MC Softmax 探索における局面の並列評価：GPU とニューラルネットワークモデルの利用

吉野拓真 [1]　五十嵐治一 [1]　川島馨 [2]

概要：選択探索の一種として，モンテカルロソフトマックス探索が提案されている．一般に，大規模ニューラルネットワークモデルによる評価関数を利用する場合，計算に時間がかかることから，α β探索のような全幅探索よりは，選択探索の方が向いている． 特に，モンテカルロソフトマックス探索においては，兄弟ノード局面をまとめて評価する際に，GPU による並列計算を用いれば，評価関数が大規模なニューラルネットワークモデルであっても容易に並列化できる可能性がある．

本研究では，dlshogi のソースコードを改変し，モンテカルロソフトマックス探索とニューラルネットワークモデルの評価関数を組み合わせたプログラムを作成した．特に，ニューラルネットワークモデルの入力層に提示する局面の特徴量表現を工夫することにより，GPU で兄弟局面を同時に並列計算する際の処理時間を短縮することを試みた．さらに，ノード選択方策に Policy Network の出力値を取り入れることにより，探索精度の向上を試みた．

キーワード：コンピュータ将棋，モンテカルロソフトマックス探索，ニューラルネットワークモデル，dlshogi

Parallel State Evaluation in MC Softmax Search using GPU and Neural Network Models

TAKUMA YOSHINO[†1]　HARUKAZU IGARASHI[†2]
KAORU KAWASHIMA[†3]

Abstract: As a kind of selective search, the Monte Carlo softmax search has been proposed. In general, selective search is better than full-width search, such as the alpha-beta search, because of the computational time required to use the evaluation function of large-scale neural network models. In particular, in the Monte Carlo softmax search, parallelization can be easily achieved even in the case of large-scale neural network models by using GPU parallelization in the evaluation of sibling nodes.

In this study, we have modified the source code of dlshogi to create a program that combines Monte Carlo softmax search and evaluation functions of neural network models. We devised a feature representation of the input layer of the neural network model to reduce the processing time of the parallel computation for the sibling nodes on the GPU. Furthermore, we attempted to improve the accuracy of the search by incorporating the output of the Policy Network into a node selection policy.

Keywords: Computer shogi, Monte Carlo Softmax Search, Neural network model, dlshogi

1. はじめに

コンピュータ将棋のプログラムは，プロ棋士のレベルを遙かに超えるまでになってきた．これらの将棋ソフトの探索部には，伝統的に全幅探索（αβ 探索）が採用されてきた．一方で，コンピュータ囲碁ではモンテカルロ木探索（MCTS）[1]と呼ばれる選択探索の手法が大成功を収め，将棋へも適用されるようになってきた．AlphaZero や dlshogi がこの例である．

同じく選択探索の一種として，原らによってモンテカルロソフトマックス探索（MC Softmax Search，または MCSS）[2]が提案されている．全幅探索では各局面を一つずつ評価し，その結果を用いて逐次的に枝刈りを行うため，計算に時間がかかる大規模ニューラルネットワークモデル（NN）による評価関数の利用には難がある．一方，MCSS では，兄弟ノード局面をまとめて評価する際に，GPU による並列計算を用いれば，評価関数が大規模なニューラルネットワークモデルであっても容易に並列化できる可能性がある．

本研究では，MCTS と NN の評価関数により構成された dlshogi[3]のソースコードを改変し，MCSS と NN の評価関数を組み合わせたプログラムを作成した．特に，MCSS ではニューラルネットワークモデルの入力層に提示する局面の特徴量表現を工夫し，GPU で兄弟局面を同時に並列計算する際の処理時間を短縮することを試みた．さらに，ノード選択方策に Policy network の出力値を取り入れることにより，従来の MCSS よりも精度の良い探索を行うことを試みた．今回は実際に，これらの特徴量の改良とノード選択方策の改良の前後で，探索の各ステップにおける処理時間を計測したので結果を報告する．

1 芝浦工業大学
Shibaura Institute of Technology
2 HEROZ 株式会社
HEROZ, Inc.

