# 『やねうら王　with お多福ラボ 2019』(WCSC29 優勝)アピール文書

やねうら王チーム　やねうらお(磯崎　元洋)

## ■ 評価関数の強化学習に用いる教師局面での工夫

本ソフトでは、評価関数として従来の KPPT 型(3 駒関係+手番)から NNUE 型を採用した。NNUE 型は、tanuki-チームメンバーである那須さんが開発された評価関数である。[1]

本大会で決勝に残った 8 チームのうちの名人コブラを除く上位 7 チームが NNUE 型を採用していたことからも、NNUE 型の優秀性が窺える。

NNUE 型の特徴として評価関数のパラメーター数が KPPT 型より少ないということが挙げられる。強化学習のために必要な教師局面の数は、評価関数のパラメーターの数におおよそ比例すると考えられるので、NNUE 型は KPPT 型より少ない教師局面数で済むはずである。

このことから、本ソフトではなるべく質の良い教師を生成することに重きを置いて開発を行った。

将棋の評価関数の強化学習は、通例 elmo 式[2]を用いて行われる。[3]

本ソフトでは、elmo 式の勝敗項の精度を高めるために短時間(1 手 0.1 秒対局)での勝率を上げるような探索パラメーターのチューニングを行い、その探索部を用いて教師を生成した。また、同時に勝率項の分散が大きくなると学習しにくくなると予想されるため、勝率項の分散が大きくなるような探索の枝刈り(Futility Pruning、Singular Extension、etc…)を抑制しながら教師を生成することにした。これについては、事前に提出したアピール文書に詳しい。[4]

---

[1] https://www.apply.computer-shogi.org/wcsc28/appeal/the_end_of_genesis_T.N.K.evolution_turbo_type_D/nnue.pdf

[2] http://www2.computer-shogi.org/wcsc27/appeal/elmo/elmo_wcsc27_appeal_r2_0.txt

[3] http://yaneuraou.yaneu.com/2017/05/23/elmo%E3%81%8C%E3%82%82%E3%81%9F%E3%82%89%E3%81%97%E3%81%9F%E3%82%AA%E3%83%BC%E3%83%91%E3%83%BC%E3%83%84%E3%81%AB%E3%81%A4%E3%81%84%E3%81%A6/

[4] https://www.apply.computer-shogi.org/wcsc29/appeal/YaneuraOu/appeal.txt

## ■ 定跡面での強化

次に、野良評価関数として従来の評価関数より二回りほど強いと言われている NNUEkai や illqha3 などが本当に強いのかを計測した。それらの評価関数は、定跡なしで対局させると確かに『将棋神やねうら王』に収録している tanuki-2018 年度版の評価関数(以下、nnue-tanuki と略す)と比べて勝率が高かったが、定跡なしの場合は、同じような進行になりやすく、公平な計測とは言い難い。

そこで、プロの対局棋譜の 24 手目から対局させた。すると nnue-tanuki と互角程度か、むしろ nnue-tanuki のほうが若干強いようであった。このことから、野良評価関数で強いとされている評価関数は序盤の 16 手目ぐらいまでの指し手が優秀で(おそらく短時間で勝ちやすい戦型に誘導しており)、そのあと 24 手目からその優位さを吐き出しているのであろうと推測がついた。

NNUE 型評価関数には、戦型を判定する機能が備わっていると考えられるので[5]、序盤に対する強い学習能力があると考えられる。逆に言うと、序盤の 16 手ほどを定跡で乗り切れば、あとは力勝負であり、24 手目以降が強い評価関数を用いれば互角以上で戦えるということである。

そこで私は与えた棋譜の各局面を depth 36 相当で探索させ、そこを leaf node として minimax 探索のようなことをして定跡木を生成することにした。定跡 5 万局面分を生成するために 100T(100 テラ ＝100 兆)局面の探索をさせたことから、この定跡のことを『**テラショック定跡**』と名付けた。生成手法については、やねうら王のブログ記事[6]に詳しい。

本大会では、一次予選のうち二次予選に勝ち上がることが確定したチームの対局棋譜を加えて定跡を追加生成した。二次予選についても同様である。最終的に決勝日の時点で、700T(700 兆)局面を探索して、35 万局面分の定跡が出来上がっていた。そのためか、本大会でやねうら王が定跡の質や定跡の長さで負けた対局はほとんどなかったように思う。[7]

---

5 http://yaneuraou.yaneu.com/2019/02/05/nnue%E8%A9%95%E4%BE%A1%E9%96%A2%E6%95%B0%E3%81%AE%E3%82%88%E3%81%86%E3%81%AA%E9%9D%9E%E7%B7%9A%E5%BD%A2%E3%81%AA%E8%A9%95%E4%BE%A1%E9%96%A2%E6%95%B0%E3%81%AF%E4%BD%95%E6%95%85%E6%9C%89%E5%8A%B9-2/

6 http://yaneuraou.yaneu.com/2019/04/19/%E3%83%86%E3%83%A9%E3%82%B7%E3%83%A7%E3%83%83%E3%82%AF%E5%AE%9A%E8%B7%A1%E3%81%AE%E7%94%9F%E6%88%90%E6%89%8B%E6%B3%95/

7 http://yaneuraou.yaneu.com/2019/05/06/wcsc29%E3%80%81%E3%82%84%E3%81%AD%E3%81%86%E3%82%89%E7%8E%8B%E5%84%AA%E5%8B%9D%E3%81%97%E3%81%BE%E3%81%97%E3%81%9F%EF%BC%81/

# チーム Barrel house のアピール文書@第 29 回世界コンピュータ将棋選手権

Barrel house とは岡山の駅前にあるビアバーです．昨年チームメイトを求めさすらって行きついたところです．マスターに許可頂いたので名前をお借りしました．その後，メンバーが変わって当初の方向性とは全く違って来ましたが，まぁ一度出した名前を変更するのもアレなので今年もそのままです．

プログラム名：Kristallweizen

昨年の Hefeweizen が濁った白ビールでしたが，Kristallweizen はフィルターでろ過した透き通った白ビールです．命名経緯は上記のビアバーに起因します．
命名時は昨年より洗練されたソフトにしたいとの気持ちでしたが，準備ペースは昨年同様で洗練とは程遠い感じです．
ちなみに，マスターによると Kristallweizen は樽で輸入するのが難しいので瓶になりますが大丈夫ですかとのことでした．
祝勝会では数が揃わないので Hefeweizen で乾杯する予定です．

チームの特徴.

