

2019 年度 卒業研究

ディープラーニングによる将棋 AI の作成

2020 年 1 月 13 日

早稲田大学 教育学部 数学科
学籍番号 1E16M068-4

峰岸 零

目次

- 第 1 章 はじめに
 - 1.1 テーマ設定
 - 1.2 プログラミング言語の選択
- 第 2 章 機械学習の概要
 - 2.1 機械学習とは
 - 2.2 ディープラーニングとは
- 第 3 章 学習モデルの作成過程
 - 3.1 学習モデルの概要
 - 3.2 盤面の入力
 - 3.3 学習データの用意
 - 3.4 使用する学習モデルの基本パターン
 - 3.5 その他のモデル
- 第 4 章 学習結果
 - 4.1 学習済みモデルの正答率の変化とグラフ
 - 4.2 学習済みモデルの戦法ごとの予想
 - 4.3 考察
- 第 5 章 学習のためのコード
 - 5.1 csa データの読み込み
 - 5.2 並列化
 - 5.3 学習モデルの枠組みの構築
 - 5.4 学習モデルへの局面の入力
- 第 6 章 学習済みモデルとの対戦方法
- 第 7 章 まとめ
- 第 8 章 参考文献

第1章 はじめに

1.1 テーマ設定

テーマを設定した動機は大きく分けて3つある。

- ・著者が将棋好きでなつかつAIに興味があり、近年将棋AIの活躍が著しく、自分自身でも将棋AIを作ってみたいと思ったから。
- ・強いAIを作りたいから。

1.2 プログラミング言語の選択

コード作成ではプログラミング言語として、python3を扱う。

Python3を言語に選択した理由として、

- ・AIに関するライブラリーが豊富な点
- ・可読性が高い点

がある。特に、「将棋」という複雑なアルゴリズムをプログラムで表現するにあたり、可読性の高さは最も重要な点であると感じた。

逆にPython3を扱うにあたりデメリットとなる点にして実行速度の遅さがある。

第2章 機械学習の概要

2.1 機械学習とは

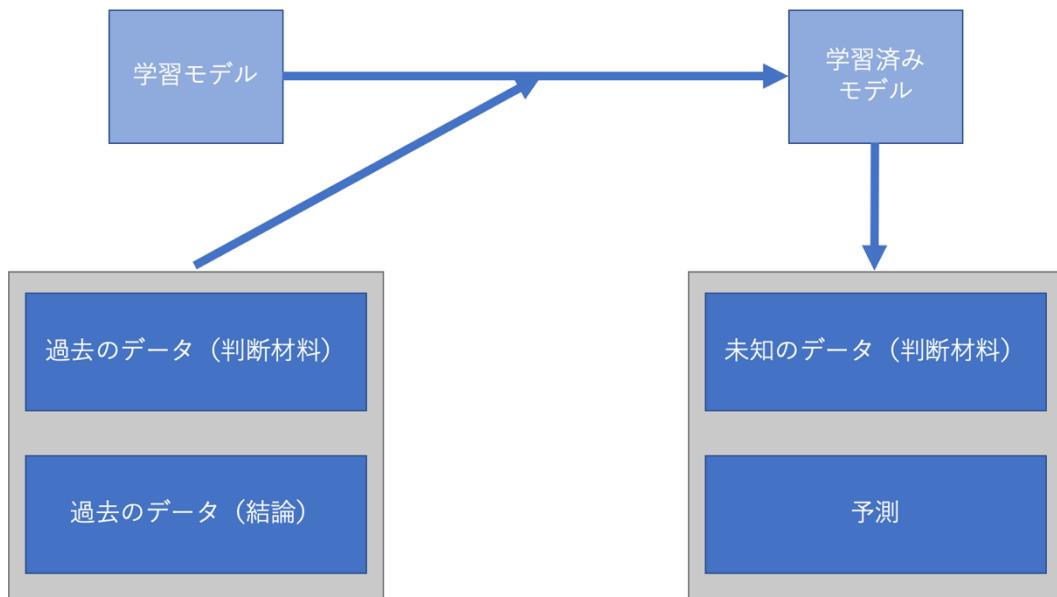
機械学習とは、コンピューターにデータを読み込ませて、規則性等を分析する手法である。



