# 第 33 回世界コンピュータ将棋選手権
# dlshogi with HEROZ アピール文章

山岡忠夫
加納邦彦
大森悠平
2023/3/26

※下線部分は、第 32 回世界コンピュータ将棋選手権からの差分を示す。

## 1　dlshogi のアピールポイント

dlshogi は、ディープラーニングを使用した将棋 AI である。

2017 年より、AlphaGo の手法を参考に開発を行っている。

ディープラーニング系の将棋 AI は、大局観に優れており、序中盤の形勢判断が従来型将棋 AI と比べて正確であるという特徴がある。一方、終盤の読みが重要になる局面では、従来型将棋 AI の方が正確な場合がある。

dlshogi は、終盤の課題に対処するために、独自の工夫を行っている。

具体的には、「MCTS の葉ノードでの短手数の詰み探索」、「ルート局面で df-pn による長手数の詰み探索」、「勝敗が確定したノードのゲーム木への伝播」、「PV 上の局面に対する長手数の詰み探索」、「強化学習時に初期局面集を使用して局面の多様性を確保する」、「強化学習時に df-pn により詰み探索を行い詰みを報酬とする」という工夫を行っている。

これらのいくつかは、現在、dlshogi 以外のディープラーニング系の将棋 AI には取り入れられているが、dlshogi 以前にこれらを導入しているディープラーニング系の将棋 AI はなかった。

今大会に向けては、モデルサイズを 30 ブロック 384 フィルタにし、モデル精度の改善を行った。

## 2　チーム参加について

今大会では、HEROZ チームとして参加する。

## 3　dlshogi の特徴

- ディープラーニングを使用
- 指し手を予測する Policy Network
- 局面の勝率を予測する Value Network
- 入力特徴にドメイン知識を積極的に活用
- モンテカルロ木探索
- 未探索のノードの価値に親ノードの価値を使用

- GPU によるバッチ処理に適した並列化
- 自己対局による強化学習
- 詰み探索結果を報酬とした強化学習
- 既存プログラムを加えたリーグ戦による強化学習
- 既存将棋プログラムの自己対局データを混ぜて学習
- 既存将棋プログラムの自己対局データを使った事前学習
- ブートストラップ法による Value Network の学習
- 引き分けも含めた学習
- 指し手の確率分布を学習
- 同一局面を平均化して学習
- 評価値の補正
- SWA(Stochastic Weight Averaging)
- 末端ノードでの短手順の詰み探索
- ルートノードでの df-pn による長手順の詰み探索
- 勝敗が確定したノードのゲーム木への確実な伝播
- PV 上の局面に対する長手数の詰み探索
- 序盤局面の事前探索（定跡化）
- 定跡作成時に floodgate の棋譜の統計を利用した確率分布を方策に利用
- マルチ GPU 対応（NVIDIA A100×8 を使用予定）
- TensorRT を使用
- Optuna による探索パラメータの最適化
- 確率的な Ponder
- ノードのガベージコレクションとノード再利用処理の改良
- 飛車と角の利きのビット演算
- 2 値の入力特徴量を 1 ビットで転送することで推論のスループットを向上
- Stochastic Multi Ponder

## 4　使用ライブラリ

- Apery[1]（WCSC28）
  →局面管理、合法手生成のために使用

### 4.1　ライブラリの選定理由

本プログラムは、将棋におけるディープラーニングの適用を検証することを目的としており、学習局面生成、局面管理、合法手生成については、使用可能なオープンソースがあれば使用する方針である。そのため、学習局面を圧縮形式(hcpe)で生成する機能を備えていて、合法手生成を高速に行える Apery を選定した。

[1] https://github.com/HiraokaTakuya/apery

## 5　各特長の具体的な詳細（独自性のアピール）

### 5.1　ディープラーニングを使用

DNN(Deep Neural Network)と MCTS を使用して指し手を生成する。

従来の探索アルゴリズム(α β法)、評価関数(3 駒関係)は使用していない。

### 5.2　Policy Network

局面の遷移確率を Policy Network を使用して計算する。

Policy Network の構成には、Residual Network を使用した。

入力の畳み込み 1 層と、ResNet 30 ブロック(畳み込み 2 層で構成)と出力層の合計 62 の畳み込み層で構成した。フィルターサイズは 3（入力層の持ち駒のチャンネルのみ 1)、フィルター数は 384 とした。

### 5.3　Value Network

局面の勝率を Value Network を使用して計算する。

Value Network は、Policy Network と出力層以外同じ構成で、出力層に全結合層をつなげ、シグモイド関数で勝率を出力する。

### 5.4　入力特徴にドメイン知識を積極的に活用

Alpha Zero では、入力特徴に呼吸点のような囲碁の知識を用いずに盤面の石の配置と履歴局面のみを入力特徴とすることで、ドメイン知識なしでも人間を上回ることが示された。しかし、その代償として、入力特徴にドメイン知識を活用した AlphaGo Lee/Master に比べて倍のネットワークの層数が必要になっている。AlphaGo Zero の論文の Figure 3 によると、ネットワーク層数が同一のバージョンでは Master を上回る前にレーティングが飽和している。

強い将棋ソフトを作るという目的であれば、積極的にドメイン知識を活用した方が計算リソースを省力化できると考えられる。

そのため、本ソフトでは、入力特徴に盤面の駒の配置の他に、利き数と王手がかかっているかという情報を加えている。それらの特徴量が学習時間を短縮する上で、有効であることは実験によって確かめている。

### 5.5　モンテカルロ木探索

対局時の指し手生成には、Policy Network と Value Network を活用したモンテカルロ木探索を使用する。

ノードを選択する方策に、Policy Network による遷移確率をボーナス項に使用した PUCT

アルゴリズムを使用する。PUCT アルゴリズムは、AlphaZero の論文[2]の疑似コードに記述された式を使用した。

また、末端ノードでの価値の評価に、Value Network で計算した勝率を使用する。

通常のモンテカルロ木探索では、末端ノードから複数回終局までプレイアウトを行った結果（勝率）を報酬とするが、将棋でランダムなプレイアウトは有効ではないため、プレイアウトを行わず Value Network の値を使用する。

### 5.6　未探索のノードの価値に親ノードの価値を使用

モンテカルロ木探索の UCB の計算時に、未探索の子ノードがある場合、そのノードの価値に何らかの初期値を与える必要がある。子ノードの価値は親ノードの価値に近いだろうという将棋のドメイン知識を利用し、それまでの探索で見積もった親のノードの価値を動的に初期値として使用する。ただし、ノードの訪問回数が増えるに従い、その価値の減衰を行い、幅より深さを優先した探索を行う(FPU reduction)。