昨年は段取りが分からず全関係者を加えておりましたが開発者に絞りました．今年からのメンバーもお願いして参加頂きました．若いメンバーも大口スポンサーもありませんので，多方面に精通した開発者の独創的なアイデアだけが勝負どころと考えています．昨年は独創的な評価関数生成と先行先読み Multi Ponder が冴え渡り優勝致しましたが，今年は別のブルーオーシャンへ漕ぎ出す予定でしたが結局レッドオーシャンだったようです．
昨年同様別々に開発を進めてネット上で連絡を取り合う程度の疎結合チームです．

CSA 使用可能ライブラリ使用表明.

最終的には以下のものだけ利用しました．
やねうら王，tanuki-，技巧

ライブラリ選定理由

やねうら王：探索部が高速なため主に探索部の利用。定跡部作成時にも利用。
tanuki-：評価関数に利用。
技巧：Multi Ponder のオーダリングに利用。

使用マシン.

普通のノートパソコンに加えて，クラウドの力をお借りしました．昨年同様 AWS の m5.24xlarge を 5 台使用しました．

クラスタリングについて

昨年 Multi Ponder のクラスタリングを行いました．今年は Go 言語で再実装して追加機能を加える予定でしたが通信遅延の隠ぺい程度の成果に終わっています．リアルタイムで各ノードの情報が得られるようになったので今後の追加機能の下準備ができたと考えます．

評価関数の設計思想

中終盤の評価値が拮抗する難解な局面において、あえて最善手を選ばずに、次善手を選択するようにすることで、相手の読み筋を意図的に外して Ponder hit させない。これは昨年の Hefeweizen で実現していた特徴で、将棋倶楽部 24 で有段者の方々とスパーリングを行って細かく調整した賜物である。将棋倶楽部 24 のユーザーさんから「まるで大山将棋のようだ」という、絶賛の声をいただいた。調整にご協力いただいた将棋倶楽部 24 のユーザーの皆様、この場を借りて御礼申し上げます。
Hefeweizen 同士の対局で生成した棋譜を教師局面としているため、千日手、手数制限による引き分けの棋譜が多数あったためか、異様に粘り強く、負けない辛い指し回しを行う。また、すぐに入玉したがるようになっている。いいんだか悪いんだか……。
学習部自体はやねうら王 Ver4.83 を使用し、人造棋士 18 号以来の改良として、低スペック PC でも学習を行えるようにメモリ下限を下げる修正を行い、評価関数はやねうら王にマージされている NNUE（tanuki-チーム提供）を使用している。これは、WCSC28 で Hefeweizen を開発していた時から感じていた「KPPT の学習の頭打ち感」を打破したいという思いがあり、打破するにはどうしたらいいかと考えていたところで NNUE が公開され、やねうら王にマージ　されたことから、流行に乗ってみようという安易な発想から採用した。
また、WCSC28 で優勝した KPPT 型の評価関数の Hefeweizen の持つ、中盤のねじり合いの局面では最善手をあえて外して次善手を選択するようにし、相手の Ponder hit を外すことによって探索部の特徴を最大限に活かす指し手選択を行うという特徴を、そのまま NNUE 型にも引き継がせることを目標にして作成した。
評価関数ができあがったのが、5/2 の 7:56（タイムスタンプより）という、ギリギリの状態であったが、親である Hefeweizen の棋風を受け継ぎつつ、その棋力を向上させた NNUE 型の評価関数とすることができた。

完成までの行程

１．NN 構成の選択

やねうら王にマージされている NNUE のディフォルトの NN 構成は、HalfKP-256x2-32-32 となっており、KPPT の探索速度に匹敵するスピードを得ることを目的に最適化されており、学習精度、探索速度のバランスでこれを超える NN 構成を模索したが、選手権までに発見することはできなかったため、最終的にディフォルトの NN 構成を採用した。

ただ、学習方法の工夫によって、学習パラメータの選択方法がおぼろげながら見えてきたため、再び NN 構成の検証を再開する予定である。

２．教師局面生成用評価関数

初期学習のための教師局面生成に WCSC28 で優勝した Hefeweizen を使用して居飛車、振り飛車を指させ、depth10 で約 3.5 億局面を生成した。本当はもっと生成したかったのだが、手元にあるマシンリソースの関係上、これが限界だった。そのため、この少ない教師を効率よく学習させる工夫を行って使い回すこととした。

Hefeweizen は KPPT 型であるが、人造棋士 18 号の評価値調整ツールによって汎化が比較的うまくいっており、中終盤の指し手の多様さが魅力である。この評価関数が生成する教師局面によって、実戦で出てきやすい局面に対して様々な応手で対局した局面が生成できたため、深層学習の学習時の問題点である過学習に比較的陥りにくかったと考えられる。

また、生成時には千日手や手数制限によって引き分け扱いになった対局データも多数生成されるのだが、それもそのまま教師データとして学習に使用している。選手権時に他の開発者と学習について話す機会があったのだが、引き分けデータは勝率に結びつかないとして教師データから取り除いているとのことであったが、千日手も 0.5 勝という立派な勝ちであるので、これを取り込まない手はないと思う。そのせいか、異様に粘り強く、辛い指し回しを行うようになった。