ここで「学習モデル」とは、「データから学習させる方法の枠組み」のことを示す。

学習モデルにデータを入力した結果生まれる「学習済みモデル」はデータの規則性等を学習したモデルのことである。

このモデルに対して、新しくデータを入力することによって、過去の経験から結果を予測することができる。



2.1 ディープラーニングとは

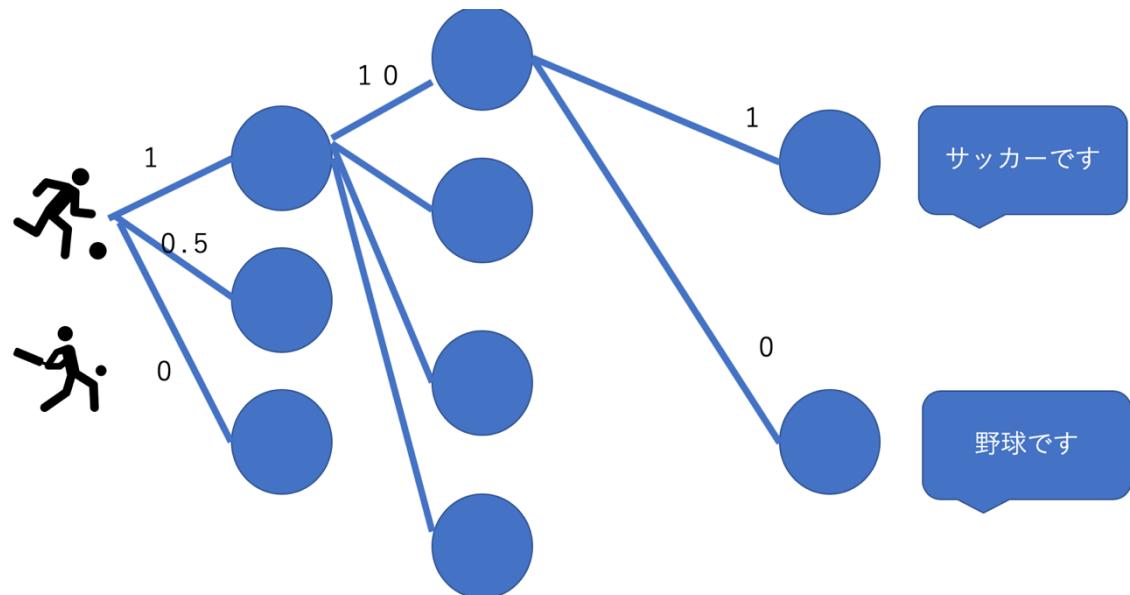
ディープラーニングとは、機械学習の手法の一種である。

特徴として

- 複雑なデータでも対応可能
 - 適切なパラメータ（予測を行うために設定する数値）を自動的に導き出せる。
- などがある。これらの特徴により
- 将棋の盤面の複雑性に対応する。
 - 「盤面のどこに重点を置くか」という問題を自動で解ける。
- といったメリットがある。

詳しい説明はここでは省くが、データの特徴を入力すると、各ノードを通って最終的な数字が出てくる。（各ノードでは、ほとんどが行列の計算を行う）この数字を正解ラベルへと寄せていくために各ノードの持っているパラメーターを調整する作業が学習である。

大量のデータに対して、反復学習を繰り返し、正解へたどり着くためのアルゴリズムを（行列の各要素を少しずつ更新することによって）変形していく。

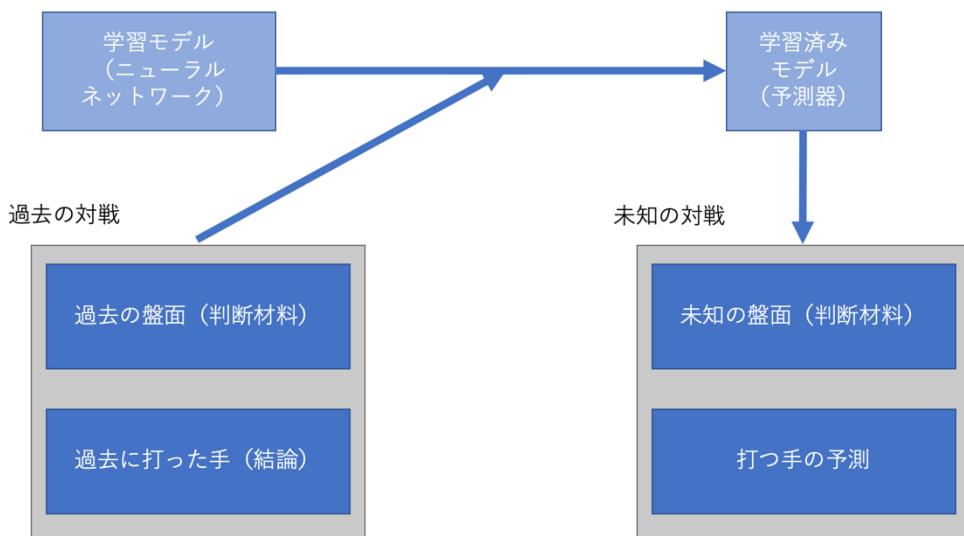


第3章 機械学習モデルの作成過程

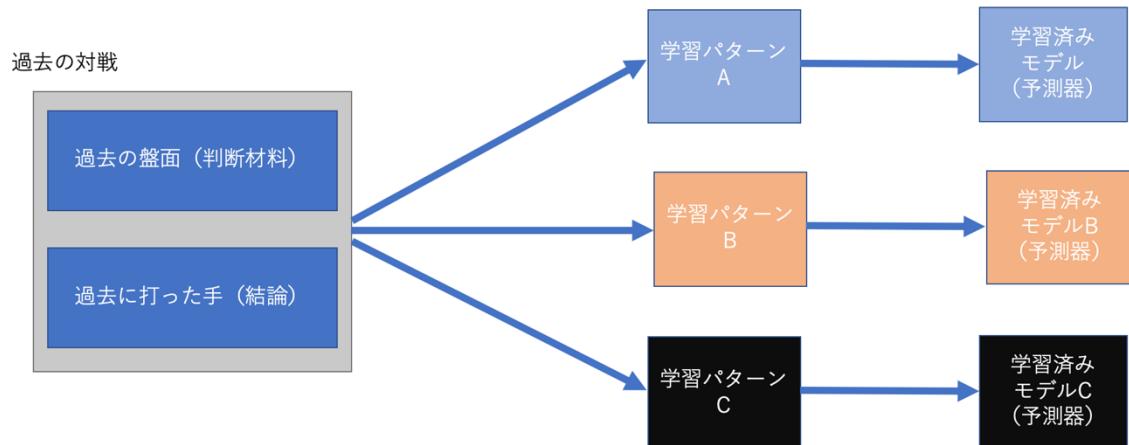
ここではどのように機械学習を行い、どのような過程で将棋AIモデルを作成してきたのかについて述べる。