### 5.7　GPU によるバッチ処理に適した並列化

複数回のシミュレーションを順番に実行した後、それぞれのシミュレーションの末端ノードの評価をまとめて GPU でバッチ処理する。その後、評価結果をそれぞれのシミュレーションが辿ったノードにバックアップする。以上を一つのスレッドで行うことで、マルチスレッドによる実装で課題となる GPU の計算後にスレッドが再開する際にリソース競合が起きる問題（大群の問題）を回避する。

GPU で計算中は、CPU が空くため、同じ処理を行うスレッドをもう一つ並列で実行する。2 つのスレッドが相互に CPU と GPU を利用するため、利用効率が高い処理が可能となる。

### 5.8　自己対局による強化学習

事前学習を行ったモデルから開始して、AlphaZero[3]と同様の方式で強化学習を行う。自己対局により教師局面を生成し、その教師局面を学習したモデルで、再び教師局面を生成するというサイクルを繰り返すことでモデルを成長させる。

2018 年の大会で使用した elmo で生成した教師局面で収束するまで学習したモデルに比べて、自己対局による強化学習によって有意に強くすることができた。

### 5.9　詰み探索結果を報酬とした強化学習

自己対局時に終局まで対局を行うと、モンテカルロ木探索の特性上、詰むまでの手順が長くなる傾向がある。勝率予測に一定の閾値を設けることで、終局する前に勝敗を判定することで対局時間を短縮できるが、モデルの精度が低いうちは誤差が大きいため、学習精度に影

[2] http://science.sciencemag.org/content/362/6419/1140
[3] https://arxiv.org/abs/1712.01815

響する。

この課題の対策として、df-pn による高速な長手数の詰み探索の結果を報酬とした。単純にすべての局面で詰み探索を行うと、自己対局の実行速度が大幅に落ちてしまう。自己対局は複数エージェントに並列で対局を行わせ、各エージェントからの詰み探索の要求をキューに溜めて、詰み探索専用スレッドで処理するようにした。エージェントが GPU の計算待ちの間に詰み探索が完了する。エージェントが探索している局面は別々のため、時間のかかる詰み探索の要求が集中することは少ない。これにより自己対局の速度を大幅に落とすことなく長手数の詰み探索を行えるようになった。

### 5.10 既存プログラムを加えたリーグ戦による強化学習

自分自身のプログラムのみで強化学習を行うと戦略に弱点が生まれる可能性がある。弱点をふさぐには多様なプログラムによるリーグ戦が有効だが、複数のエージェントを学習するにはエージェント数の分だけ余分に計算資源が必要になる。

計算資源を省力化して、リーグ戦の効果を得るために、オープンソースで公開されている既存の将棋プログラムを 1/8 の割合でリーグに加えて強化学習を行うようにした。

### 5.11 既存将棋プログラムの自己対局データを使った事前学習

本プログラムを使用して、Alpha Zero と同様に、ランダムに初期化されたモデルから強化学習を行うことも可能だが、使用可能なマシンリソースが足りないため、スクラッチからの学習は行わず、既存将棋プログラムの自己対局データを教師データとして、教師あり学習でモデルの事前学習を行う。

教師データには、elmo で生成した自己対局データを使用した。

### 5.12 既存将棋プログラムの自己対局データを混ぜて学習

以前の dlshogi は、入玉宣言勝ちできる局面でなかなか入玉宣言勝ちを目指さないという課題があった。

自己対局では入玉宣言勝ちの棋譜が少ないため、それを補うため既存将棋プログラム(水匠)の自己対局で、入玉宣言勝ちの棋譜を生成し、dlshogi の自己対局のデータに混ぜて学習した。

### 5.13 ブートストラップ法による Value Network の学習

Value Network の学習の損失関数は、勝敗を教師データとした交差エントロピーと、探索結果の評価値を教師データとした交差エントロピーの和とした。

このように、本来の報酬（勝敗）とは別の推定量（探索結果の評価値）を用いてパラメータを更新する手法をブートストラップという。

経験的にブートストラップ手法は、非ブートストラップ手法より性能が良いことが知られている。

### 5.14 引き分けも含めた学習

将棋はルールに引き分けがあるゲームであるため、引き分けも正しく学習できる方が望ましい。そのため、自己対局で引き分けとなった対局も学習データに含めて学習した。

### 5.15 指し手の確率分布を学習

以前の dlshogi では、指し手のみを学習していたが、AlphaZero と同様に、自己対局時で MCTS で探索した際のルート局面の子ノードの訪問回数に従った確率分布を学習するように変更した。確率分布を学習することで、最善手と次善手の行動価値が近い場合に、次善手の行動価値を正しく学習できるようになる。

確率分布を学習することで、floodgate の棋譜に対する一致率が向上することが確認できたが、対局して強さを計測すると弱くなるという現象が確認できた。原因は、モデルの方策の出力の性質が変わるため、探索パラメータの調整が必要なためであった。Optuna を使用して探索パラメータを最適化(5.27 参照)することで、指し手のみを学習したモデルよりも強くすることができた。

### 5.16 同一局面を平均化して学習

自己対局では、序盤の同一の局面の教師データが多く生成される。それらの重複した局面を別のサンプルとして学習すると、モデルの学習に偏りが起きる。

局面の偏りをなくするために、同一の局面を集約し、指し手の確率分布と勝敗を平均化し、1 サンプルとして学習した。

### 5.17 評価値の補正

自己対局で生成するデータには、MCTS で探索して得られた勝率(最善手の価値)を局面の評価値を記録し、学習時にブートストラップ項(5.13 参照)として使用している。記録した評価値(勝率)が、実際の対局の結果から算出した勝率と一致しているか調べたところ、乖離しているという現象が確認できた。そのため、評価値を実際の自己対局での勝率に合うように、補正を行った。

### 5.18 SWA(Stochastic Weight Averaging)

画像認識の分野でエラー率の低減が報告されている手法である、SWA(Stochastic Weight Averaging)をニューラルネットワークの学習に取り入れた。一般的なアンサンブル手法では、推論結果の結果を平均化するが、SWA では学習時に一定間隔で重みを平均化することでアンサンブルの効果を実現する。