３．学習方法の工夫

NNUE の学習、というかやねうら王に実装されている学習部の仕様は、1epoch 目のベクトルの傾きによって学習度合い、按点のハマり具合が大きく左右されることがわかっている。この問題に対処するため、以下の手順で学習することで、安定した学習を行えるようにした。以下、これを彫刻作業になぞらえて説明する。

3.1. シード（種）学習 …… 木の切り出し

学習ゼロの状態でいきなり大量の教師データで学習した場合、1epoch 目の学習ベクトルの傾きが芳しくないと、いくら大量に学習したとしても棋力が向上しないことが検証により判明した。こうなってしまうと学習をやり直すしかないのだが、こういった無駄な時間を少しでも減らすため、居飛車、振り飛車両方の局面をほどよく混ぜた教師データを 200 万局面用意し、eta 1、lambda 1 で学習させ、ロスと accuracy を確認し、問題があれば再度やり直すという「傾きガチャのリセマラ」を短時間で繰り返すことによって、不要な過学習、鞍点へのハマりを抑えるようにした。

これはなかなか骨の折れる作業であるため、自動化も考えたが、自動化のための基準をコーディングする手間のほうがかかってしまうのは本末転倒であるので、目視によって確認していった。

3.2.ミニミニバッチ学習 …… 粗削り

生成した約 3.5 億の教師データを、1 ファイルにつき 200 万局面（つまりシード用の教師データと同じ単位）に分割し、それを適当なフォルダに放り込んでおき、1 ファイルずつランダムに取得しながら（targetdir でフォルダ指定するのではなく、Linux の ls コマンドの結果をランダムソートして for ループで取り出している）nn_batch_size 10、batchsize 10000、eta 1、lambda 1 で学習するループを適度に行った。ファイルを読み込む度に初期ベクトルは変わり、バッチサイズも小さいため精度も低くなるが、多様な傾きでの学習が行われることで、精度は低いものの、それなりに均等な　値を持たせることができたと思っている。

3.3.通常のミニバッチ学習 …… 彫刻刀による彫刻

約 3.5 億の教師データを今度はシャッフルしながら 1 ファイルにまとめ、nn_batch_size 1000、batchsize 10000000、eta 0.1、lambda 0.5 で loop パラータで回数指定して学習を行った。こうすることで、起動時に傾きガチャを行って下る方向を決めたら、後はその傾きで一気に学習を行うことで、粗削りな評価値を大きめのバッチサイズで精度を上げつつ、accuracy を上げていった。その際、適宜 lambda を調整しながら勝敗項稼ぎのループ、勝率項稼ぎのループを交互に回しながら上げていった。この段階では、move accuracy は 37%が限界だった。

3.4.最終調整 …… やすりがけ

3.3.の状態では、教師の素である Hefeweizen にも勝ち越せない程度でしかなく、途方に暮れていたのだが（この時点で 5/1 だった）、最後にやけくそで eta 0.01、lambda 0.1 で 3.3.の学習を 1 ループだけ行ったところ、奇跡的に move accuracy がぐんぐん伸び、最終的に 42%まで上昇した。この状態で仮想敵としていた orqha1018+やねうら王 4.83 と 5000 万ノード指定で 10 対局させたところ、8 勝 1 敗 1 分という結果が出た。対局数が少ないと思われるかもしれないが、5/1 時点でそれほど対局を回せるはずもなく、また、全対局の指し回しを目視していて、異様に辛くて粘り強い、ビールに辛子と納豆を入れたような棋風に仕上がっていたため、これを採用とした。

このように、学術的な根拠などまるで皆無な思いつきで作成された評価関数であるが、それでも昨年の Hefeweizen の棋風を受け継ぎつつ、その棋力を向上させることに成功したのは、僥倖としかいいようがない。

実験結果や追試について記述を求められていますが，上記のような選手権開催前日に完成した評価関数を持ち込んだため事前の実験結果というべきものは皆無です．事後実験に可能なものは公開させて頂きましたので，御興味ある方は是非実験して結果を公開して頂ければ関係者皆が喜ぶと思います．

# 狸王 狸王詳細アピール文書

*ザイオソフト コンピュータ将棋サークル*

*野田久順 岡部淳 鈴木崇啓 河野明男*

## 開発動機

コンピュータ将棋ソフトを通した技術の習得を目的として開発を行いました。

## 開発過程

評価関数・探索パラメータの自動調整・クラスタの Lazy Cluster 以外の部分の実装を野田が、Lazy Cluster の基本部分を河野が担当しました。残りのメンバは理論部分の検証・考察を担当しました。

### 評価関数

tanuki-wcsc28 に収録した評価関数を強化学習しました。強化学習の手法には「教師データが不足した環境での機械学習結果改善手法」と「elmo 式学習法」を用いています。教師局面の生成時には、思考エンジンの探索パラメータを短時間の思考に最適化し、思考ノード数を固定して自己対局を行っています。

### 探索パラメータの自動調整

wcsc26 で導入した探索パラメータの自動調整を用いました。探索パラメータと自己対局の勝率のサンプリングには約 1 ヶ月かけました。

### クラスタリング

はじめにチームメンバの河野が、Lazy Cluster の基本部分を実装しました。実装は tanuki-wcsc28 のクラスタマスタープログラムを改造する形で行いました。その後、野田が Multi Ponder を実装しました。また、Multi Ponder の実装を再利用する形でゲーム木の分割による並列探索を実装しました。最後に、Multi Ponder とゲーム木の分割による並列探索の対象ノードを、詰将棋専用エンジンでも探索させるよう改良しました。