3.1 学習モデルの概要

学習の流れとして、以下のように過去の盤面からディープラーニングの手法を用いて学習済みモデルを作成し、未知の盤面を入力して打つ手を予測する。



モデルの学習方法は複数用意し、それに伴い生成される学習済みモデルも複数用意する。



ここで作られた学習済みモデルに対して、以下の処理を行い、そこ結果を比較することで、学習済みモデルの調査を行う。

- ・未知の盤面の入力し、その結果を見る。
- ・既知の盤面を入力し、その結果を見る。
- ・一局を通して、序盤、中盤、終盤ごとの正答率を見る。

3.2 盤面の入力

基本方針として、ディープラーニングの中でも、画像認識と同じ方法を用いる。つまり、将棋の盤面をひとつの画像であると捉えて学習する。

画像データとは一つのドットであるピクセルの集合である。

一つのピクセル内部では赤、青、緑の3つの要素が存在する。そしてその3つそれぞれに0から255の整数値が割り当てられており、それにより赤の度合い、青の度合い、緑の度合いを表すことができる。この3つの要素を合わせることで一つのピクセルの色を表すことができる。

(つまり、今回の場合は一つのピクセルに255の3乗である16777216通りの色が当てはめられる可能性がある)

今回の学習では、この一ピクセルが将棋の盤面の一マスにあたることになり、また赤、青、緑にあたるのが、

王将、玉将、金、銀、桂馬、香車、角、飛車、歩、そしてさらに銀成、桂成、香成、と金、馬、龍である。

画像認識技術では、画像データの学習モデルへの入力方法として、
画像データを赤、青、緑の3つの二重配列に分け、
青に関する0~255の値をひとつのセルとした二重配列、
赤に関する0~255の値をひとつのセルとした二重配列、
緑に関する0~255の値をひとつのセルとした二重配列、
この3つの配列を機械学習のデータとして使う。

今回の学習では、
金に関する0~1の値をひとつのセルとした二重配列、
銀に関する0~1の値をひとつのセルとした二重配列、

桂馬に関する 0~1 の値をひとつのセルとした二重配列、

…

馬に関する 0~1 の値をひとつのセルとした二重配列、

龍に関する 0~1 の値をひとつのセルとした二重配列、

として、これらの配列を機械学習のデータとして使う。

金に関する二重配列では、金のあるマス目を 1、ない場合は 0 を要素として入れる。

3.3 学習データの用意

学習データは実際の将棋の対戦データとする。

将棋 AI の一般的な学習では、csa と呼ばれる形式で蓄積されるデジタルデータを用いる。

csa ファイルは開始局面とプレイヤーの指し手によって構成されており、

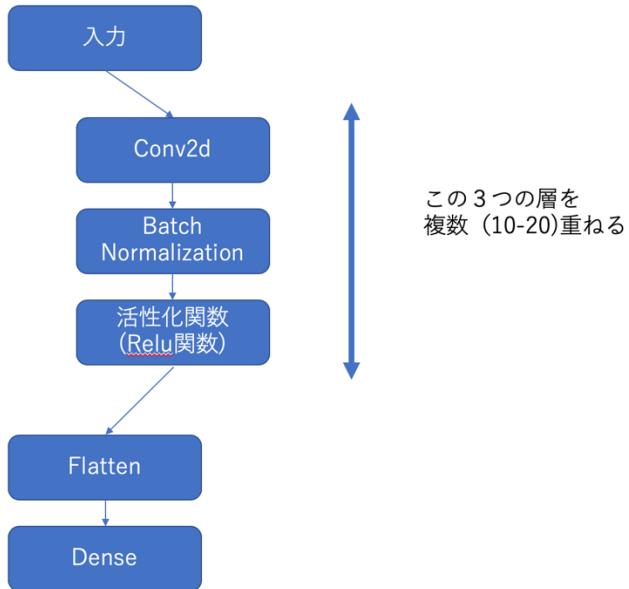
学習のためにはプログラムの中で局面をプレイヤーの指し手とともに変化させつつ、データを取り込む必要がある。

データは 2016 年度の csa ファイルを用いる。以下のサイトからダウンロードした。

<http://wdoor.c.u-tokyo.ac.jp/shogi/>

3.3 使用する学習モデル

今回はいくつかモデルを用意し、それらを比較してみるが、基本的なモデルは以下の通りである。



Conv2d は畳み込みと呼ばれる作業を行う。

通常の入力方法では、特徴量を全て一次元のベクトルに直すが、その方法では画像データなどの特徴量同士の位置関係も考慮に入れるべき時に、その情報を消してしまうことになる。

この畳み込み層では、特徴量同士の位置関係の情報を保持しながら、学習を進めることができる。

Batch Normalization はデータの正規化を行う。

活性化関数はデータを転送するかしないかを決める関数である。

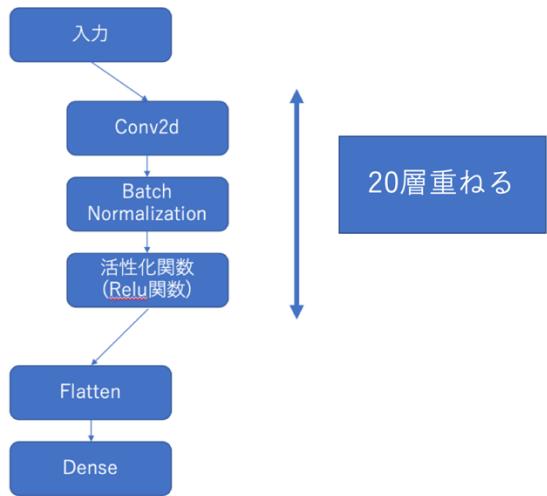
今回の Relu 関数と呼ばれる活性化関数は、送られてきたデータが 0 以上であれば、そのまま元のデータを通し、0 以下であれば、0 を返すという仕組みである。