### 5.19 末端ノードでの短手順の詰み探索

モンテカルロ木探索の末端ノードで、5 手の詰み探索を行い、詰みの局面を正しく評価で

きるようする。並列化の方式により、GPU で計算中の CPU が空いた時間に詰み探索を行うため、探索速度が落ちることはない。

### 5.20 ルートノードでの df-pn による長手数の詰み探索

モンテカルロ木探索は最善手よりも安全な手を選ぶ傾向があるため詰みのある局面で駒得になるような手を選ぶことがある。

対策として、詰み探索を専用スレッドで行い、詰みが見つかった場合はその手を指すようにする。

詰み探索は、df-pn アルゴリズムを使って実装した。優越関係、証明駒、反証駒、先端ノードでの 3 手詰めルーチンにより高速化を行っている。

### 5.21 勝敗が確定したノードのゲーム木への確実な伝播

モンテカルロ木探索で構築したゲーム木の末端ノードで詰みが見つかった場合、その結果をゲーム木に伝播して利用する。つまり、モンテカルロ木探索に、AND/OR 木の探索を組み合わせて、詰みの結果を確実にゲーム木に伝播するようにする。

### 5.22 PV 上の局面に対する長手数の詰み探索

ディープラーニング系の将棋 AI は、選択的に探索を行うために、終盤の局面で読み抜けがあると、頓死することある。

頓死を防ぐため、PV 上の局面に対して、df-pn による長手数の詰み探索を行い、詰みが見つかった場合、局面の価値を更新するようにする。

### 5.23 序盤局面の事前探索（定跡化）

出現頻度の高い序盤局面は、対局時に探索しなくても、事前に探索を行い定跡化しておくことができる。また、事前に探索することで、対局時よりも探索に時間をかけることができる。

ゲーム木は指数関数的に広がるため、固定の手数までの定跡を作成するよりも、有望な手順を選択的に定跡に追加する方が良い。自分が指す手は、1 つ局面につき最善手を 1 手（または数手）登録し、それに対する応手は、公開されている定跡や棋譜の統計情報を使って確率的に選択する。その手に対して、また最善手を 1 手（または数手）登録する。この手順により、頻度の高い局面については深い手順まで、頻度の低い局面については短い手順の定跡を作成することができる。

### 5.24 定跡作成時に floodgate の棋譜の統計を利用した確率分布を方策に利用

定跡を自分自身の探索のみで作成した場合、読み抜けがあった場合に定跡を抜けた後に不利な局面になる恐れがある。そのため、モンテカルロ木探索の PUCT の計算で、方策ネット

ワークの確率分布と floodgate の棋譜の統計を利用した確率分布を平均化した確率分布を利用し、致命的な読み抜けを防止する。

### 5.25 マルチ GPU 対応

複数枚の GPU を使いニューラルネットワークの推論を分散処理する。

「5.7 GPU によるバッチ処理に適した並列化」の方式により、GPU ごとに 2 つの探索スレッドを割り当てることで、GPU を増やすことでスケールアウトすることができる。ノードの情報は、すべてのスレッドで共有する。

確認できている範囲で 4GPU までほぼ線形で探索速度を上げることができている。

### 5.26 TensorRT を使用

モデルの学習にはディープラーニングフレームワークとして PyTorch を使用しているが、対局プログラムには、推論用ライブラリの TensorRT を使用する。

TensorRT を使うことで、事前にレイヤー融合などのニューラルネットワークの最適化を行うことで、推論を高速化することができる。TensorCore に最適化されており、TensorCore を搭載した GPU では CUDA+cuDNN で推論を行う場合より、約 1.33 倍の高速化が可能になる[4]。

また、対局の実行環境にディープラーニングフレームワークの環境構築を不要とすることを目的とする。

### 5.27 Optuna による探索パラメータの最適化

PFN により公開された Optuna[5]を使用して、モンテカルロ木探索の探索パラメータ（PUCT の定数、方策の温度パラメータ）を最適化した。

Optuna は、主にニューラネットワークの学習のハイパーパラメータを最適化する目的で利用されるが、将棋エンジン同士の連続対局の勝率を目的関数として、探索パラメータの最適化に使えるようにするスクリプト[6]を開発した。Optuna の枝刈り機能により、少ない対局数で収束させることができる。

### 5.28 確率的な Ponder

モンテカルロ木探索は確率的にゲーム木を成長させる。その特性を活かして、相手が思考中に、相手局面からモンテカルロ木探索を行うことで、確率的に相手の手を予測して探索を行うことができる。予測手 1 手のみを Ponder の対象とするよりも、効率のよい Ponder が実

---

[4] https://tadaoyamaoka.hatenablog.com/entry/2020/04/19/120726

[5] https://optuna.org/

[6] https://github.com/TadaoYamaoka/DeepLearningShogi/blob/master/utils/mcts_params_optimizer.py

現できる。

### 5.29 ノードのガベージコレクションとノード再利用処理の改良

世界コンピュータ将棋オンライン大会でノード再利用に 10 秒以上かかる場合があるこがわかったため、ノード再利用の方式の見直しを行った。

以前は、オープンアドレス法でハッシュ管理を行っており、ルートノードから辿ることができないノードをすべてのハッシュエントリに対して線形探索してノードの削除をおこなっていた。

これを、Leela Chess Zero のゲーム木の管理方法[7]を参考に、ゲーム木をツリーで管理を行うようにし、ルートの兄弟ノードをガベージコレクションする方式に変更した。ノードの合流の処理が行えなくなるという欠点があるが、ノード再利用を短い時間で行えるようになった。

### 5.30 飛車と角の利きのビット演算

第 31 回世界コンピュータ将棋選手権の Qugiy のアピール文章[8]による、飛車、角の利きをビット演算により求める方法を実装した（実装はやねうら王のソースコードを参考にした)。ZEN2 の CPU で NPS が約 1% 向上した。

### 5.31 2 値の入力特徴量を 1 ビットで転送することで推論のスループットを向上

マルチ GPU を使用した場合、4GPU 以上では CPU と GPU 間の帯域がボトルネックになるため、2 値の入力特徴量を float の代わりに、1bit で表現し、GPU にビットで転送後、GPU 側で CUDA のプログラムでバッチ単位に並列に float に戻す処理を実装した。こうすることで、転送量が削減でき、NPS が 36.6%向上した。

### 5.32 Stochastic Multi Ponder

相手番に、相手番の局面から探索を行う Stochastic Ponder（5.28 参照）と、次の相手の指し手を複数予測し、並列に分散して探索を行う Multi Ponder を組み合わせて使う。

shotgun で実装されていた Multi Ponder[9]では、技巧 2 の Multi PV の結果を利用しているが、dlshogi の Stochastic Ponder では、ほとんどの場合、相手局面でのゲーム木が展開済みであり、ルートの子ノードの訪問回数を参照することで、有望な予測手を N 手取得することができる（ゲーム木が未展開の場合は、方策ネットワークの推論結果を使用する)。