## 実験結果

### 評価関数

対 orpha-1018・やねうら王互角局面集 24 手目から対局・1 スレッド 1 手 200 万ノード

対局数 1000 先手勝ち 499(52%) 後手勝ち 455(47%) 引き分け 46
engine1
勝ち 500(52% R16.77) 先手勝ち 271(28%) 後手勝ち 229(24%)
宣言勝ち 6 先手宣言勝ち 4 後手宣言勝ち 2
先手引き分け 27 後手引き分け 19
engine2
勝ち 454(47%) 先手勝ち 228(23%) 後手勝ち 226(23%)
宣言勝ち 4 先手宣言勝ち 2 後手宣言勝ち 2
先手引き分け 19 後手引き分け 27

### 探索パラメータの自動調整

パラメータを自動調整していない思考エンジンとの対局・やねうら王互換局面集 24 手目から対局・1 スレッド 1 手 300 万ノード

対局数 1000 先手勝ち 458(49%) 後手勝ち 470(50%) 引き分け 72
engine1
勝ち 523(56% R44.42) 先手勝ち 266(28%) 後手勝ち 257(27%)
宣言勝ち 5 先手宣言勝ち 4 後手宣言勝ち 1
先手引き分け 40 後手引き分け 32
engine2
勝ち 405(43%) 先手勝ち 192(20%) 後手勝ち 213(22%)
宣言勝ち 9 先手宣言勝ち 7 後手宣言勝ち 2
先手引き分け 32 後手引き分け 40

## 追試可能か

ソースコード・実行バイナリ・評価関数ファイルを GitHub で公開しているため、追試は可能だと思われます。 https://github.com/nodchip/tanuki-

# elmo アピール文書 word 版

瀧澤 誠 twitter: @mktakizawa

## 開発動機

元々はボナンザメソッドと呼ばれる機械学習の仕組み(識別でも回帰でも無い(pairwise な rank 学習))に興味を持ち、何かより良く出来るのではないかと考えたことがきっかけだったと思います

## 開発過程

### 調査

最初は既存技術の調査をしつつ今後の開発方針を考えます。半年位。今回は NNUE のネットワークやパラメータを変えてどう変わるかという調査と、高速な CNN 実装について検討しました。

### 最先端へのキャッチアップ

具体的には公開されている illqha/NNUEkai/orqha に追いつく作業ですが、教師データの生成に時間が掛かることと、思うような結果が出ず、上記検討していた新規実装は見送りになっています。この間あれこれ試した内容がアピールポイントとなっています(概ね不採用でしたが)。

### 新規仕組みの提案

上記状態のため断念しています。パワーポイント版のアピール文書に内容をぼかして(正直云うと少々外して)書いた内容です。評価関数部分で色々考えていますが探索部触りたい病とかが発病しているので今後はそっち方向に行ってしまうかもです。

## 開発内容

### elmo 式の学習法

既に差別化要素ではないですが。教師データ生成時の勝敗情報と探索結果の評価値の両方を用いて学習する方式です。当時のアピール文書で簡単な説明と、やねうら王、Apery で実装例があります。

http://www2.computer-shogi.org/wcsc27/appeal/elmo/elmo_wcsc27_appeal_r2_0.txt

https://github.com/yaneurao/YaneuraOu/blob/master/source/learn/learner.cpp

https://github.com/HiraokaTakuya/apery/blob/master/src/usi.cpp

### 教師データ生成法の変更

既存の Depth 指定の学習方法では以下のような課題があります。

☹ 終盤の複雑な局面になるに連れて、探索ノード数が飛躍的に増加する(局面によって教師データの生成コストが大きく異なる)

特に、決着がつかないような長手数となった場合、散々計算した挙句教師として採用できない結果となることがあり効率が悪いです。現実的にはこれを回避するために評価値 3000 などで勝ったものと見做す設定をしますが、この場合宣言勝ちを学習することは困難となります。

elmo では tanuki-チームに倣い、Depth 指定ではなくノード数指定で生成するようにして教師生成に掛かる時間を平準化しました(元々は時間指定としていましたがノード数の方が計算資源を有効に活用出来る、異なる CPU でも質を均一化出来る、CPU 割り当ての不平等を解消出来る等のメリットがありそうです)。ただ、効率面での改善はあるものの相対的には序盤の探索数を増やし

て、終盤の探索数を減らす形になるので評価関数にもそのような傾向(序盤に強く、終盤に弱い)があるようです。

### 定跡生成法

昨年に引き続きですが以下、2 つの手法で生成したものを合成して作成しています。

1. 対局結果からの採用
2. 深く読んだ際の指し手の採用

何れも大会版の評価関数を用いて定跡手を作成していますが 1.の局面生成には他評価関数も利用しています。1.は 1 手 1.2～2.4 億ノードで対局した結果、評価値の逆転が無いまま勝利したものを採用しています。2.は定跡を意図的に外してくる対策として、depth33 程度で広く探索した結果を採用しています。1.と 2.で重複した場合は 1 が優先されます。1.は 2 年前に急ごしらえで作成した定跡生成法ですが、比較的少ない計算量で実践的な定跡が作成可能であり気に入っています。

## 実験結果

まず、定跡については短時間の思考時間では定跡を利用することで却って勝率が下がってしまうようでした(指しこなせない)。生成時の 1 手 1 億ノード以上で対局するのは十分な数が稼げないため、実験対象外としています。また、平手局面から開始したものとやねうら王互角局面集で比較していますが(前者の方が良いだろう)想定と異なり有意な差は確認出来ませんでした。

### vs illqha4 (elmo から見て 勝 – 引分 – 負 )

- 平手局面： 301 - 27 - 295
- 互角局面集：301 - 29 - 291

### vs 水匠

- 平手局面： 812 - 59 - 718
- 互角局面集：260 - 15 - 239

### 対局条件

- 100 万ノード/手
- 2thread(vs 水匠), 4thread(vs illqha4)
- 探索部は dolphin1.01 を利用
- 投了値 -999
- 250 手で引き分け
- 定跡を利用する場合は 12 手まで
- 500 局以上(水匠戦だけ多いですが特に意味無いです)