この 3 つの層が 10 個重なった層の集合が、今回のモデルの中核である。

3.3 その他の学習モデル

学習モデルの二つ目のパターンは 3.2 で紹介した 3 つの層の集合を 20 層重ねたモデルである。

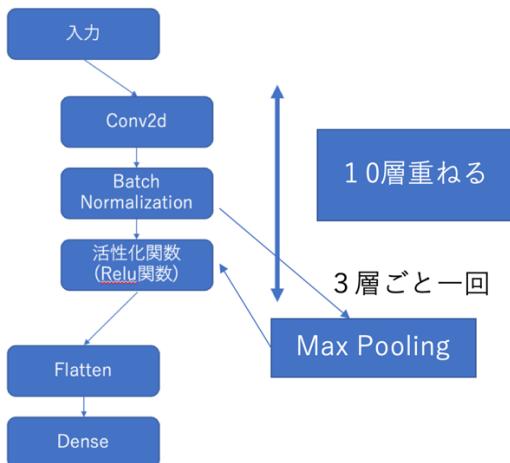
このモデルを使用する理由として、レイヤー数が増えた場合、正答率がどう上がるかを調べるためにある。



さらに、学習モデルの 3 つ目のパターンとして、「プーリング層」と呼ばれる層を追加しておく。

このプーリング層では、行列の要素の最大の値を各エリアから取り出す動きをする。画像認識では有効な層であり、この層の存在によって微細な画像ごとの位置のズレを修正することができる。

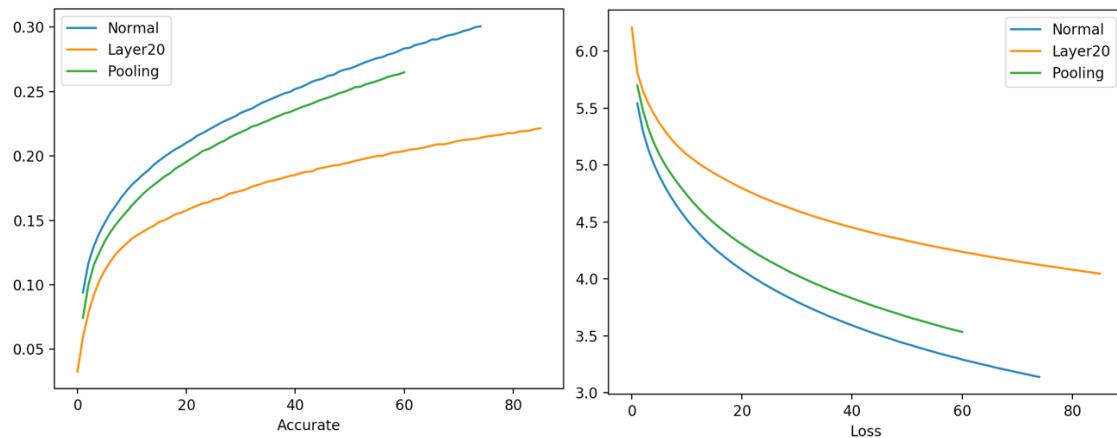
この層を加えたパターンでは、盤面の全体像をつかむ認識力が強まる予想できる。



第4章 学習結果

4.1 学習モデルの正答率の変化とグラフ

3 パターンの訓練データへの正答率



● 正答率

全体的なグラフの形状を見た限り、正答率については、レイヤー数を増やすほど正答率が低くなるという結果が出た。

総合的に、エポック数が 60 しかないのもあり、レイヤー数が多い方がパラメーターの学習に時間がかかり、学習時間が不完全であった可能性もあるが、グラフの形状から、そのまま正答率の訓練データでの順位は変わらないという見方もできる。

今回は、時間の関係上、エポック数を 60 のみにしているので 60 エポック以上の学習をした場合は訓練データでの正答率の優劣が変わる可能性があるが、少なくとも学習を始めた段階では、レイヤー数が 10 である場合の方が、20 である場合よりも高いことがわかった。

● 損失関数

教師ありの機械学習をするためには、「予測 \Rightarrow うまく予測できたかどうか \Rightarrow 予測の仕方の変更」を繰り返し、徐々に正解に寄せていく必要がある。

この「うまく予測できたかどうか」を損失関数によって評価する。

損失関数は減点方式であり、この計算結果が 0 に近づけば近づくほど、良い結果が出ていることになる。（「うまく予測できたかどうか」を正答率にしない理由は、計算の都合上である）

そして、この損失関数の結果も、正答率と同じになった。

4.2 学習済みモデルの戦法ごとの予測

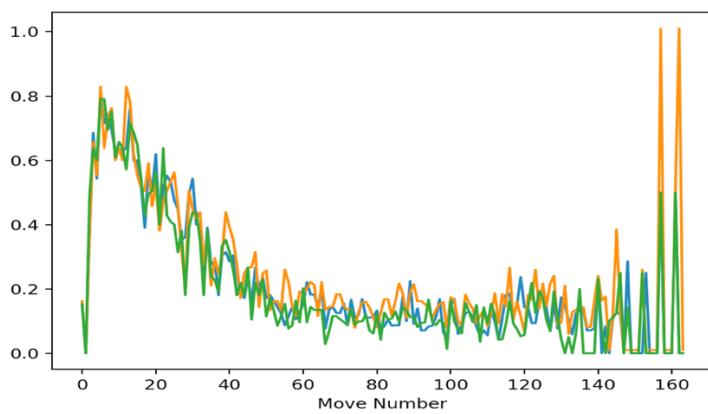
この章ではプロの棋士の棋譜を作成したモデルに入力し、実際に予想してみる。

棋譜は、<https://shogidb2.com/>

に入力した結果からそれぞれ最大 240 局抽出した。

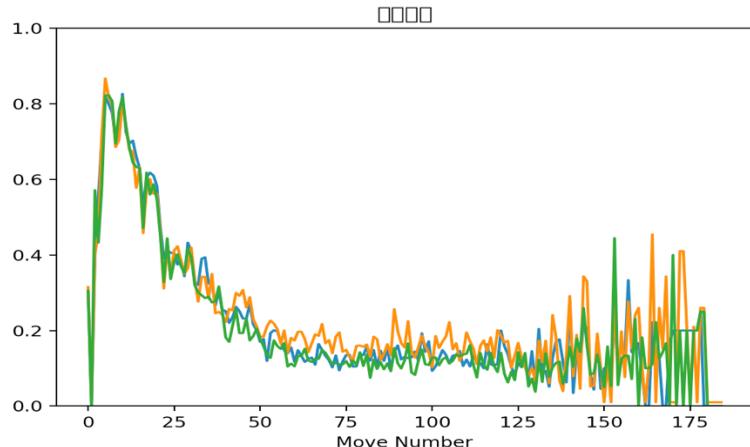
全ての棋譜の手数ごとに正答率を出している。（グラフの最後尾の振れ幅が大きいのは、最後尾まで続く棋譜数が少なく、例えば 2 局しかない場合は、0 パーセントか、50 パーセントか、100 パーセントしかないためである。）