また、予測した N 手以外の手が指された場合、Stochastic Ponder でも並列に探索を行っているため、Multi Ponder を使用しない場合と遜色のない手を指すことができる。

---

[7] https://tadaoyamaoka.hatenablog.com/entry/2020/05/05/181849
[8] https://www.apply.computer-shogi.org/wcsc31/appeal/Qugiy/appeal_210518.pdf
[9] http://id.nii.ac.jp/1001/00199872/

ponderhit した場合、次の局面の指し手予測の第一候補をその ponderhit したエンジンに割り当てることで、前回の探索結果を再利用する。

# Ryfamate Cross Network

Komafont*

2023 年 4 月 21 日

## 1 はじめに

Ryfamate は、従来よりコンピュータ将棋の発展に寄与してきた NNUE 型評価関数 [1] と、近年目覚ましい成長を遂げる Deep Learning (DL) 系評価関数 [2] を組み合わせ、それぞれの良さを活かすことを目標としている。2021 年には NNUE 型評価関数と DL 系評価関数の変則合議を採用し、2022 年には変則合議に用いる DL 系評価関数に、自然言語処理の分野で注目される Transformer 型の評価関数 [3] [4] を導入した。2023 年の今回は、DL 系評価関数の主流である ResNet に、NNUE や Transformer の持つ性質を取り入れた、新しいアーキテクチャ Ryfamate Cross Network (RyfcNet) を採用する予定である。

本書では、日本語と図を用いて簡潔に RyfcNet を紹介する。

## 2 背景

現在、コンピュータ将棋に用いられる DL 系評価関数は、画像認識において高い性能を有する深層畳み込みニューラルネットワーク、ResNet [5] である。この畳み込みは、盤面のうち主に 3×3 の部分空間ごとに特徴量を得るものであるが、その性質上、離れた位置にある駒の関係を認識するためには多数の畳み込みを行う必要がある。また、この畳み込みにおいては、位置によらず同じ重みパラメータが適用されるため、駒の絶対座標の情報を学習に生かすことが困難である。これらの問題を解決するため、RyfcNet では、既存の ResNet に、次に紹介する新しい層を導入する。

* 駒の書体 (Komafont) https://twitter.com/komafont

# 3　アーキテクチャ

RyfcNet では、畳み込みを、任意の次元について入力空間と同じ長さを持つカーネルを、それ以外の次元の方向に移動させながら適用する変換と捉え、その次元を層ごとに適切に選択することで、入力空間上の離れた位置にある情報の関係を少ない回数の演算で効率的に認識する。具体的には、ブロック数やチャンネル数などに応じ、次の層を組み合わせて構成される。

(1) S-Layer

通常の畳み込み層であり、チャンネル方向に入力空間と同じ長さのカーネルを持ち、縦横方向にカーネルを移動させながら適用する。

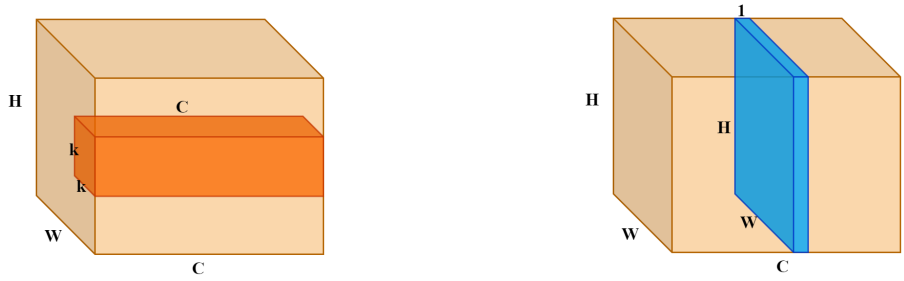
(2) F-Layer (**図** 1)

チャンネルごとの全結合または空間方向 (縦横方向) の self-attention を行う。これによって、盤面全体の情報を、少ない回数の変換で認識することができる。ただしこの変換は、チャンネル数によっては通常の畳み込みと比べて著しく推論速度が遅くなるため、S-Layer と組み合わせてチャンネル数を調整することが有効である。

(3) C-Layer (**図** 2)

任意の 2 つ (以上) の次元について入力空間と同じ長さを持つカーネルを、それ以外の次元についてのみ移動させながら適用する畳み込みを行う。盤面が 9×9 の将棋の場合、通常の畳み込み層は 3×3 の部分空間ごとの計算を 81 回行うのに対し、この畳み込み層では 9×1 や 1×9 の部分空間ごとの計算を 9 回行う。これにより、通常の畳み込み層と同数のパラメータを持ちながら、少ない演算回数で、格子方向に離れた位置にある駒の関係を認識することができる。例えば、図 3 のように、離れた位置にある飛車や香の間接的な効きを少ない回数の畳み込みで認識することができる。さらに、この畳み込み層は選択した次元に移動しないため、重みパラメータは位置に依存したものとなり、駒の絶対座標の情報を学習に生かすことが可能である。これにより、例えば、駒が特定の行 (段) や列 (筋) にあるときのみ反応する畳み込みを行うことができる。

上記の層に加え、ネットワークの構成に応じて、畳み込み層への入力に対し、絶対座標に依存する学習可能パラメータを付与する possition embedding を行う。これにより、例えば自陣や敵陣でのみ反応する畳み込みや、角や桂が初期配置から移動できる位置でのみ反応する畳み込みを行うことができる。