2. 本研究で使用した dlshogi について

2.1 dlshogi とは

dlshogi は山岡が作成したプログラムである．MCTS に基づいた探索手法とニューラルネットワークを用いた評価関数によって構成される．本研究では書籍版[3]のソースコードを用いた．

書籍版の dlshogi は，Python とディープラーニングフレームワークである Chainer を使用して実装されている．なお，世界コンピュータ選手権等に参加している大会版の dlshogi は C++で実装されており，書籍版と比較して高速化と強化学習によるモデルの精度の向上が行われている．

2.2 dlshogi におけるモンテカルロ木探索

AlphaGo[4]と AlphaGoZero[5]では，MCTS に基づいた探索手法が用いられている．AlphaGo では，局面評価にプレイアウトと呼ばれる手法も用いているのに対し，AlphaGoZero では用いていない．したがって，AlphaGoZero の探索においては，ノード選択とバックアップの処理においてプレイアウトを用いた計算処理が除去されており，AlphaGo よりも処理が簡素化されている．本節では，以下，AlphaGo と AlphaGoZero の探索法について詳しく述べる．

まず，AlphaGo の探索手法は，以下の 4 ステップを繰り返す．

① 未展開のノードに達するまでノードを選択
② 未展開ノードの評価&プレイアウト
③ ②の子ノードをすべて生成
④ ①で選択した各ノードに，②で得た評価値とプレイアウト結果（勝敗）を加算（バックアップ）

AlphaGo の探索手法では，②の未展開ノードの評価に「プレイアウト」が用いられている．プレイアウトとは，未展開のノードから終局まであるシミュレーション方策に従ってプレイすることである．プレイアウトによって勝利した場合は 1 を，敗北した場合は 0 をプレイアウトの結果として記録する．このシミュレーション方策を学習するために，AlphaGo では，インターネット対局サイト「東洋囲碁」の強いプレイヤの 800 万局面棋譜を教師用学習データとするロジスティック回帰を行った．

AlphaGo における①では，以下の値が最大となるノードを選択する．

$$UCB(s,a) = (1-\lambda)\frac{w_v(s,a)}{N_v(s,a)} + \lambda\frac{w_r(s,a)}{N_r(s,a)} + c_{puct}P(s,a)\sqrt{\frac{\sum_b n(s,b)}{1+n(s,a)}} \quad (1)$$

ここで，$\mathrm{w_v(s,a)/N_v(s,a)}$は局面 s から指し手 a を指した後の局面の評価値平均，$\mathrm{w_r(s,a)/N_r(s,a)}$は局面 s から指し手 a を指した後の局面から終局までプレイアウトした結果の平均，$\mathrm{P(s,a)}$は Policy network の出力，$\mathrm{n(s,a)}$は局面 s から指し手 a を選択した回数を示す．λ，$\mathrm{c_{puct}}$は定数である．

一方，AlphaGoZero の探索手法は，以下の 4 ステップを繰り返す．

① 未展開のノードに達するまでノードを選択
② 未展開ノードの評価
③ ②の子ノードをすべて生成
④ ①で選択した各ノードに，②で得た評価値を加算（バックアップ）

この探索手法は，②においてプレイアウトの代わりに Value network のみによる局面評価を行う点で，AlphaGo とは異なる．そのため，①では，プレイアウト結果に関する(1)の第 2 項を除いた

$$UCB(s,a) = \frac{w_v(s,a)}{N_v(s,a)} + c_{puct}P(s,a)\sqrt{\frac{\sum_b n(s,b)}{1+n(s,a)}} \quad (2)$$

の値が最大となるノードを選択する．また，④でもプレイアウトの結果のバックアップは除かれている．

dlshogi では，上記 2 つのソフトのうち，AlphaGoZero の探索法を採用している．

2.3 ニューラルネットワークを用いた評価関数

dlshogi の評価関数には Policy network と Value network の二種類の NN を用いている．しかし，これらは AlphaGoZero と同様に，中間層を共有し，一つの深層ネットワークに統合されている．

Policy network では，ある局面 s の画像を入力とし，局面 s における各合法手の着手確率を予測して出力する．ネットワークの構成は，畳み込み 1 層の入力層，ResNet5 ブロック（畳み込み層 10 層），畳み込み 1 層の出力層により構成されている．

Value network では，ある局面 s の画像を入力とし，局面 s の評価値を出力する．出力層以外は Policy network と同じネットワークを用いており，出力層にはスカラー値を出力するための全結合層をつなげている．

これらの Policy network と Value network には，floodgate の 2016 年の棋譜 29758 局を学習データとした教師あり学習により得られた値が実装されている．