- 加藤一二三の予想



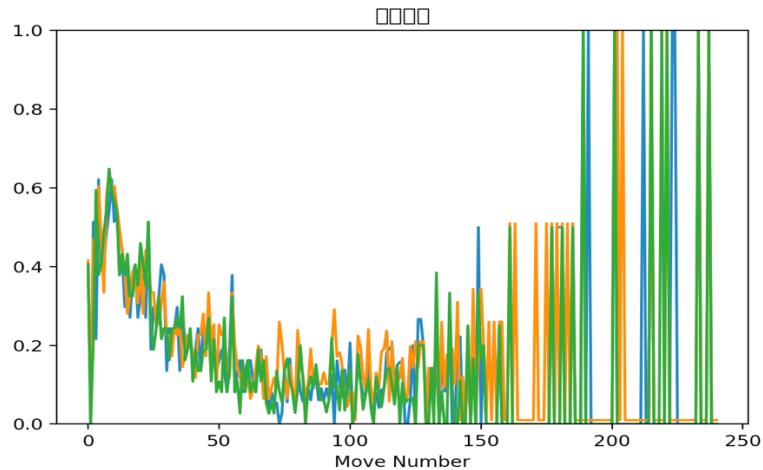
今回のモデルによる予想は、序盤の正答率は高い結果になり、さらにどのモデルも、大きい差は見られなかった。

- 羽生善治の予想



加藤一二三での予想と大きい差はないが、中盤の 50 手から 125 手にかけて、標準モデルの予測の正答率が高くなっている。（これは訓練データと同じ結果である）また、0 手から 25 手にかけて、それぞれのモデルの建てた予測精度に変化がないのが特徴的であった。

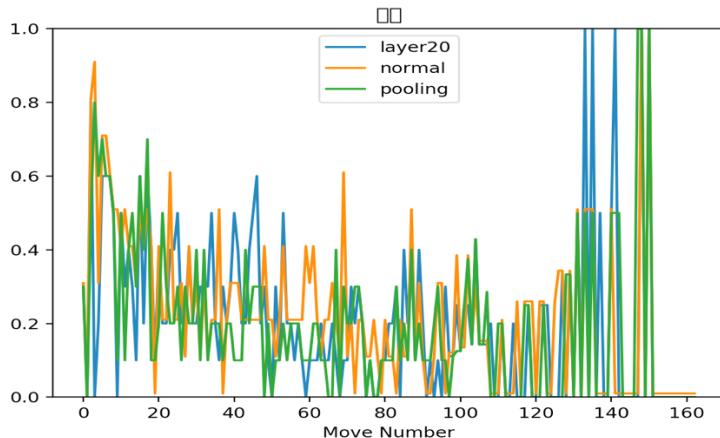
● 三間飛車の予想



3間飛車の対局は、32局しか集めることができなかつたが、印象として、序盤の正答率がどれも低いということが見受けられた。（0手から50手までで、60パーセントの正答率）

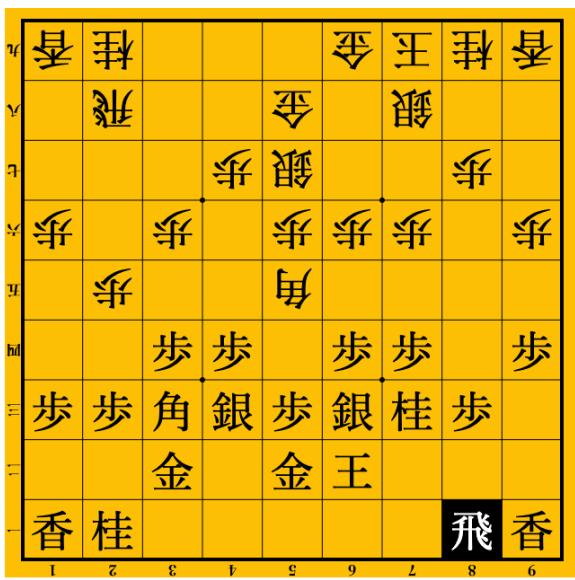
考えられる可能性として、学習データ内に三間飛車の棋譜が存在しないことが考えられる。