**(a)** S-Layer　　**(b)** F-Layer

**図 1:** S-Layer と F-Layer
S-Layer がチャンネル方向に多くの情報を伝えるのに対し、F-Layer は空間方向 (縦横方向) に多くの情報を伝える。

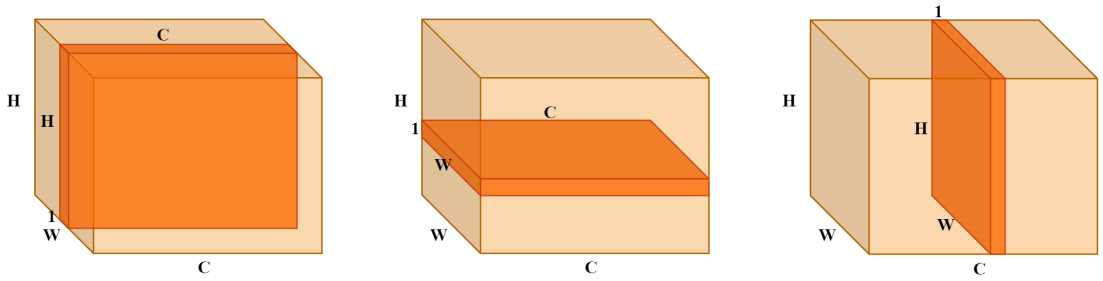
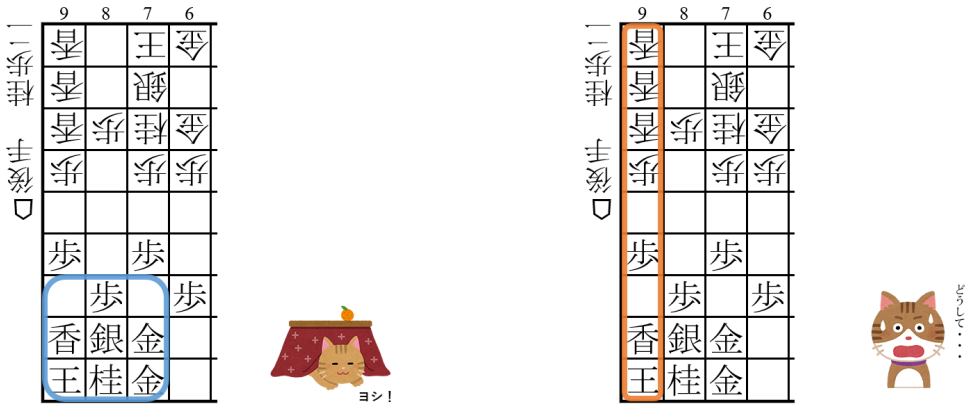
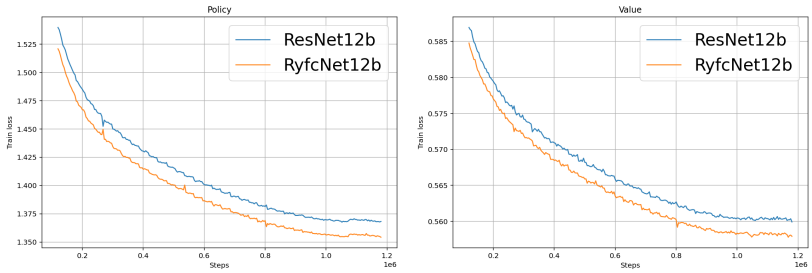



**図 2:** C-Layer
複数の種類の C-Layer を組み合わせることで、離れた位置にある情報を効率的に伝えることができる。




**(a)** S-Layer　　**(b)** C-Layer

**図 3:** S-Layer と C-Layer
S-Layer では離れた位置にある駒の影響を認識するためには多数の畳み込みを行う必要があるが、C-Layer では飛車や香の間接的な駒の効きなど格子方向に離れた位置にある駒の関係を少ない回数の畳み込みで認識することができる。




| | Res12b | Ryfc12b |
|---|---|---|
| Test Policy Loss | 1.400 | 1.385 |
| Acc. | 53.49% | 53.74% |
| Test Value Loss | 0.581 | 0.579 |
| Acc. | 66.81% | 66.87% |
| NPS | 34937 | 37271 |

**図 4:** 学習結果

dlshogi [6] を用いて、2.4 億局面の教師 (hcpe3 形式) を作成し、10 epoch 学習を行った。バッチサイズ 2048。学習率は、Warmup+Cosine Annealing によって調整している。

テストデータは、floodgate および DL 系評価関数と NNUE 型評価関数による自己対局の棋譜を用い、評価値から推測される期待勝率が 99.995% 以内の 512604 局面をサンプリングしたものを用いた。推論速度 (NPS) は、世界コンピュータ将棋選手権および世界将棋 AI 電竜戦にて現れた 5 局面を 10 秒間推論させ、平均値を求めた。

# 4 実験結果

既存の ResNet と本書の RyfcNet を比較するため、同一条件下で ResNet 12 ブロック 192 チャンネル (Res12b) と、RyfcNet 12 ブロック 192 チャンネル (Ryfc12b) を学習した。その性能を比較したところ、図 4 のとおり、Ryfc12b は精度・推論速度ともに Res12b を上回り、対局の結果、表 1 のとおり、 +48.1 (±19.8) のレーティングを記録した。[*1]

| # | PLAYER | : | RATING | ERROR | PLAYED | (%) | CFS(%) | W | D | L |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | dr2_exhi_600ms_140 | : | 49.7 | 25.9 | 800 | 53.6 | 55 | 411 | 36 | 353 |
| 2 | Ryfc12b_ep010_1000ms_140 | : | 48.1 | 19.8 | 1600 | 56.8 | 100 | 870 | 76 | 654 |
| 3 | Res12b_ep010_1000ms_140 | : | 0.0 | ---- | 1600 | 46.6 | 100 | 706 | 80 | 814 |
| 4 | YO763_S5_SC24_16t850ms | : | -50.3 | 26.1 | 800 | 39.6 | --- | 300 | 34 | 466 |

**表 1:** 対局結果

cshogi のリーグ戦機能 [7] を用い、Res12b、Ryfc12b、dr2_exhi [8] による 3 者リーグと、Res12b、Ryfc12b、水匠 5 [9] による 3 者リーグをそれぞれ 1200 局、合計 2400 局実施した。Res12b、Ryfc12b の思考時間は 1 手 1 秒とし、それとおおむね互角の強さとなるよう dr2_exhi と水匠 5 の思考時間を調整した。Res12b、Ryfc12b の探索パラメータは、dr2_exhi のデフォルト値を用いた。初期局面は、36 手目の互角局面集を作成し用いた。

レーティングは Ordo [10] を用いて計算している。

[*1] 実験環境：
Ryzen Threadripper 2950X, GeForce RTX 3090, WSL2+Docker(nvidia/pytorch:22.12)

# 5 おわりに

本書では、比較実験のために 12 ブロックのモデルを用いたが、大会では、15 ブロックまたは 20 ブロックのモデルを用いる予定である。比較実験においては、自作の教師データのみを用いて学習したが、大会用のモデルでは、自作の教師データに加え、加納氏・山岡氏の公開した教師データ [11] から約 6 億局面、たややん氏が公開した教師データ [12] から約 1.2 億局面を利用している。また、探索部は、dlshogi とやねうら王 [13] を一部改良して用いているほか、教師局面の作成や計測には floodgate の棋譜や公開された多くの評価関数を利用している。限られた時間と計算資源の中で新しいモデルアーキテクチャを開発するには、これらの膨大なオープンソースが必要不可欠であり、ここに感謝の意を表したい。

なお、本書で紹介した新しいモデルは、ご協力いただける方がいれば学習を進めたうえで公開したいと考えており、多くの方にご賛同いただければ幸いである。

# 参考文献


[1] 那須悠. 高速に差分計算可能なニューラルネットワーク型将棋評価関数. 2018.