対局は Apery の対局ツール(初期版)を独自に書き換えたものを用いています。(以前からですが)一般に公開されているレーティングと乖離が出ていて悩ましいです。投了値のせいなのか、スレッド数のせいなのか。探索ノード数のせいでは無さそうなのですが。

## 追試について

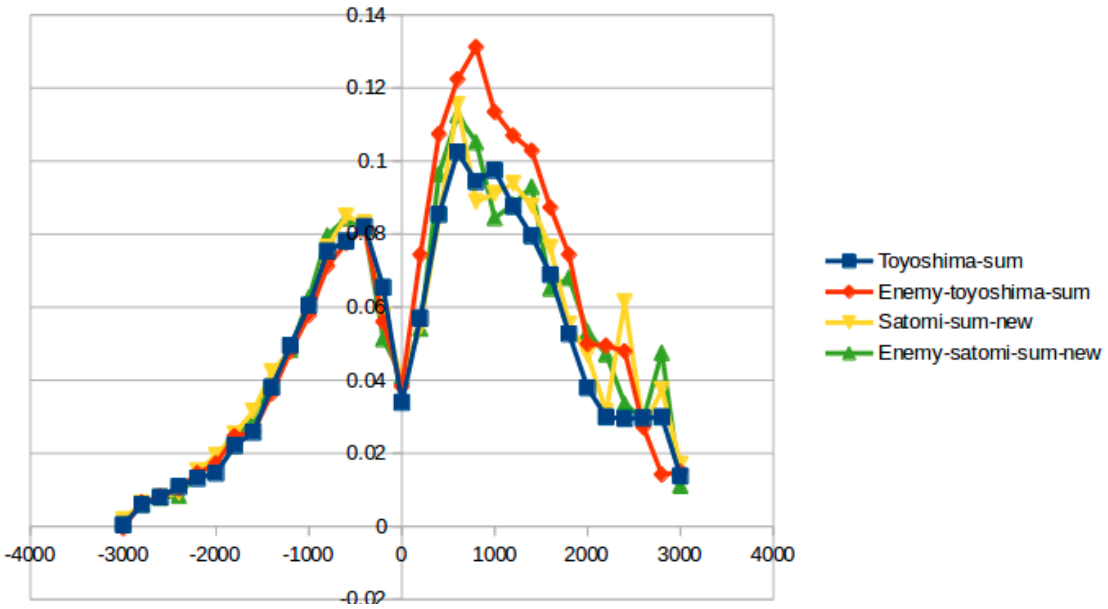
やねうら王は key コマンドのレスポンスを hex で返しているのですが、上記対局ツールでは数値で返す必要がありました。

Qhapaq di molto(QDM)の詳細技術文書
パッショーネ将棋チーム：Ryoto Sawada

１．開発動機
賞金を手に入れて参加費の元を取りたかった。

２．Qhapaq_di_molto の構成
探索部：github に公開されているやねうら王を利用(*1)。技術的独自性は特になし。
評価関数：github に公開されている NNUE 型評価関数(*1。tnk の実装をやねうら王にマージされたもの)を利用。少ない棋譜で過学習を起こさないように学習、解析を工夫している（後述）
並列計算：Hefeweizen が実装した Multiponder を拡張した Preponder 付き Multiponder を独自に実装(*2)
定跡：序盤で端歩をつくことをベースにしたものを自前で作成

３．技術的な独自性
３－１．評価値に応じた動的な学習率の調整
図は将棋棋士、及び将棋ソフトの悪手率（手を指した前と後の評価値の差分から与えられる、どのぐらい悪い手を指してしまったかの指標）を手を指す前の局面の評価値別にプロットしたものである。対局者によって山の高さには差があるものの、ソフト、人間双方とも、評価値 1000 前後で最も悪い手を指しやすいことが解った。評価関数の表現力に限界があると仮定した場合、より重要な局面を正しく評価できるようにする必要がある。そこで、QDM の学習では教師データの評価値に応じて学習率を動的に変えることで、重要な局面を効率的に学習させることを考えた。具体的には、学習率に d^2/dv (1/(1+exp(-v/600))) で定義されるボーナスを与えた。




また、今回の評価関数の教師データにはレート測定用の自己対局（４コア８スレッド１手１秒、定跡には tanuki-が公開している互角局面を利用）、及び、運営しているレーティングサイトに投稿された棋譜を用いた。教師データの総数は約１０００万局を用いた。

３－２．Preponder つき Multiponder
並列計算の手法としては Preponder つき Multiponder を用いた。従来の ponder は相手の手番中に相手が一番指しそうな手（候補手）についてのみ探索を行う。即ち、候補手が当たれば相手の思考時間を自分の持ち時間に実効的に加えることができるが、候補手が外れれば相手の思考時間中の読みは意味を失ってしまう。これに対し、Multiponder では複数のノードを用意しそ

れぞれに別の候補手を読ませることで ponder の一致率を上昇させている。Preponder は Multiponder の発展形であり、自分の手番中に、２手先の展開について予め他のノードに計算を行わせることで、自分の思考時間も ponder に用いようとするものである。

３－３．合法手列挙におけるソートの高速化
詳しくは https://github.com/TokusiN/SuperSort/ を参照

４．実験結果
強化学習について；
Qhapaq Research Lab で公開されている、orqha1018、及び、QDM の評価関数は前述の方法を用いて学習されている。学習元であった illqha（めきっと氏が開発。orqha の学習元）や t.n.k の評価関数よりも強い評価関数が得られている。
ただし、ボーナス値の付け方の最適化は行えていない。レート測定用の自己対局という深く正確な教師データを用いて、学習率を下げて微調整をしたことがレート上昇の本質的な要因である可能性も否定できない。