● 右玉



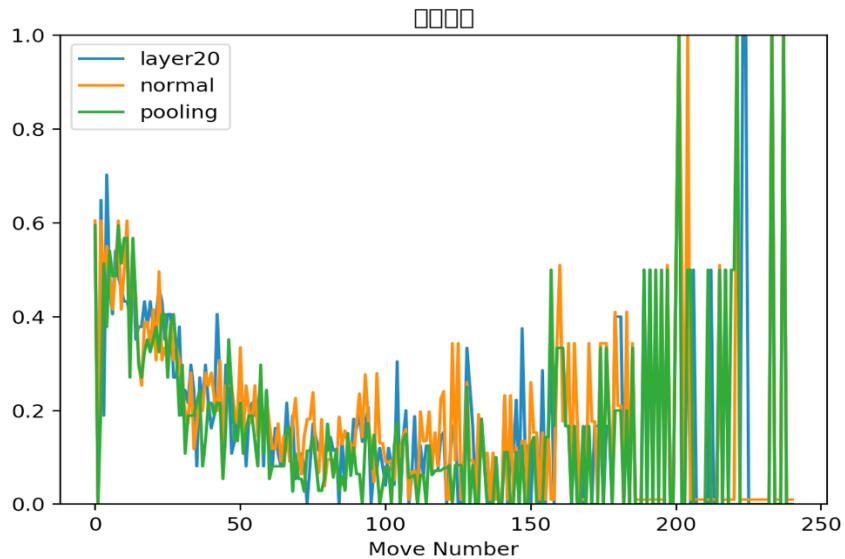
右玉の対局データは、14局しか集めることができなかつたが、全体の精度が比較的高いという、かなり特徴的な結果になった。

右玉は、かなり特殊な戦法で、下の図のように、飛車を引いて全体のバランスを保つつつ、自陣の打ち込む隙を与えない戦法である。全体のバランスがいい序盤のような状態が長く続くことにより、上記のようになったと考えられる。



2012-06-21 順位戦中川大輔 vs. 南芳一 順位戦

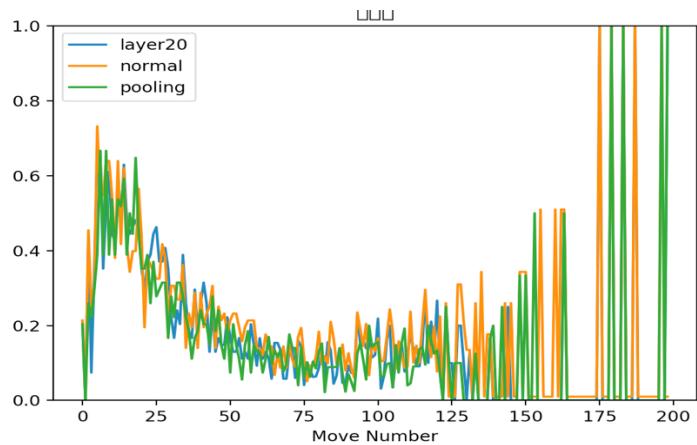
● 四間飛車



四間飛車の正答率に関しても、序盤があまり高くないのが特徴的であった。

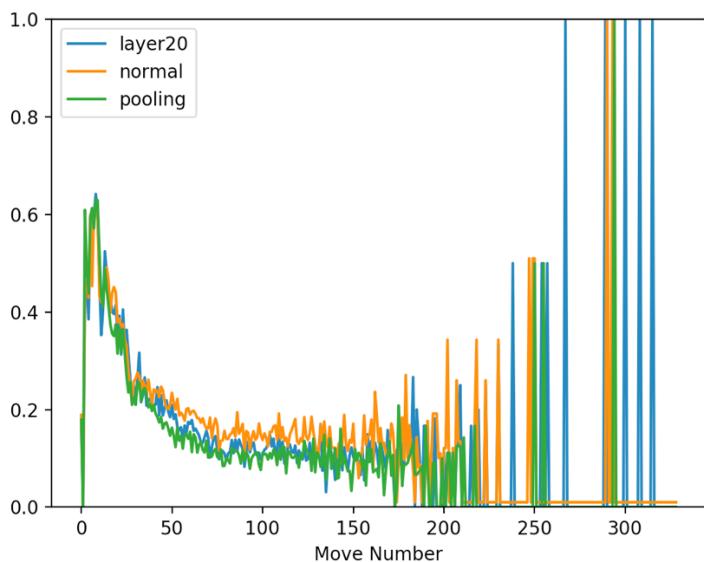
三間飛車の結果も含めて考えた場合、振り飛車での精度があまり高くないことが考えられる。モデルごとの違いについては、pooling モデルの 100 手から 150 手までの精度が他と比べて低いことが挙げられる。

● 中飛車

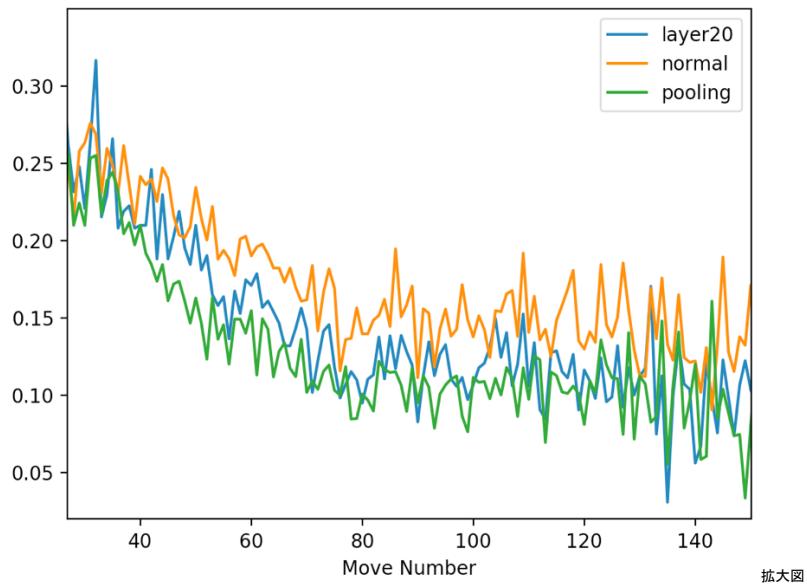


振り飛車と似た形状のグラフとなったが、振り飛車よりも僅かながら序盤の正答率は高いようであった。

● 全て(600 局)



