関数による行動を選択する。

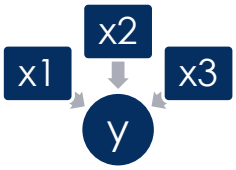
Q 学習では行動価値関数の更新式は　$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (R_{t+1} + \gamma * \max(Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)))$　となる。

この中の $R_{t+1} + \gamma * \max\big(Q(s_{t+1}, a_{t+1})\big) - Q(s_t, a_t)$ を **TD 誤差**といい、この誤差を 0 に近づけることで行動価値関数が正確になる。Q 学習は行動価値関数を表形式で表す。

I. ニューラルネットワーク

1) ニューラルネットワーク

人間の脳は大量の神経細胞で構成されている。それぞれの神経細胞をニューロンといい、ニューロンをモデル化すると以下のようになる。活性化関数で値を処理する。



ニューロンを層にしてたくさん積み重ねたものを**ニューラルネットワーク**という。python でニューラルネットワークを実現するライブラリに tensorflow というものがある。本研究ではこれを利用する。

2) 活性化関数

活性化関数はいくつか種類があり、用途によって使い分ける。

以下のもの以外にも linear 関数や step 関数などがある。

(A) SIGMOID 関数

sigmoid 関数は $f(x) = \frac{1}{1+e^{-x}}$ という式であらわされる。主に 2 つの解をもつ問題で使われている

(B) TANH 関数

tanh 関数は$f(x) = \tanh(x)$ という式であらわされる。sigmoid 関数の問題点を改良したものである。

(C) RELU 関数

relu 関数は$f(x) = \max(0, x)$ という式であらわされる。画像処理などで使われている。

(D) SOFTMAX 関数

softmax 関数は出力の総和が 1 になるようにする関数である。複数のものに分類する問題で使われている。

3) CNN

ニューラルネットワークの中で畳み込みという処理をするものを **CNN (畳み込みニューラルネットワーク)**という。畳み込みとは、入力から特徴を抽出する操作である。

4) RESNET

一般に CNN の層を多く積み重ねすぎると性能が低下してしまう。そこでショートカットする経路を加えることによって層を多く積み重ねることができるようにしたものを **ResNet** という。以下にサンプルコードを載せる。

```
from keras.models import Model
from keras.layers import Input,Dense,Dropout,Conv2D,BatchNormalization,Reshape,Activation,
Add,GlobalAveragePooling2D
from keras.regularizers import l2
from keras.optimizers import Adam
INPUT_SIZE=(9,9,2)
OUTPUT_SIZE=9**2
def conv(filters):
```

```
    return Conv2D(filters,3,padding="same",use_bias=False,kernel_initializer="he_normal",k
ernel_regularizer=l2(0.0005))
def ResNet():
    def func(x):
        sc=x
        x=conv(128)(x)
        x=BatchNormalization()(x)
        x=Activation("relu")(x)
        x=conv(128)(x)
        x=BatchNormalization()(x)
        x=Add()([x,sc])
        x=Activation("relu")(x)
        return x
    return func

def NNet():
    input_layer=Input(shape=INPUT_SIZE)
    x=conv(128)(input_layer)
    x=BatchNormalization()(x)
    x=Activation("relu")(x)

    for _ in range(15):
        x=ResNet()(x)

    x=GlobalAveragePooling2D()(x)

    p=Dense(OUTPUT_SIZE,kernel_regularizer=l2(0.0005),activation="softmax",name="pi")(x)

    v=Dense(1,kernel_regularizer=l2(0.0005))(x)
    v=Activation("tanh",name="v")(x)
    return Model(inputs=input_layer,outputs=[p,v])
```

II. DQN

Q 学習は表形式であるため、状態数が多いと学習に時間がかかる。Q 学習の行動価値関数をニューラルネットワークに置き換えたものを **DQN (Deep Q Network)**という。DQN では以下の工夫をして学習が効率的にすすむようにしている。

1) EXPERIENCE REPLAY

時間的に相関関係が高いデータを連続して学習すると学習が安定しないため、経験として一時的にデータをためて、そこからランダムに学習することを **Experience Replay** という。

