# 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。