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

那須 悠 †

† ザイオソフト コンピュータ将棋サークル

2018 年 4 月 28 日

概要

現在のコンピュータ将棋ソフトの多くで使用されている評価関数である三駒関係は，高速な計算が可能であるが，非線形な評価を行うことができない．本稿では，CPU で高速に動作するニューラルネットワーク評価関数，「NNUE 評価関数」(Efficiently Updatable Neural-Network-based evaluation functions) を提案する．NNUE 評価関数は，CPU での演算に最適化した設計と，差分計算を始めとする高速化技術により，三駒関係と同等の実行速度で動作する非線形評価関数である．NNUE 評価関数を使用する初めての将棋ソフト『the end of genesis T.N.K.evolution turbo type D』は，第 28 回世界コンピュータ将棋選手権に参加を予定している．

NNUE: Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi

Yu Nasu†

†*Ziosoft Computer Shogi Club*

April 28, 2018

Abstract

Most of the strongest shogi programs nowadays employ a linear evaluation function, which is computationally efficient but lacks nonlinear modeling capability. This report presents a new class of neural-network-based nonlinear evaluation functions for computer shogi, called NNUE (Efficiently Updatable Neural-Network-based evaluation functions). NNUE evaluation functions are designed to run efficiently on CPU using various acceleration techniques, including incremental computation. The first shogi program with a NNUE evaluation function, *the end of genesis T.N.K.evolution turbo type D*, will be unveiled at the 28th World Computer Shogi Championship.

0. まえがき

本稿は，第 28 回世界コンピュータ将棋選手権 [1] に参加を予定しているソフト『the end of genesis T.N.K.evolution turbo type D』のアピール文書の一部である．同ソフトの評価関数に関する技術を論文風の体裁で説明するが，実験は掲載しない．同ソフトの具体的な棋力にも本稿では言及しないので，選手権の結果を参照されたい．

1. はじめに

コンピュータ将棋ソフトの指し手の決定は，主として形勢判断に相当する「評価関数」と，読みに相当する「探索」によって行われる．高い棋力を実現するためには，評価値の精度が高く，高速に計算できる評価関数を用いることが望ましい．評価値の精度が高ければ正確な形勢判断ができ，計算が高速であればより深く読むことができるためである．

2018 年 3 月現在，コンピュータ将棋ソフトで使用される最も代表的な評価関数の方式は，「三駒関係」と呼

ばれる線形モデルである．機械学習によって駒 3 つの位置関係に重みを付けておき，局面上に出現する位置関係に付けられた重みの総和を評価値とする．2003 年に初めてアイデアが示され [2]，2009 年に玉を含む組み合わせに限定して実装した『Bonanza』 [3] のソースコードが公開されて以降，広く用いられるようになった．2014 年に『NineDayFever』 [4] で導入された手番評価など，いくつかの拡張を経て，トップクラスの棋力を持つ将棋ソフトのほとんどで現在も採用されている．

三駒関係の優位性は，膨大な数のパラメータによって表現される評価関数でありながら，高速に評価値を算出できる点にある．ある局面から 1 手指した先の局面の評価値を求めるとき，移動した駒に関する重みを元の局面の評価値に足し引きする差分計算によって，全体を計算した場合と同じ結果を，遥かに効率的に求めることができる．差分計算の容易さは，評価関数に線形モデルを採用することによってもたらされる大きな利点である．

しかし，線形モデルであることは，評価関数の表現能力の足枷にもなっている．特定の駒の位置関係が，ある状況においては有利に働くが，別の状況では無駄になる，といった非線形な評価を，三駒関係では直接表現することができない．線形モデルのまま表現能力を高めるためには，評価項目を変える必要があるが， 2018 年 3 月現在，三駒関係より優れた線形モデルは知られていない．