3. MCSS の探索法と評価局面の作成

3.1 MCSS の探索法

モンテカルロソフトマックス探索（MCSS）は，元々は，出現局面の合法手の最善応手手順の葉局面（principal leaf）だけではなく，探索木の leaf すべてを学習対象とするため

に，五十嵐ら[6]により提案された選択探索の一方式である．バックアップ操作を，minimax 演算ではなく，softmax 演算で行うという特徴がある．すなわち，ノード選択とバックアップの操作にはボルツマン分布を用いた計算が行われる．この分布関数を用いて，任意ノードからそれ以下の子孫ノードへの累積した選択確率（実現確率）や，その逆の祖先ノードへの累積したバックアップ確率（累積バックアップ確率）などを定量的に計算することができる．これらの確率値は，探索深さの評価[7]や探索制御[8]への応用のほか，複数の教師あり学習や強化学習を同時に実行する学習方式にも利用できる[9].

MCSS は，以下の４ステップを繰り返す[2].

① ルートノードから未展開ノードまでノード選択確率π_sに従ってノードを選択
② 未展開ノードに辿り着いたら子ノードをすべて生成
③ 生成した子ノードの評価値を計算
④ ①で選択した各ノードの評価値を更新（バックアップ）

①における，局面 s での指し手 a の選択確率$\pi_s(a|s)$は，次のように算出される．

$$\pi_s(a|s) = \exp\left(\frac{E(v(a;s))}{T_s}\right)/Z_s \quad (3)$$

$$Z_s \equiv \sum_{x\in A(s)} \exp\left(\frac{E(v(x;s))}{T_s}\right) \quad (4)$$

上式中の$E(v(a;s))$は，局面 s から指し手 a を指した後の局面$v(a;s)$の評価値，T_sは選択温度を表している．また，④における評価値の更新は次のように行われる．

$$E(s) = \sum_{x\in A(s)} \pi_b(x|s) E(v(x;s)) \quad (5)$$

$$\pi_b(a|s) = \exp\left(\frac{E(v(a;s))}{T_b}\right)/Z_b \quad (6)$$

$$Z_b \equiv \sum_{x\in A(s)} \exp\left(\frac{E(v(x;s))}{T_b}\right) \quad (7)$$

ここで，T_bはバックアップ用の温度パラメータ，π_bはバックアップ方策である．なお，$T_s = T_b$であれば，ノード選択方策(3)とバックアップ方策(6)は一致する．

3.2 MCTS と MCSS との相違点

dlshogi の探索法は，2.2 で述べた AlphaGoZero の探索手法（MCTS の一手法）をそのまま踏襲している．本節では，dlshogi の探索法（以下では MCTS）と MCSS との相違点について述べる．両者は，ノード選択，評価，バックアップの 3 つにおいて次のように異なる．

まず，ノード選択においては，MCTS では，式(2)の値が最大となるノードを決定論的に選択している点に対し，MCSS では，(3)のノード選択確率$\pi_s(a|s)$によって確率的にノード選択を行っている．

次に，局面評価においては，MCTS では②で未展開ノード 1 個のみを評価してから③でその子ノードをすべて生成する．一方，MCSS では②で辿り着いた未展開ノードのすべての子ノードの生成を行ってから，③でそれらの子ノード全てを評価する．すなわち，MCTS では未展開ノードを親ノードとすると,「親ノード一つのみを評価」してから子ノードをすべて生成するのに対して，MCSS は逆に親ノードを展開してから「子ノードをすべて評価」する．したがって，後者は兄弟ノードを並列的に評価できると効率が良い．

さらに，④のバックアップにおいては，MCTS では評価値の加算を行っている点に対し，MCSS では式(5)のようにバックアップ方策 π_b を用いた期待値操作による評価値の更新を行っている．後者の期待値操作は，子ノードが親ノードの評価値にどのような影響を与えているのかを定量的に表現しており，局面評価関数の学習において重要な役割を果たす[9].