Preponder について：

下図は Qhapaq の自己対局で、通常の ponder を用いたもの vs 3 ノードを用意した Preponder を用いたものの自己対局で Preponder 側の ponder hit の内訳を分類したものである。似通った探索、評価部であれば、実効的な持ち時間を通常の Ponder に比べ増やすことができている。

| | Ponder | Multiponder | Preponder |
|---|---|---|---|
| 自分の思考時間 | 100 | 100 | 100 |
| 第一候補手がponder hitすることで得られた時間 | 50 | 50 | 50 |
| 第二、第三候補手がponder hitすることで得られた時間 | ー | 35 | 35 |
| Preponderによって得られた時間 | ー | ー | 35 |
| 思考時間の総和 | 150 | 185 | 220 |

Qhapaqの自己対局におけるPonderのhit率比較(値は相対的なもの)。ノード数は3

５．追試が可能であるかについて
学習については教師局面のシャッフルなどに乱数が伴うため、同じデータを作りなおすことは難しい。Preponder についてはソースコードを公開しているので、それを用いれば、上記実験結果を再現できる。

６．開発過程
2018/10 月頃まで　orqha を作る
2019/2 月頃　ライブラリルールの変化に伴い、tnk からの再学習が必要になる
2019/4 月頃　評価関数の学習の疲れからか Preponder を実装を開始する
2019/5/5　ソートが相当早くなる

# WCSC29 詳細アピール文書（PAL）

山口祐

## 1　開発動機

将棋ソフトの開発要素として主に探索、評価関数、定跡がある。一般的には、まず (1) 探索部の調整を行い、(2) 評価関数を学習し、最後に (3) 定跡を作成するという順番で実施される。しかしこれらの調整・学習は本来は密接に関係しており、例えば探索部の枝刈りやオーダリングに関するパラメータは、学習時の思考時間（探索ノード数）や評価関数によって最適値が異なると指摘されている。また、評価関数も特定の序盤局面に誘導することによって擬似的にレーティングが向上することがあり、評価関数の学習毎に局面が偏らないように定跡手順を作成し、それ以降の局面を適切に学習させる必要がある。

理想的には (1)(2)(3) を繰り返せばよいが、開発・計算コストが著しく増大する。そこで本ソフトではこれら 3 要素を並列的に学習させる Trinity Learning（三位一体学習）を提案し、学習効率を向上させることを試みた。具体的には学習中の探索部のパラメータ調整、学習時の局面選択におけるモンテカルロ木探索の導入とその定跡への転用を実装した。

## 2　開発過程

線形評価関数を採用する将棋ソフトの探索部は、チェスエンジンと同様に $\alpha\beta$ 法が主流となっている。探索部は指し手のオーダリングや枝刈りを、短時間の自己対戦で計測しながら調整していくのが一般的である。

これに対し、深層ニューラルネットワーク（DNN）で採用される評価値ベースのモンテカルロ木探索（Value-based Monte Carlo Tree Search、以下 MCTS）では、探索は方策関数の学習結果として自動的に調整される。また、DNN の方策関数は（雑に言うと）$\alpha\beta$ 法のプログロムでは探索済みの巨大な置換表にも相当し、初期局面から学習することで自然に定跡を包含することになる。したがって、線形評価関数でもこれに類する学習が実現すれば、探索・評価・定跡を同時に向上できると考えられる。

表1: 探索手法の特徴

| | $\alpha\beta$ 法 | DNN+MCTS |
|---|---|---|
| 探索速度 | 速い | 遅い |
| 枝刈り | 主に後向き | 前向き |
| 探索の改良 | 主に手調整 | 強化学習 |
| 定跡 | 別途作成 | 方策関数に内包 |

まず線形評価関数に MCTS を導入することを試みた（注：選手権には不採用）。KPPT 等の評価関数はそれ自体で深い探索に対して一手一致率が 40% 近くあるため、方策としても有望である。そこで YaneuraOu[*1] に通常の評価関数とは別にもう一つの評価関数（= 方策関数）を読み込み、局面評価時に同時に差分計算を行うようにした。さらに置換表ではなく探索木を直接保持し、末端ノード ($s_t$) で全合法手（子ノード $a_i|s_t$）を評価し、方策関数 $P$ の親ノードと子ノードの差を softmax 関数にかけたものを方策確率 $p$ として親ノードに登録するようにした。

$$p_i = \frac{exp(P(a_i|s_t) - P(a_{t-1}|s_{t-1}))}{\sum_j exp(P(a_j|s_t) - P(a_{t-1}|s_{t-1}))} \quad (1)$$

[*1] https://github.com/yaneurao/YaneuraOu

探索時には方策確率と評価値の統計値に基づき行動関数を計算し、どのノードを探索するかを決定する。子ノードに対して実際に探索した回数の分布 $y_i$ に方策関数を近づけるように、損失関数をクロスエントロピー ($L = -\sum y_i log(p_i)$) で規定できる。指し手 $a_i$ によって出現する特徴に対し、$L$ を最小にするよう方策関数を更新する学習部を実装し、WCSC28 の PAL の評価関数（KPPT）に Apery[*2]の評価関数で生成した教師データで追加学習させたものを用いて、方策関数を学習させた。

次に MCTS 探索とは別に、Stockfish 型の探索部を用いた場合の探索・評価関数・定跡の一体学習を試みた。探索パラメータの調整には Population Based Training[*3]を用い、一定の幅でパラメータをランダム初期化した 16 エージェントを並列に学習させながら、エージェント間のレーティングを計測し、レーティングが低いものを高いものに順次代替しながらパラメータを調整させた。