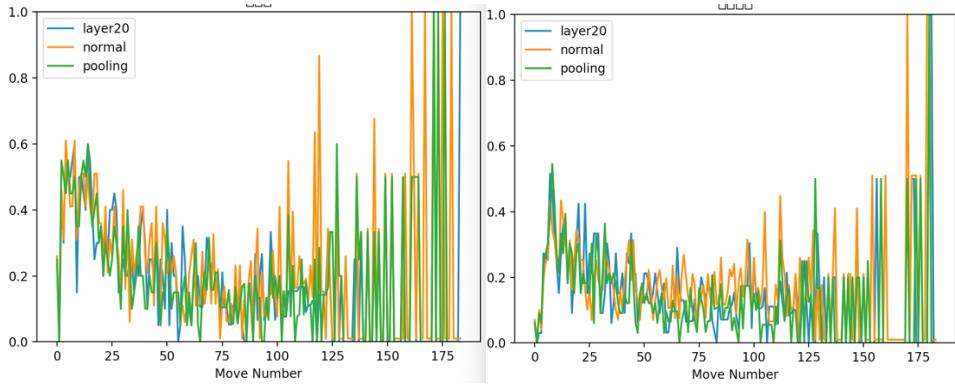
局数を増やした今回、モデルごとの中盤の精度の結果に大きな差が出た。もっとも精度の高いモデルは標準的なモデル、次に layer20 モデル、最も低い精度となったのが pooling モデルであった。



4.2 考察

今回のモデルの作成は画像認識をベースとしているため、全体のバランスを見て打つ序盤の盤面が画像としての意味合いが強く、正答率として高いと考えられる。序盤の正答率が高いと考えられるさらなる事象として、試合のほとんどが序盤のような右玉の棋譜の正答率が高いことが挙げられる。

加藤一二三は棒銀を得意とし、さらにその序盤の正答率は高かった。（棒銀は居飛車の戦法）逆に、振り飛車である、三間飛車と四間飛車の正答率は低かった。
確認のために、「振り飛車」「居飛車」と棋譜検索した棋譜の集合を比べてみた。



左図は居飛車、右図は振り飛車

予想通り序盤の正答率は、居飛車の方が高い。

複雑な終盤の正答率が 0 ではないことが序盤の正答率から考えた時、予想外の結果となつた。

これは、玉の位置関係をもとに、終局するための手を打つのではないかと考えられる。（画像認識という観点で考えた場合、玉の周囲の位置関係のみが、終局に近いかどうかの判断材料になるため）

モデルごとの差については、棋譜の数を増やした（600局）場合にはっきりとしたが、10層のNormal モデルが最も高く、次に 20 層のモデル、最後に Pooling 層のモデルの順であった。

10 層のモデルが 20 層のモデルより正答率が高い理由として、学習が不十分で、20 層分のパラメーターをきちんと設定できなかった可能性がある。

Pooling 層は、画像認識の中で、画像をぼやけさせる効果があるが、精密な手を必要とする将棋では、逆効果であった。

第5章 学習のためのコード

2.3 ファイルから学習データへの変換コード

まず、将棋の棋譜を読み込む

DataStore.py

```
def open_csafile(csa_filepath):
    csa_text = ""
    with open(csa_filepath, encoding='utf-8') as f:
        csa_text = f.read()
    return shogi.CSA.Parser.parse_str(csa_text)[0]
```

次に読み込んだテキストから、指し手のリスト(moves)を作成し、局面(board)を指し手(move)によって動かす。

動かす前の局面(board)とその局面でどう動かしたか(move)をひとつのデータとして取り出す。

DataStore.py

```
def runBoard( kifu_text):
    moves = kifu_text['moves']
    board = shogi.Board()
    board.clear_kifu_str = kifu_text
    for usi in moves:
        #ボードの状態とどう動かすかを返す
        yield board, usi
        board.push_usi(usi)

    del moves
    del board
```

open_csafile によって読み込んだテキストを runBoard へ代入し、特徴量を取得する関数である grep_X と grep_Y (後述) により、特徴量を取得する。

DataStore.py

```
def csa_text_list_to_X_y(csa_text_list):
    X = []
    y = []
    for csa_text in csa_text_list:
        for board, usi in runBoard(csa_text):
            X.append( GrepFeature.grep_X(board) )
            move = shogi.Move.from_usi(usi)
            y.append( GrepFeature.grep_y(move,board.turn) )

    return X,y
```

grep_X では、board という局面のデータを受け取り、特徴量ベクトルを作成する。
grep_Y では、board 内の turn にあるプレイヤーターンの情報と、move という駒の動きのデータを受け取り、クラスラベルを割り当てる。

2.4 学習モデルへの通常のデータの入力方法について

ライブラリーである Keras を使った学習モデルへのデータの入力は通常は以下のように行う。

```
model.fit(  
    <学習データの特徴ベクトル>, <学習データの答えのラベル>,  
    batch_size=self.batch_size,  
    epochs=self.epochs,  
    verbose=1,  
)
```

上の model は Keras ライブラリを利用することによって構築された学習モデルである。

今回は、`Sequential()` のインスタンスである。

この fit と呼ばれるメソッドを実行した時に、学習が始まる。

この方法のメリットの一つとして、学習データの入力が明確であることがある。学習データの特徴ベクトルとそれに伴う学習データのクラスラベルを入力することで、学習を簡単に行うことができる。

しかし、メモリの使用に限界があるような場面では、実行することができない場面もある。

今回の場合、棋譜とそれにより作られる局面のデータを画像認識と同じ方法で処理するので、多くのメモリを使うことになり、実行することができない。

そこで今回は、別の方法を用いる。

2.4 Sequence を用いたデータの入力方法について

今回は、棋譜の入力と、学習を同時に使う方法を用いる。

つまり、

「1 エポックの学習の中で、Csa 形式のテキストから局面を動かしつつ入力データへ変形し、そのデータを学習モデルへ入力し、学習する」
という処理を繰り返す形で、学習を進める。