[2] 山岡忠夫, 加納邦彦. 強い将棋ソフトの創りかた Python で実装するディープラーニング将棋 AI. マイナビ出版, 2021.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, Vol. 30, , 2017.

[4] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision. arXiv preprint arXiv:2006.03677, 2020.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[6] `https://github.com/TadaoYamaoka/DeepLearningShogi`.

[7] `https://tadaoyamaoka.hatenablog.com/entry/2021/04/06/225615`.

[8] `https://tadaoyamaoka.hatenablog.com/entry/2021/08/17/000710`.

[9] https://twitter.com/tayayan_ts/status/1463107309492531202.

[10] https://github.com/michiguel/Ordo.

[11] https://tadaoyamaoka.hatenablog.com/entry/2021/05/06/223701.

[12] https://tadaoyamaoka.hatenablog.com/entry/2021/05/06/223701.

[13] https://github.com/yaneurao/YaneuraOu.

やねうら王　アピール文書(あとで書き直す)

今回はDeep Learning(以下DLと略す)を用いたやねうら王の亜種である「ふかうら王」を用いる。
ふかうら王はdlshogi互換エンジンであり、dlshogiとほぼ同等の機能を持つ。
また詳しくは述べないが、dlshogiはAlphaZero型のMCTSを用いた思考エンジンである。

ふかうら王では、その探索の時にleaf nodeにおいて簡単な詰将棋を調べている。dlshogiでは3手～5手詰めを調べていたが、やねうら王では探索ノード数を固定したdf-pn詰将棋ルーチン(以下df-pnと略す)を用いている。df-pnを用いているのは、ノード数を固定し、一定時間で動作が完了するほうが良いと考えたからである。しかしながら、このdf-pnには置換表は用いていない。置換表を用いると大量のランダムなメモリアクセスが発生して、それにより深刻な探索速度の低下を引き起こすからである。

ところが近年、将棋AIにおけるDLを用いた評価関数は囲碁AIと同様にニューラルネット(以下NNと略す)の層をより深くして評価精度を上げる方向で進化している。層を深くするとそれだけ一つの局面の評価に要する時間は増えるのだが、それを補って余りあるだけ局面の評価精度が向上するようなのだ。

今回弊チームで採用する評価関数モデルは、わりと層が深いので探索速度は従来のものより十分に低いので、従来のものほどCPUはメモリアクセス自体を行わない。だから、leaf nodeでは「置換表ありのdf-pn」を用いたほうが得なのではないかと予想するに至った。

実際に探索中に各leaf nodeで呼び出されるような置換表ありdf-pnを実装する上でいくつかの技術的な困難がある。

・各探索スレッドから並列的にアクセスされる(並列的にアクセスされてうまく動くようにしなければならない)
・先手番からも後手番からも呼び出される。(先手用と後手用の置換表を分ける方法も考えられるがそれは効率が低下するので出来れば情報は共有したい)

という条件を満たすようなdf-pnを開発しなければならない。これは従来のdf-pnより一段と難しい。

またdf-pnだが、naiveなdf-pnアルゴリズムから近年改良が進んで、オープンソースの強い詰将棋ルーチンが公開されている。

安定性が向上した詰将棋エンジン『KomoringHeights v1.0.0』を公開した
https://komorinfo.com/blog/komoring-heights-v1/

上記のKomoringHeightsを参考にさせていただいて、ふかうら王のdf-pnを改良したいと思う。

# WCSC33 W@nderER アピール文書

Updated 2023/05/02
Dated 2023/01/27

---

これまでと同様 に、入玉宣言による勝利を積極的に目指します。
昨年の第3回電竜戦にてNNUEのアーキテクチャ改造が思ったより手ごたえあったため、今年はNNUEのいろんなアーキテクチャで遊びながら頑張る予定です。
(2023/05/02)追記
→最終的に評価関数については第3回電竜戦で用いたHalfKP-KingSafety Distinguish Goldsのアーキテクチャを使用することにしました。
エンジンについてはやねうら王、定跡についてはFloodgate上位プレーヤーの棋譜を手調整したものをベースに使用予定です。

# 第３３回世界コンピュータ将棋選手権「アストラ将棋」アピール文書

令和５年４月
恒岡 正年

1. 全体の構成
　dlshogi の探索部、学習部を利用。
2. 独自に実装した部分
　Network 構造、学習方法、model の生成の工夫。（詳細は後述）
3. 開発動機
　ディープラーニングの特に学習部に興味があり、「強い将棋ソフトの創り方」という書籍の内容をトレースする事で開発をはじめた。
　したがって、自前の model を作る事に注力している。
4. 主な開発内容
　独自構造の model の作成・学習方法
　棋譜生成：6000po 程度、及び 15 秒+F1.2 秒程度の物を中心に現在約 30 万の棋譜作成
　学習には、自己生成した棋譜以外に上記書籍の学習用データを利用
　AobaZero の棋譜も一時期利用したが、現在は未使用。
　探索パラメータの調整(手作業・optuna 利用)
5. 探索ロジックの検討(UctSearch.cpp の改造)
　標準の UCB 値の計算式は n が十分に大きい時に最良の計算式となるが、限られた試行回数では必ずしも最適な方法ではないと思う。
　オリジナルの計算式を独自の式と置き換える。
　探索パラメータの再調整後、R+36 程度の向上を確認。
6. 持ち時間の節約のための簡単な定跡を手入力で準備した。
　ベストラインでは深く、それを外れた場合は(有利になっているはずなので)浅い。
7. 今後の開発予定(大会後)
　指し手選択ロジックの追加(後述)
　思考時間制御(後述)

---

8. Network 構造

Network を次の 3 つに分ける。

(1)データ入力部 (2)ResNET 部 (3)データ出力部

1）データ入力部

3 つの情報の合流位置とデータのサイズを振って実験した。

現在はオリジナルに Conv2d()を 1 つ追加した構造にしてる。

データサイズは、9x9 以外にも 11x11、11x9 を試した。Play Out 数固定での評価では有意に強くなるが、探索速度の低下が大きく探索時間一定の条件では弱くなった。ネットワーク層数またはカーネル数を増やす方が良い。