より高い表現能力を持つ評価関数の実現を狙ったアプローチとして，非線形モデル，特にニューラルネットワークの利用も試みられている．先駆けとなったのは『習甦』 [5] で，自玉および相手玉の安全度によって駒の価値を重み付けした，非線形な評価関数を特徴とする将棋ソフトである．近年では，コンピュータ囲碁において，大規模な CNN (Convolutional Neural Networks) による評価関数を用いたソフトが成功を収めた [6] ことから，コンピュータ将棋でも CNN を利用する試みが行われている．しかし， CNN による評価関数を活用するためには，線形モデルよりも大きな計算リソースが必要となることが課題である．CNN を利用する将棋ソフトのほとんどは，並列演算を得意とする GPU を使用し，複数の局面をまとめて評価するバッチ処理を行って高速化している．それにもかかわらず， 2018 年 3 月現在，ニューラルネットワークに最適化された特殊なハードウェアを使用する『AlphaZero』 [7] を除いて，トップクラスの将棋ソフトに比肩する棋力を実現した例は報告されていない．

本稿では， CPU で高速に動作するニューラルネットワーク評価関数を提案する．CPU での演算に最適化した設計と，差分計算を始めとする高速化技術により，複数の隠れ層を持つニューラルネットワーク評価関数を，三駒関係と同等の実行速度で動作させることができる．以降，この方式の評価関数を「NNUE 評価関数」 (Efficiently Updatable Neural-Network-based evaluation functions) と呼ぶことにする．

2. NNUE 評価関数

NNUE 評価関数は， GPU もバッチ処理も用いず，CPU で 1 局面ずつ評価するニューラルネットワーク評価関数である．三駒関係と同様， 1 局面を評価するまでのレイテンシが小さく，枝刈りを伴うミニマックス系の探索手法と組み合わせやすい．

『the end of genesis T.N.K.evolution turbo type D』に搭載する NNUE 評価関数の模式図を図 1 に示す．CPU で高速に計算するために設計した構造であり，『AlphaZero』などで採用されているニューラルネットワークの構造とは大きく異なる．

以下では， NNUE 評価関数の設計と高速化技術について述べる．

2.1 全結合ニューラルネットワーク

将棋盤の二次元構造を利用する CNN ではなく，全結合ニューラルネットワークを使用する．主な利点は，メモリアクセスパターンが単純で高速化しやすいことと，後述する差分計算が容易に実現できることの 2 点である．

メモリアクセスパターンは，三駒関係より処理効率が高い評価関数を設計するために重要な要素である．三駒関係では，評価値を算出するためにかかる時間の大半を，メモリへのランダムアクセスが占めている．効率的なメモリアクセスパターンを用いれば，同じ時間でより複雑な計算が可能になる．

全結合ニューラルネットワークによる評価値の計算を次式に示す．入力となる特徴ベクトルを $\boldsymbol{x}$，隠れ層の数を L，層 l の重み行列を $\boldsymbol{W}_l$，層 l のバイアスベクトルを $\boldsymbol{b}_l$，活性化関数を σ とすると，評価値 y は

$$[y]_{1\times 1} = \boldsymbol{z}_{L+1} \tag{1}$$

$$\boldsymbol{z}_l = \boldsymbol{b}_l + \boldsymbol{W}_l \boldsymbol{a}_{l-1} \tag{2}$$

$$\boldsymbol{a}_l = \begin{cases} \sigma\left(\boldsymbol{z}_l\right) & (\text{if } l > 0) \\ \boldsymbol{x} & (\text{if } l = 0) \end{cases} \tag{3}$$