それを実現するためのモジュールとして、Keras のモジュールから Sequence クラスを利用する。

```

class MySequence(Sequence):

    def __init__(self, folder, batch_size, start_index, end_index):
        self.csa_text_list = collect_csa_text(folder, start_index, end_index)
        self.batch_size = batch_size
        self.length = math.ceil(len(self.csa_text_list) / self.batch_size)

    def __getitem__(self, idx):
        start_idx = idx * self.batch_size
        last_idx = start_idx + self.batch_size

        csa_text_list_OneBatch = self.csa_text_list[start_idx:last_idx]

        X, y = csa_text_list__to__X_y(csa_text_list_OneBatch)
        y = yFormatter(y)
        X = XFormatter(X)

        img_rows = 9
        img_cols = 9

        return X, y

    def __len__(self):
        return self.length

```

Sequence クラスを継承した MySequence クラスを使う。
`__getitem__` メソッドにより、バッチサイズ分だけデータを取得し、それを学習モデルへ入力する。

```

    self.csv_logger = keras.callbacks.CSVLogger(self.keras_log_file,
separator=',', append=False)
    self.checkpoint =
keras.callbacks.ModelCheckpoint(filepath=os.path.join(self.weights_file_path,
"model-{epoch:02d}.h5"), save_best_only=False) # 精度が向上した場合のみ保
存する。

    self.model.fit_generator(
        generator=self.mnist_sequence,
        workers = self.cpu_count,
        epochs=self.epoch_num,
        verbose=2,
        use_multiprocessing = True,
        callbacks = [self.checkpoint, self.csv_logger]
    )

```

MySequence クラスを利用した学習には fit_generator メソッドを利用する。

Fit_generator メソッドと sequence クラスを、利用することにより、データの入力と学習を同時にを行うことができる。

2.5 学習モデル

Keras では、 Sequential というクラスのインスタンスにレイヤーを追加(add メソッド)する形でモデルを構築する。

Conv2D は 2 次元の畳み込みレイヤーである。

引数の一つ目は関数内部では filters という名前があり、今回は 104 に設定している。

これは

引数の二つ目は関数内部では kernel_size という名前で、畳み込みの際に使用されるウインドウの大きさを指定する。

BatchNormalization は平均を 0

前の層の出力を平均は 0、標準偏差は 1 にするレイヤーである。

Activation は活性化関数で、今回は relu 関数を使用する。

以下、モデルの設計をするコード

```
self.model = Sequential()
for i in range(self.layer_num):
    self.model.add(Conv2D(104, (3, 3), input_shape= self.input_shape,
padding='same'))
    self.model.add(BatchNormalization(axis=1))
    self.model.add(Activation('relu'))
self.model.add(Flatten())
self.model.add(Dense(self.classes_num, activation='softmax'))

if (self.load_weight_path):
    self.model.load_weights(self.load_weight_path)

sgd = keras.optimizers.SGD(lr = 0.1, clipnorm=1.)
self.model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy'])
```

第6章 学習モデルとの対戦方法

学習したデータとの対戦方法として、将棋 GUI ソフト「将棋所」を利用する。

このソフトにより、自作したモデルとの対局を行うことができる。



以下はソフトとの通信を可能にするためのコード

```
def main():
    ct, my_color, board = WaitingConnection(ip_adress = "192.168.11.9")

    next_move_choice = NextMoveChoice(Model)

    while True:
        if board.is_checkmate():
            exit()

        next_move = next_move_choice.Conv2D_2(board)
        board.push(next_move)

        board = SendPCMoveAndRecievePlayerAction(ct, my_color, next_move,
board)
```

```
def WaitingConnection(ip_adress = "192.168.56.1", username="mine",
password="toor"):
```

```
    try:
        ct = CSA.TCPProtocol(ip_adress, 4081)
        ct.login(username, password)
        game_summary = ct.wait_match()
        sfen = game_summary['summary']['sfen']
        my_color = game_summary['my_color']
        board = shogi.Board(sfen)
        ct.agree()
    except UnboundLocalError:
        print('open shogi soft')
        exit()
    return ct, my_color, board
```

```
def SendPCMoveAndRecievePlayerAction(ct, my_color, next_move, board,):
```

```
    recieved_cmd = ct.move(board.pieces[next_move.to_square], my_color,
next_move)
    (_, usi, spend_time, message) = ct.parse_server_message(recieved_cmd,
```

```
board)
    move = shogi.Move.from_usi(usi)
    board.push(move)
    return board
```

第7章　まとめ

今回の将棋 AI 作成では、画像認識の要領で機械学習を進めたが、その中でいくつかわかったこととして、

- ・定石が固まった序盤の認識は有効だが、複雑になる終盤の画像認識は難易度が上がる。
- ・Pooling 層のある画像認識では、正確な予想が難しい。
- ・レイヤー数を多くしても、精度が上がるとは限らない。

等がある。

また、将棋 AI を作成するにあたって、学習に時間がかかり、いかにコンピューターの計算の速度が重要であるか、「なぜ計算数を減らすことにこだわる必要があるのか」を実感することができた。

今回は時間の関係上、十分な学習数を回すことができず、当初の「強い AI」を作るところまではたどり着けなかったものの、そのための足がかりとすることができた。

もし再び将棋 AI の作成や、画像認識技術に携わることがある場合、今回の失敗の経験や、成功の体験を生かしていきたい。

第8章 参考文献

棋譜検索サイト <https://shogidb2.com/>

GUI ソフト将棋所 <http://shogidokoro.starfree.jp/>

将棋用純正 python ライブライ <https://github.com/gunyarakun/python-shogi>

斎藤康毅 著 「ゼロから学ぶ Deep Learning ~Python で学ぶディープラーニングの理論と実際~」 オライリージャパン出版 2016年9月

山岡 忠夫 著 「将棋 AI で学ぶディープラーニング」 マイナビ出版 2018年3月