2) FIXED TARGET Q-NETWORK

Q 学習では更新式で自分自身を利用していたが、そうすると学習が安定しないため、ネットワークを分離することを **Fixed Target Q-Network** という。

3) REWARD CLIPPING

環境の違いに対応するために報酬を -1 ～ 1 に固定することを **Reward Clipping** という。

4) HUBER LOSS

ニューラルネットワークの誤差が大きいときに誤差関数に mse を使用すると、出力が大きすぎて学習が安定しない。そこで誤差関数に huber 関数を使用することを **Huber Loss** という。

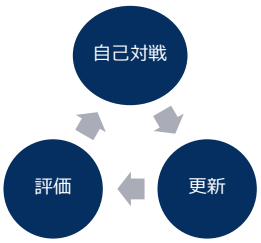
III. PV MCTS

ニューラルネットワークを用いて、次に打つ手の打つ確率分布 p と盤面の価値 v を求める。Alpha Zero は UCT の UCB1 の代わりに次の式（アーク評価値）を用いたものを **pv MCTS** という。

$\frac{w}{n} + c_{puct} * p * \frac{\sqrt{N}}{1+n}$ （w : 累計価値（v で初期化）　n : ノードの試行回数　N : 総試行回数　cpuct : 定数）

IV. ALPHAZERO の構成

論文によると、Alpha Zero の構成は以下のようになっている。



本研究でも同様の構成にする。

B. 結果

python3 で実装した。100 世代と対戦した結果、20 戦中 16 回勝利し、1 回も負けなかった。処理時間は 10000ms ぐらいであった。

C. 考察

UCT と比べても、とても強かった。ただ python3 で 10000ms かかるので javascript ではさらに時間がかかるかもしれない。

8. 結論

4.～7.より AlphaZero が最も性能が優れている。Javascript で実装したものの中では UCT が最も優れていた。

9. 今後の展望

2018 年に発表された R2D2 でポーカーなど（不完全情報ゲーム）の AI を実装してみたい。

10. 参照文献

@narisan25. *Chainer で DQN。強化学習を三目並べでいろいろ試してみた。(Deep Q Network,Q-Learning モンテカルロ)*. 2016 年 12 月 01 日. <https://qiita.com/naisan25/items/e64a5741864d5a3b0db0>.

AlphaZero 深層学習・強化学習・探索. 日付不明.

DeepMind. *Playing Atari with Reinforcement Learning*. 2013. pdf 文書.

11. 付録

A. 五目並べを遊ぶには

本研究で作成したプログラムは Web アプリとして公開している。

https://7777777TEST.github.io/Alpha-Gomoku にアクセスすると遊ぶことができる。

B. ソースコードを使うには

本研究で作成したプログラムは MIT ライセンス（どんなに自由に使ってもよい、ただし無保証）で公開している。

I. ソースコードをダウンロードする

ソースコードは以下のサイトで公開している。Download ボタンを押すとダウンロードできる。

https://github.com/7777777TEST/Alpha-Gomoku

II. ソースコードの構成

index.html は html の中心部分で、css/main.css はその見た目を設定している。

src ディレクトリに入っているファイルがアプリの中心部分である。

index.js → 描写を行う処理

gomoku.js → 五目並べのゲームのロジック

特に AI ディレクトリに入っているファイルは AI の中心部分である。

minimax.js → Minimax を実装したもの

MCTS.js → MCTS を実装したもの

UCT.js → UCT を実装したもの

C. プログラミング言語について

I. JAVASCRIPT について

Javascript については以下のサイトで詳しく解説されている。

Mozilla Developer https://developer.mozilla.org/ja/docs/Web/Javascript

II. PYTHON3 について

AlphaZero をつくるときに python3 を利用した。python3 については以下のサイトが詳しい。

Python https://python.org

D. ライブラリについて

AlphaZero を実装する際に Tensorflow というライブラリを利用した。Tensorflow の公式のサイトには Tensorflow の利用例が多く載っている。Tensorflow は Tensor という行列のようなものを使用してニューラルネットワークを実現する。また Tensorflow に統合されている Keras の公式サイトには Keras の使用方法が載っている。

Tensorflow https://www.tensorflow.org

Keras https://keras.io