3.3 ノード選択方策の改良：選択評価値

本研究では，局面 s での指し手 a に対する Policy network の出力$P(s,a)$を①のノード選択に積極的に活用し，棋力の向上を図る．この P(s,a)と評価値$E(v(a;s))$を用いて，次のような選択評価値$E_s(v(a;s))$を定義し，式(1)に適用する．

$$E_s(v(a;s)) = E(v(a;s)) + c_{puct} P(s,a) \sqrt{\frac{\sum_{b\in A(s)} n(s,b)}{1+n(s,a)}} \quad (8)$$

第二項は，(2)に示した AlphaGoZero や dlshogi と同様の項（PUCT）であり，局面 s からの指し手 a の選択回数$n(s,a)$が少なく，$P(s,a)$が高いほど，第二項の値は高くなる．尚，c_{puct}は定数である．

後述の 5.1 の対局実験で分かるように，Policy network の P(s,a)を除くと格段に棋力が低くなってしまう．これは MCTS でも同様である．

3.4 差分を用いた入力特徴の作成

dlshogi 及び本研究のプログラムで用いる入力特徴は，9×9 の二値画像 104 枚で構成される．すなわち，盤上の駒 14 種類の座標を 1 で表す二値画像 14 枚と，玉を除く 38 枚の駒に対し持ち駒となっていれば全て 1，持ち駒となっていなければ全て 0 で表す二値画像 38 枚の合計 52 枚を先手後手に対してそれぞれ生成する（図 1）．入力特徴作成の流れを以下の A)~E)に示す：

A) 以下を自分，相手の手番について行う
B) 盤上の駒 14 種について座標を調べる
C) 該当する座標のみ 1 となる二値画像を生成する

D) 駒 7 種 38 枚について持ち駒かどうかを調べる
E) 持ち駒に対して全て 1 の二値画像，それ以外に対して全て 0 の二値画像を生成する．

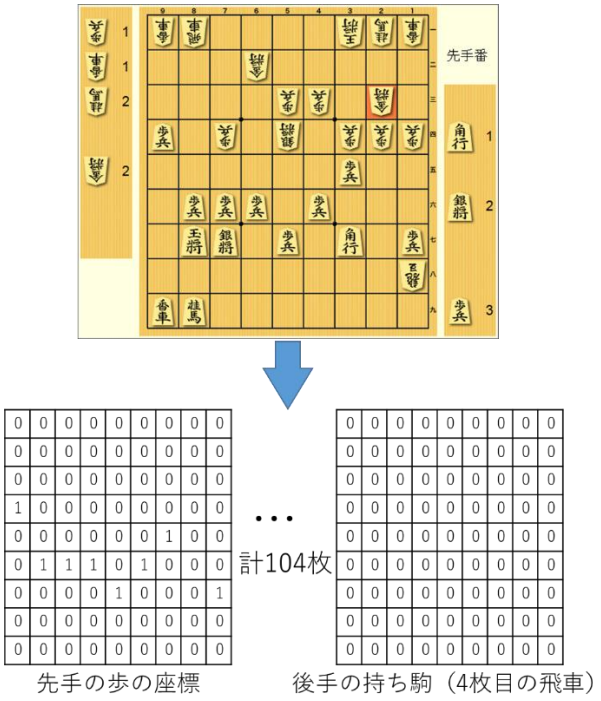

図 1　入力特徴の作成

3.1 で述べた③の評価値計算において，MCSS では複数の兄弟ノードをまとめて評価する．複数のノードをまとめて評価する場合，各ノードに対し，二値画像 104 枚分の入力特徴を作成する必要がある為，入力特徴の作成に時間を要する．しかし，兄弟ノードは，親ノードとの差分が最大でも盤上の駒 2 種及び持ち駒 1 枚分に限られる為，親ノードとの差分を利用することで，入力特徴の作成を効率化できると考えられる．

本研究では，上述の処理 C 及び E において，親ノードとの差分がある箇所に対してのみ二値画像を生成し，それ以外の箇所は，親局面の入力特徴をそのまま用いることで，二値画像の生成にかかる時間の短縮化を図った．

4. 探索の各処理時間の計測実験

4.1 実験条件と実験結果

dlshogi と探索の部分を MCSS に改造した本研究のプログラム 2 種（入力特徴の作成に差分を用いていないものと用いたもの）の 3 つのプログラムを用意する．

ここで，ルート局面からノード選択により末端ノードまで下りていく過程を「シミュレーション」と称する．実験では，ある一つの中盤局面（図 2）に対し，シミュレーション回数 1000 回，選択温度 0.05，バックアップ温度 0.001，c_{puct} 1.0 で探索の試行を 30 回ずつ行う．探索の各処理に要した時間を計測し，その結果（30 回試行の平均値）を表 1 に示す．「NPS」は，「評価局面数」を「探索時間」で割った値（1 秒あたりの評価局面数），「GPU へ転送」は入力特徴の GPU 上への転送時間，「CPU へ転送」は NN の出力値の CPU 上への転送時間，「現在局面の更新」は探索におけるノード遷移時間を示す．