評価関数の学習では、一般的には棋譜や定跡からランダムな指し手を数手加えた局面から学習データを生成するのに対し、初期局面からの単一の探索木を学習を通じて保持し、末端ノードを順次学習の開始局面として学習データを生成する手法を採用した。ノード展開は探索深さ 12、multiPV=8 で最大の評価値から 200 以内の手を子ノードとして登録する。指し手は MCTS と同様、子ノードの中から評価値の平均値に探索回数によるペナルティを付与した行動関数によって選択され、末端ノードの評価値を順次上流のノードに伝播させ更新するようにした。これにより、棋譜や定跡から外れた局面の学習データを効率的に生成できる他、有力な手については優先的に探索が割り振られることが期待される。また、探索木は、有力な指し手ほど探索回数が多くなるので、適当な閾値を設定することで任意の局面の定跡として転用することができる。

Trinity Learning の学習は探索深さ 8-10 で生成した 2 億局面毎に 1 世代とし、評価関数の学習～レーティング計測～エージェント更新を KPPT および NNUE 型評価関数[*4]で 40 世代程度行った。学習環境は AWS EC2 c5.18xlarge 32 インスタンスを用いた。

## 3　結果・考察

学習方法と、前年度の PAL の評価関数・探索部とのレーティング差を表2に示す（表中のカッコ内は 95% 信頼区間）。レーティング計測は互角局面集[*5]から 20 ～ 60 手目の局面をランダムに選択し、1 スレッド 1 秒で 1000 対局程度行った。

表2: 開発手法とレーティング差

| 学習方法 | レーティング差 |
|---|---|
| MCTS 探索 | -265 (± 29) |
| TL + KPPT | +70 (± 22) |
| TL + NNUE | +139 (± 24) |

MCTS 探索は探索部のみの変更であるが、$\alpha\beta$ 法より明らかに弱くなった。原因して、方策関数の学習が進まなかったことが挙げられる。一手一致率は 40% 程度になったものの、必然手の方策確率が（特に終盤で顕著に）低くなった。このことから、線形関数だけでは方策確率を十分に表現できない可能性が示唆される。

一方、Trinity Learning で一体的に探索部・評価関数を学習させた結果、KPPT ・ NNUE 型評価関数ともに有意にレーティングが向上した。大会ではよりレーティングが高い NNUE 型評価関数を使用した。（追試については実施していないが、Population Based Training の性質上完全な再現は難しいと考えられる）

定跡については、特に相掛かり・角換わりで不利になる局面に飛び込む変化が散見された。そこで採用する探索回数の閾値を大きくし、角換わりを中心に 30 手程度までに抑えたものを選手権では使用した。（一部の変化については手動で延長して登録）

[*2] https://github.com/HiraokaTakuya/apery
[*3] https://arxiv.org/abs/1711.09846
[*4] tanuki-(WCSC28) を初期値に使用
[*5] https://github.com/tttak/ShogiGokakuKyokumen/

# 水匠アピール文書（２）

令和元年５月１３日

参加者　杉　村　達　也

## 第１　開発動機

今大会での目的は、３月２１日付「水匠アピール文書」第３に記載した工夫（教師局面の事後的変更）によって、NNUE 評価関数を有意に強くすることを目標としており、開発動機もそれに一致します。

## 第２　独自工夫の詳細

１　前後２手の評価値を見た評価値の修正

n 手目の評価値を Score(n)としたとき、①n 手目の手番側が負けており、かつ② Score(n) ＜ Score(n-2) ＜ Score(n+2) and Score(n-1) ＜ Score(n+1)となっている場合、Score(n)の値は間違っている（事後に反省をしている）可能性が高いと考えられるため、Score(n)を Score(n-2)の値まで引き上げるという修正をしました。

２　勝敗と評価値のずれの排除

勝利側の手番の評価値であるにも関わらず、負の評価値となっている場合、又は敗北側の手番の評価値であるにも関わらず正の評価値となっている場合、その局面の判断が間違っている可能性が高いため、教師局面から排除してしまうこととしました。その代わり、elmo 式学習の際の LAMBDA は概ね１としました（勝敗項を見ない学習、詳細は、やねうら王の learner.cpp を参照[1]）。

## 第３　実験結果（選手権後の事後的実験）

１　方法

（１）任意の NNUE 評価関数（以下の実験は、Qhapaq Research Lab(QRL)におけるトップレーティング付近の評価関数を使用しました。）を準備する[2]。

（２）当該評価関数から、ランダムムーブを入れず、depth18 で作成した検証用局面１（30 万局面、手数は 24 手目から記録）、及び初手から連続 10 手のランダムムーブを入れた、depth18 で作成した検証用局面２（5 万局面）を準備する。

（３）depth14 で作成した、教師局面約 4000 万局面を準備する。

（４）上記（１）乃至（３）の評価関数、検証用局面及び教師局面から、以下の３通りの学習方法を試行する（その他の条件は同じ（eta = 0.01 , batchsize 2000000 , nn_batch_size 2000 など。））。

①　通常の elmo 式学習（Lambda = 0.5）

②　通常の elmo 式学習（Lambda = 0.8）

③　本文書第２で記述した学習（Lambda = 1）

２　結果

（１）学習前評価関数

---

[1] https://github.com/yaneurao/YaneuraOu/tree/master/source/

[2] 今回は、https://www.qhapaq.org/shogi/ における illqha4 を使用しました。

検証用局面１： hirate eval = 13 , test_cross_entropy_eval = 0.590526 , test_cross_entropy_win = 0.519564 , test_cross_entropy = 0.555045 , norm = 1.53195e+08 , move accuracy = 37.7632%

検証用局面２： hirate eval = 13 , test_cross_entropy_eval = 0.230496 , test_cross_entropy_win = 0.156125 , test_cross_entropy = 0.193311 , norm = 1.4517e+08 , move accuracy = 35.9364%

（２）通常の elmo 式学習（Lambda = 0.5）

検証用局面１： hirate eval = 14 , test_cross_entropy_eval = 0.592798 , test_cross_entropy_win = 0.506909 , test_cross_entropy = 0.549854 , norm = 1.86824e+08 , move accuracy = 37.8687%