将来的にネットワーク層数またはカーネル数を増やしても改善効果が小さくなった場合に、データサイズ 11x11 を試してみたい。

2）ResNET 部

オリジナルの ResNET に限らず何通りかの形状(Conv2D()3 層の ResNET や入れ子構造の ResNET 他)のネットワークを試したが、最終的にオリジナルと同じ Conv2d() 2 層の ResNET にした。

データサイズは 9x9。(11x11、11x9 も試したが不採用)

カーネルサイズは 3x3 のみ評価。

データ入力側と出力側でカーネル数が異なる構造を採用。

ResNET 部の層数は 25。

3）データ出力部(Policy/Value Network の分岐後)

オリジナルの構造に Conv2D()を 1 つ追加している。

ResNET を何層か追加する実験も行ったが優位性は認められなかった。

4）その他

活性化関数は 主に ReLU を、最も出力に近い Conv2D()にのみ SiLU を使っている。

計算量は入力部と出力部に Conv2D()を追加しているため ResNET27 層相当。

9. 学習方法

1）今回の使用モデルでは、学習率(lr)：0.2 から始めて 0.000000012 まで等比数列的に減少させ学習した。減少率は主に 0.85。最後の数エポックは

学習率を固定値とした。
100 エッポックの学習で 2 週間～3 週間かかる。
　2）バッチサイスは 512 から初めて GPU メモリの許す限り順次大きくしている。
　3）マクロバッチを実装し、最終的に 12 倍まで学習単位を大きくしている。8 倍～12 倍に最適値が有りそう。
(FP16 だとこれより大きくすると精度が足りない。(学習率) x (バッチサイズ) x (マクロバッチ倍率) が一定値を超えると収束付近で学習効果が無くなる様に思う。)
　4）小さめの学習セットで 1 エポックの学習を数回実行し、最も D 値 (=Loss-(PolicyAcc+ValueAcc)*0.5 )が小さくなった物を採用し以降の学習を行っている。初期値の分布の素性の良さそうな物を選ぶ。
　5）25 エポックまでに 3 回、入力に近い部分の ResNET を入れ替えて学習する。これにより収束を早める効果を狙っている。
　6）26 エポック以降も D 値をモニターし、D 値の減少が期待値以下の場合に、意図的にイレギュラーなデータを与える。データの与え方は 2 種類の手法を用意した。
　特定の次元が局所解に陥っている可能性を考慮しそこから脱出するのを期待している。
　7） D 値をモニターしていると収束に近づいているかどうかがわかるので D 値の挙動によって終了するエポックを決める。今回使用するモデルの場合約 100 エポックまで学習した。何エポックまで学習するかはモデルに依存する。

10. model の生成の工夫
　学習済のモデルは、最も Loss 値が低い、または最も Accuracy が高い物が勝率が高いモデルとは限らない。これは、Loss 値や Accuracy 値は出現頻度の高い特徴を持つ局面での正解率に強く依存し、出現頻度の低い特徴を持つ局面での正解率の寄与割合が低いためと考えた。将棋の勝敗は出現頻度の低い特徴の局面で正しい手を指せるかにも依存する。
　すべてのモデルでベンチマークを取るのを回避しつつ、この問題の影響を低減する工夫を行った。

11. 定跡

　手入力で定跡を用意した。思考時間の節約を目的としており、最長でも30 手程度と思う。

今回のモデルを用いて思考時間一手１分～５分で思考させる。２手目４通りの開始局面からベストラインを長く登録している。

ベストラインからの枝は短く評価値が(先手に)振れたら打ち切っている。

一本道の局面では打ち切らず、同程度の評価値の候補手が複数現れたら打ち切る。

-以上-

第33回世界コンピュータ将棋選手権　「東横将棋」アピール文書

2023.3.31（2023.5.1改訂）

東横コンピュータ将棋部

定跡と標準NNUE評価関数の極北を目指しています。
・従来の強化学習手法に加え独自の標準NNUE評価関数の強化
・手作業による定跡の生成。今回は角換わり定跡に主眼を置いています
・探索部はやねうら王、いわゆるやねうら王チルドレンです。やねうら王＋最新のStockfishのキャッチアップを予定

よろしくお願いいたします。

んんんｗｗｗ
性懲りもなくまた出るんですなｗｗｗ

・役割論理とは
第32回世界コンピュータ将棋選手権でdlshogiと水匠（2回）に大勝利して（辞退）完全かつ最終的に確立された論理ですなｗｗｗぶっちゃけs-book_blackが無双しただけですぞｗｗｗ
定跡と標準NNUE型評価関数と高級スリッパ＆クラウド重課金(笑)の圧倒的火力によって評価値ダメージレースを制する必勝の戦術ですなｗｗｗ

・定跡について
floodgateその他で収集した棋譜を元に手作業で作成した居飛車定跡「火葬砲定跡」を使用しますぞｗｗｗ
先日、電竜戦さくらパイルール2023が開催され棋譜も公開されているのでそちらも利用させて頂くしかありえないｗｗｗ

・戦型について
相掛かり、角換わり共に先手必勝が確定しましたなｗｗｗ後手番は対策が急務となっておりますぞｗｗｗ
ちなみに振り飛車は先手後手ともに必敗ですなｗｗｗ

横歩取り：先手必勝ですなｗｗｗ
一手損角換わり：先手必勝ですなｗｗｗ
雁木：後手番も選択可能な戦型ではありますが先手番があえて指す必要はなさそうですなｗｗｗえぐいですなｗｗｗ
矢倉：先手があえて指す必要はなさそうですなｗｗｗ矢倉は終わりましたｗｗｗ
相振り飛車：なんでわざわざ振り飛車を指す必要があるんですかなｗｗｗ
筋違い角：後手の振り飛車党への嫌がらせの精神攻撃ですなｗｗｗそれ以上でもそれ以下でもありませんぞｗｗｗ

まずは後手番でどこまでダメージを最小限にするかが重要ですなｗｗｗ
現状ほぼ先手後手が同じ割合になるので後手番でも勝てそうな相手(酷い)に必然力で対戦することが重要ですぞｗｗｗ
互角の別れで逃げられればNNUE型評価関数の終盤の強さと高級スリッパ＆高級クラウドコンピューティング()の超火力で踏み潰すだけですなｗｗｗ
もちろん定跡を整備しないとdl系やや○うら王などの強豪には手も足も出ませんぞｗｗｗ