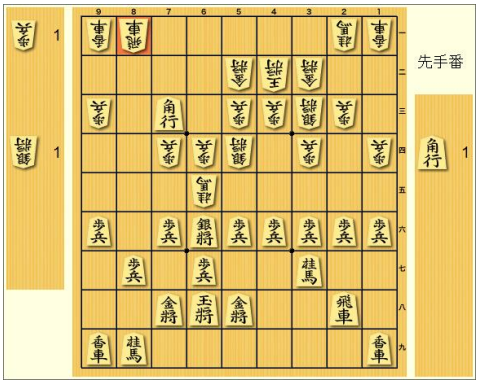

図 2　処理時間の計測に用いた中盤局面

表 1　評価局面数と処理時間の計測結果：ある局面から 1000 回シミュレーションした際の合計値で，30 回試行した値の平均値．

処理の種類	dlshogi	MCSS差分無	MCSS差分有
NPS	110	1777	3089
評価局面数	1000	84956	84956
探索時間	9.1	47.8	27.5
GPUへ転送（秒）	0.12	3.4	3.4
CPUへ転送（秒）	0.19	3.2	3.2
評価（秒）	4.3	4.6	4.6
子ノードの生成（秒）	3.6	3.6	3.6
入力特徴を作成（秒）	0.29	26.6	7.1
現在局面の更新（秒）	0.12	2.5	2.5

4.2 実験結果の考察

MCTS では 1 回のシミュレーションで 1 つのノードのみを評価するのに対し，MCSS では複数のノードをまとめて評価する．したがって，MCTS の dlshogi ではシミュレーション回数（1000）と評価局面数とは等しい．しかし，表 1 を見ると，MCSS の 2 つのプログラムでは，「評価局面数」は約 85 倍に増えており，NPS の値も約 17~30 倍に大きく

向上し，元の dlshogi よりは多くの局面を読んでいる．この時，評価局面数の増加に比べて，「探索時間」の増加の程度は低く，特に本論文で提案した MCSS（差分有）のプログラムでは，dlshogi の約 3 倍程度までに抑えられている．

しかし，評価局面数は増えてもトータルの「評価」時間は，3 つのプログラムにおいてほぼ同じ値であった．これは，ニューラルネットワークモデルの計算を GPU で並列化すれば，局面数が 1 でも 100 程度でも処理時間は変わらないことを示している．

一方で，局面数が多くなりデータ転送量が増えたので，入力特徴の「GPU への転送」時間や評価結果の「CPU への転送」時間は MCSS の方が約 17～28 倍と大きくなっている．「現在局面の更新」時間も同様である．

また，「入力特徴作成」に差分を用いたことで，入力特徴の作成時間には 26.6(s)から 7.1(s)と大幅な短縮に成功している．しかし，dlshogi の 0.29(s)と比較すると，作成局面が多いために約 24 倍以上の時間がかかっている．

以上をまとめると，CPU と GPU 間のデータ転送と， NN への入力データ作成とが MCSS にとってはボトルネックになっており，元の dlshogi と比べて探索時間が約 3 倍かかってしまっていることがわかる．

5. dlshogi（MCTS）との対局実験

5.1 予備実験

本節では，(2)と(8)の$P(s,a)$に Policy network の出力を用いることの効果を評価する予備実験を行った．すなわち，本研究のプログラムにおいて，3.3 の選択評価値において Policy network の出力を利用しないものを別途作成し，2 種類の本研究のプログラム同士で 100 局分の対局実験を行った．ただし，共にシミュレーション回数は 1000 回，バックアップ温度 0.001，選択温度は 0.05，c_{puct}は 1.0 とする．対局結果を以下に示す．

表 2　本プログラムにおける対局結果

	対PolicyNet有
PolicyNet無	8勝90敗2分

また，元の dlshogi において，Policy network の出力を用いず，式(2)の P(s,a)を，子ノード数 n を用いて，

$$P(s,a) = \frac{1}{n} \tag{9}$$

とするプログラムも別途用意し，2 種類の dlshogi 同士で 100 局分の対局実験を行った．対局結果を以下に示す．

表 3　dlshogi における対局結果

	対PolicyNet有
PolicyNet無	0勝100敗

どちらのプログラムにおいても，Policy network の出力を利用しないものは，利用しているものに大幅に負け越す，あるいは，まったく勝てない結果となった．したがって，(2)と(8)において，$P(s,a)$に Policy network の出力を用いることはきわめて大きな効果があることがわかる．逆に，$P(s,a)$の項がない単純な UCB を用いただけでは，ノード選択の精度はかなり低いと言える．