検証用局面２： hirate eval = 14 , test_cross_entropy_eval = 0.229684 , test_cross_entropy_win = 0.142708 , test_cross_entropy = 0.186196 , norm = 1.56518e+08 , move accuracy = 36.0255%

（３）通常の elmo 式学習（Lambda = 0.8）

検証用局面１： hirate eval = 21 , test_cross_entropy_eval = 0.589921 , test_cross_entropy_win = 0.510048 , test_cross_entropy = 0.549985 , norm = 1.76822e+08 , move accuracy = 38.0958%

検証用局面２： hirate eval = 21 , test_cross_entropy_eval = 0.22914 , test_cross_entropy_win = 0.147632 , test_cross_entropy = 0.188386 , norm = 1.52022e+08 , move accuracy = 36.1327%

（４）本文書第２で記述した学習（Lambda = 1）

検証用局面１： hirate eval = 41 , test_cross_entropy_eval = 0.590012 , test_cross_entropy_win = 0.50995 , test_cross_entropy = 0.549981 , norm = 1.78723e+08 , move accuracy = 38.1903%

検証用局面２： hirate eval = 41 , test_cross_entropy_eval = 0.229326 , test_cross_entropy_win = 0.148904 , test_cross_entropy = 0.189115 , norm = 1.48885e+08 , move accuracy = 36.4891%

３　まとめ

以上の実験結果から、本文書で提示した学習方法は、lambda を１と指定している（elmo 式を利用していない）にもかかわらず、elmo 式と同様に交差エントロピーを下げることができ、かつ move accuracy（検証用局面との最善手の一致率）を elmo 式よりも上昇させることができると考えられます（勝率の測定については、３月２１日付アピール文書のとおり。）。

４　追試について

本参加者の公開したデータに、教師局面変更のプログラム（Python）、水匠の評価関数、上記 Depth14 の 4000 万教師局面（シャッフル前）等を置いておきます[3]ので、ご興味がある方は、追試等いただけると幸いです。

以上

[3] https://1drv.ms/f/s!AjMACEoJwb5ngpY6NcU9qhdV5XS4UQ

# 「名人コブラ」アピール文書

## 概要

WSCS29版の名人コブラは、やねうら王をベースに改造されたコンピュータ将棋エンジンです。主な改造点はマルチポンダー機能の追加です。また、決勝進出ソフトで唯一、評価関数に3駒関係（KPPT型）を採用していました（開発スケジュールの遅れで、NNUE型評価関数の学習が間に合わなかったため）。

## 使用ライブラリとその選定理由

ベースのエンジンとしてはやねうら王をそのまま利用し、マルチポンダー機能はGUI（将棋所）とやねうら王の間に挟んだブローカープログラムで実装されています。

やねうら王の選定理由は以下の2点です:

- 昨年のWCSC優勝チーム他、多数のチームに採用されて大会での実績がある
- ソースコードにコメントが多数あり、理解しやすかった

その他、AperyとelmoをEval関数のブレンドに加えさせていただきました。開発スケジュールの遅れから、本年のWCSCで使用した評価関数は昨年のWCSC28版のApery、elmo、名人コブラの3種ブレンドという安直なものになってしまっています。

AperyとelmoのEval選定理由は以下の2点です:

- 単体の評価関数としてWCSC28での実績がある
- Apery系、やねうら王系とそれぞれ別々の系統の評価関数である

## 独自に工夫した点

今回、一番大きな改造点はマルチポンダー機能の実装です。これはPythonスクリプトによるGUIとエンジン（やねうら王）間のブローカーとして実装しました。この方式は比較的お手軽に実装できるのが良い点ですが、置換表が共有できないのが残念な点です。

実装にあたっては、元PonanzaチームのAki.さんが書かれた自己対戦用のRubyスクリプトを大変参考にさせていただきました。(https://github.com/ak110/USIGameRunner)

## 開発動機

元々コンピュータ将棋に取り組み始めたのは、今はなき電王トーナメントに参加して、女流棋士の先生のインタビューが受けたいといったよこしまな動機です。

元々負けず嫌いな性格なため熱くなってしまい、次はもっと良い成績をという事で参加し続けております。

大会に参加して様々な開発者の興味深いお話を聞かせていただけるというのも、開発や大会参加の動機の一つです。

## 開発過程

aki.さんの自己対戦用Rubyスクリプトを自分でPythonに移植したものを元々利用していたので、それをベースにマルチポンダー用ブローカーを開発しました。

費用上、開発時にクラウドのサーバを何台も借りるわけにはいかないので、localhostにSSHでつないで複数のエンジンを起動してテストをしました。

当初のアピール文書にはMCTSを利用した定跡作成や学習をうたっていたのですが、スケジュールの遅れで計算が間に合わず、今回は実戦投入できませんでした。

## 実験結果

元々テストが苦手な事と、複数サーバを借りるのにはコストがかかるため、まともな実験はできておりません。

ローカルマシンで簡易的にマルチポンダーの実験をしましたが、弱くはなっていないが、有意に強くなったとの確信も得られておりません。これは同条件のエンジンで戦わせるとシングルポンダーでもかなりの確率でヒットしてしまい、マルチポンダーのメリットが得られないためかと思われます。

## 追試可能か

評価関数とマルチポンダー用のブローカーを公開させていただきました。
https://github.com/hmatsuya/chotgun/releases/tag/wcsc29

やねうら王エンジンとクラウドのサーバを複数台（2次予選では5台、決勝では10台）ご用意いただければ、追試可能かと思われます。

## 謝辞

参加者の皆様との交流等、大会中は本当に楽しい経験をさせていただきました。運営の皆様、スポンサー様、他の参加者の方々には深く御礼申し上げます。