と求められる．

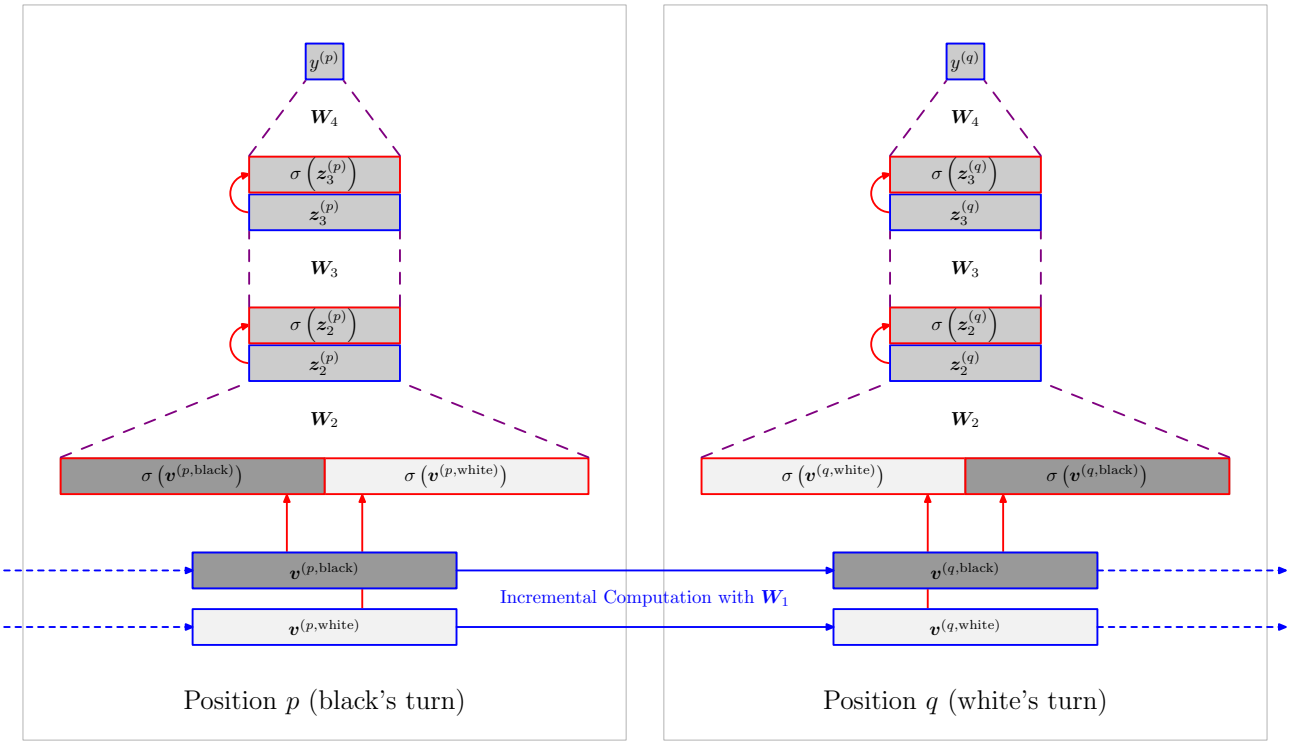

図 1　『the end of genesis T.N.K.evolution turbo type D』の評価関数．局面 p から 1 手指した先の局面 q について評価値を求めるとき，処理の一部で差分計算を利用する．評価には手番が考慮されている．

2.2　活性化関数

活性化関数には，層 l のユニット数を N_l として，次式で表される clipped ReLU を用いる．

$$\begin{aligned}\sigma\left(\boldsymbol{z}_l\right) &= \sigma\left(\begin{bmatrix} z_{l,1} \\ z_{l,2} \\ \vdots \\ z_{l,N_l} \end{bmatrix}_{N_l \times 1}\right) \\ &= \begin{bmatrix} \sigma(z_{l,1}) \\ \sigma(z_{l,2}) \\ \vdots \\ \sigma(z_{l,N_l}) \end{bmatrix}_{N_l \times 1}\end{aligned} \tag{4}$$

$$\sigma\left(z_{l,i}\right) = \begin{cases} 0 & (\text{if } z_{l,i} \leq 0) \\ z_{l,i} & (\text{if } 0 < z_{l,i} < 1) \\ 1 & (\text{if } z_{l,i} \geq 1) \end{cases} \tag{5}$$