・必然力とは
論者を圧倒的勝利に押し上げる力ですなｗｗｗ
強　豪　に　絶　対　に　先手を引く、後手番での当たりが良いｗｗｗ裏街道最高ｗｗｗなどは必然力とされていますなｗｗｗ
ヤーティ神への信仰によって得られる加護とされていますぞｗｗｗ
やはりこれが最も重要なファクターとなっていますなｗｗｗ

結局「評価関数の強化」と「定跡の強化」という極めて当たり前の結論に至る訳ですなｗｗｗ
将来的にはDL系とマメット・ブンブク形式（halfkp_1024x2-8-32）のハイブリッド＋強力な定跡が主流になっていくのではないですかなｗｗｗ
公開しないorあれこれ制限をかけまくるスタイルが主流になって停滞し尽くす未来もありえますがなｗｗｗ

・評価関数について
みんな大好き(笑)標準NNUE評価関数を使用しますぞｗｗｗ
今さらマメット・ブンブク形式（halfkp_1024x2-8-32）で後追いしても車輪の再開発どころかまともに転がるものも作れそうにありませんからなｗｗｗ
標準NNUEなんてもう完全にサチってでオワコンで通常の強化学習ではゴミのような評価関数を量産するだけですが無理矢理しばき倒して強化するしかありえないｗｗｗ
そしてさすがに水匠5よりは強くないとお話になりませんなｗｗｗゴミｗｗｗ
もちろん振り飛車評価関数は総合的にロジックするまでもなくありえないｗｗｗ

・使用予定の評価関数
SQMZ_Cendrillon2：あふろん@Grampus氏()の未公開のはずの標準NNUEtakeshiの振り飛車評価関数ですなｗｗｗその中身は単にSQMZをブースト処理しただけの代物（らしい）ですぞｗｗｗ振り飛車評価関数を完全否定しつつ適当に振り飛車評価関数を使う、ぶっちゃけ舐めプですなｗｗｗ
SQMZ_tempest_20230102：あふろん@Grampus氏()が第1回マイナビニュース杯電竜戦ハードウェア統一戦の準決勝で使用して惨敗した未公開のはずの評価関数ですなｗｗｗ振り飛車成分を完全脱臭している（らしい）のでもはやSQMZと呼べる代物ではありませんぞｗｗｗ
SQMZ_tempest_20230403：あふろん@Grampus氏()が電竜戦さくらパイルール2023で使用して8戦全勝した未公開のはずの評価関数ですなｗｗｗコンセプトは20230102()とほぼ同じですがかなりコンサバ()な評価関数（とのこと）ですぞｗｗｗ

・シードについて
前回あれだけやらかしたのに普通に第9シードですなｗｗｗ感謝しかありえないｗｗｗ
そのWCSC32で溶鉱炉に沈めた第10シードの水匠電竜()はやねうら王チームに参加しているようですなｗｗｗ

・東横将棋について
A.東横将棋はGrampusですか？
Q.違いますなｗｗｗ東横将棋は東横将棋であってそれ以上でもそれ以下でもありませんぞｗｗｗ

・余談
あふ「準決勝で相入玉した時点で勝ったと思いましたよ、まだ宣言勝ちの修正をしていないと思っていたので」
たや「そんなのとっくに修正してるに決まってるじゃないですかｗｗｗ」
あふ「(　；∀；)」

・floodgateテスト
世界で一番かわいそうな高級スリッパを使って色々実験的な放流をしてみましたぞｗｗｗ
結果はぼちぼちでんなｗｗｗ

## 二番絞り

二番絞りの誕生経緯については昨年のアピール文などを参考に頂ければ幸いであるが基本的には最高精度を目指す大きなモデルを作成しようとする試みである．

昨年度は幸いにも準優勝という好結果を得た．準備段階では手ごたえがなかったがその後の計測で大変高精度なものが完成していたことが分かった．今年度も似たような状況である．まともな計測に至っていないが，もし昨年度より弱いと判断した場合は昨年度版で出場する可能性がある．

ちなみに，昨年度版の局面評価精度は驚愕の域に達しており，既発表であるが一手の局面展開も行わず将棋倶楽部 24 でレート 2949，八段認定頂いている．前人未到の領域といって過言ではないだろう．
また，電竜戦ハードウェア統一戦においても準優勝となり，記念に 2017 年に行われたハードウェア統一戦の第 5 回電王トーナメントを思い起こし GTX1080Ti の二番絞りを floodgate に投入したところ短期レート 4500 台を記録した．（もちろん一時的なものでありしばらくしてレートは 4100〜4200 程度で落ち着いた）

今年度はこれを上回ることを目指しているが上記の通り昨年に続き未計測である．

参考：
芝，「将棋の PV-MCTS に向けた深層学習モデルの最適化」，第 45 回ゲーム情報学研究会
芝，「探索アルゴリズムに適した時間利用に関する研究」，第 46 回ゲーム情報学研究会
第 32 回世界コンピュータ将棋選手権，https://bleu48.hatenablog.com/entry/2022/05/06/145915
芝，「コンピュータ将棋における高精度な深層学習モデル」，ゲームプログラミングワークショップ 2022
二番絞り＠将棋倶楽部 24 の戦型分析，https://bleu48.hatenablog.com/entry/2023/03/08/062634
二番絞りの計測の件（GX1080Ti 編），https://bleu48.hatenablog.com/entry/2023/03/09/132936

[第33回世界コンピュータ将棋選手権]

# 大将軍
# （たいしょうぐん）
# アピール文書

横内　健一
横内　靖尚

# 大将軍の概要

- 評価関数に主眼を置いた将棋プログラム。
- 評価関数の特徴としては、盤上の3駒の位置関係を考慮。

  過去には4駒の位置関係の評価関数で参加したこともありますが、結局は3駒の位置関係に落ち着きました。

# 大将軍の概要

- 過去の遺産をベースとし、評価関数の学習においては、（いまさらながらではあるが・・・）
  - 駒の位置関係の相対位置による評価
  - 手番の学習

  を考慮。
- 3駒関係（kppt型）を基本とするが、学習対象とする局面は浅い探索の末端局面と現局面をミックスし学習するよう調整。教師データも3駒系とNNUE系の棋譜をミックスして利用。

# 使用ライブラリ

- やねうら王
  - 最新のStockfishの探索ルーチン
  - わかりやすいコード（過去および現在のアイデアを適用しやすい）
- 水匠２～４
  - 評価関数学習用の棋譜作成(教師データ作成)に利用