5.2 対局実験

本研究のプログラムと dlshogi による 100 局分の対局実験を行った．ただし，シミュレーション回数は 1000 回，バックアップ温度 0.001，c_{puct}は 1.0 とする．選択温度T_sは 0.05，0.02，0.001 の 3 通りのものを用意した．対局結果を以下に示す．

表 4　本プログラム(MCSS)と dlshogi との対局結果：
カッコ内は選択温度T_sの値を表している

	対dlshogi
MCSS（0.05）	24勝63敗13分
MCSS（0.02）	29勝55敗16分
MCSS（0.001）	46勝40敗14分

選択温度T_sを 0 に近づけ，ノード選択方策をミニマックスに近づけることで，dlshogi とほぼ互角の結果を残した．しかし，これは 3.3 の MCSS における確率的なノード選択を，MCTS の(2)のような決定論的なノード選択に近づけることに相当する．したがって，シミュレーション回数が同じであれば同じような探索結果になると考えられるので，ある意味当然の結果といえる．

今後は，温度を動的に変化させる確率的なノード選択方策，探索深さ情報の利用，バックアップ方策の改良による探索精度の向上と，ニューラルネットワークモデルの構成の改良による局面評価の精度向上と高速化や，GPU の利用法の改良などによる総探索時間の短縮化が必要である．

6. おわりに

本研究では，モンテカルロ木探索とニューラルネットワークモデルの評価関数により構成された dlshogi のソースコードを改変し，モンテカルロ・ソフトマックス探索とニューラルネットワークモデルの評価関数を組み合わせたプログラムを作成した．特に，ニューラルネットワークモデ

ルの入力層に提示する局面の特徴量表現を工夫し，GPU で兄弟局面を同時に並列計算する際の処理時間を短縮することを試みた．さらに，ノード選択方策に Policy network の出力値を取り入れることにより，従来のモンテカルロ・ソフトマックス探索よりも精度の良い探索を行うことを試みた．実際に，これらの特徴量の改良とノード選択方策の改良の前後で，探索の各ステップにおける処理時間を計測した．

その結果，ニューラルネットワークモデルによる複数局面の評価において GPU の並列化は効果があり，将棋の兄弟局面を処理時間を増やすことなく同時に評価できることが確認できた．また，ニューラルネットワークへの入力特徴量の改良と，Policy network の出力値をノード選択方策に利用することの有効性を対局実験により確認することができた．

今後は，ノード選択方策やニューラルネットワークモデルの改良，GPU と CPU 間のデータ転送などにおける処理時間の短縮について検討して行く予定である．

参考文献

[1] Coulom, R. . Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29–31, 2006.

[2] 桐井杏樹，原悠一，五十嵐治一，森岡祐一，山本一将．確率的選択探索の将棋への適用．ゲーム・プログラミング・ワークショップ 2017 論文集,2017, p. 26-33.

[3] 山岡忠夫．将棋 AI で学ぶディープラーニング．マイナビ出版, 2017.

[4] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[5] Silver, D., Schrittwieser, J., Simonyan, K. et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017). https://doi.org/10.1038/nature24270

[6] 五十嵐治一，森岡祐一，山本一将．方策勾配法による静的局面評価関数の強化学習についての一考察．ゲーム・プログラミング・ワークショップ 2012 論文集，2012, p. 118-121.

[7] 岩本裕大，五十嵐治一．コンピュータ将棋における MC Softmax 探索のための探索深さの指標．ゲーム・プログラミング・ワークショップ 2020 論文集（発表予定）.

[8] 原悠一，五十嵐治一，森岡祐一，山本一将，ソフトマックス戦略と実現確率による深さ制御を用いたシンプルなゲーム木探索方式，ゲーム・プログラミング・ワークショップ 2016 論文集，2016, p. 108-111.

[9] 五十嵐治一，森岡祐一，山本一将．MC Softmax 探索における局面評価関数の学習．ゲーム・プログラミング・ワークショップ 2018 論文集，2018, p. 212-219.