値域が有限であるため整数化に適しており，後述する SIMD 演算によって効率的に計算できることが利点である．

2.3　特徴量

特徴ベクトルには，原則として，各要素が 0 または 1 の値を取る二値ベクトルを使用する．この制約も差分計算を容易にするためである．高速に差分計算を行うためには，1 手指す前後の局面で値が変化する要素の数が少ないことが望ましい．

このような性質を満たす特徴量の例としては，自玉または敵玉と，玉以外の駒 1 つとの位置関係を表す KP (King-Piece) が挙げられる．『the end of genesis T.N.K.evolution turbo type D』で実際に採用している特徴量は，KP をベースとして改変したものである．詳細は後述する．

2.4　アフィン変換とメモリレイアウト

式 (2) のようなアフィン変換を行うためには，通常，行列の行とベクトルの内積を計算するのが効率的である．$\boldsymbol{W}_l$ の i 行目を $\boldsymbol{W}_l(i,:)$ として，

$$\boldsymbol{z}_l = \begin{bmatrix} z_{l,1} \\ z_{l,2} \\ \vdots \\ z_{l,N_l} \end{bmatrix}_{N_l \times 1} \tag{6}$$

$$z_{l,i} = b_{l,i} + \boldsymbol{W}_l(i,:)\boldsymbol{a}_{l-1} \tag{7}$$

と計算する．行列の行の要素を連続して参照するメモリアクセスパターンであるから，メモリレイアウトは row-major とする．

しかし，ベクトル $\boldsymbol{a}_{l-1}$ がスパースである（要素の大

部分が 0 である）場合は，行列の一部の列だけを用いて，次式のように計算する方が高速になる．$\boldsymbol{W}_l$ の j 列目を $\boldsymbol{W}_l(:,j)$ として，

$$\boldsymbol{z}_l = \boldsymbol{b}_l + \sum_{j \in \{j | a_{l-1,j} \neq 0\}} a_{l-1,j} \boldsymbol{W}_l(:,j) \tag{8}$$

と計算する．この場合は，行列の列の要素を連続して参照するメモリアクセスパターンになり，メモリレイアウトを column-major とする方が効率が良い．

NNUE 評価関数では，特徴ベクトル $\boldsymbol{x}$ のアフィン変換

$$\boldsymbol{z}_1 = \boldsymbol{v} \tag{9}$$

$$\boldsymbol{v} = \boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{x} \tag{10}$$

を，式 (8) の方法によって計算する．前述の通り，$\boldsymbol{x}$ は二値ベクトルである．$\boldsymbol{x}$ がスパースであれば，式 (8) によってアフィン変換を高速に計算できる[*1]．さらに，二値ベクトルであることを利用すると，式 (8) において $a_{l-1,j} = x_j \in \{0,1\}$ であるので，

$$\boldsymbol{v} = \boldsymbol{b}_1 + \sum_{j \in \{j | x_j = 1\}} \boldsymbol{W}_1(:,j) \tag{11}$$

と単純化できる．

2.5 差分計算

差分計算は，ある局面から 1 手指した先の局面の評価値を計算する際に，元の局面との差異が少ないことを利用して高速化する方法である．NNUE 評価関数では，局面 q の特徴ベクトル $\boldsymbol{x}^{(q)}$ をアフィン変換した結果のベクトル

$$\boldsymbol{v}^{(q)} = \boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{x}^{(q)} \tag{12}$$

を，1 手前の局面 p について計算済みの $\boldsymbol{v}^{(p)}$ を利用して差分計算する．具体的には，次式によって $\boldsymbol{v}^{(q)}$ を求める．

$$\begin{aligned} \boldsymbol{v}^{(q)} = \boldsymbol{v}^{(p)} &- \sum_{j \in \left\{ k \middle| x_k^{(p)}=1 \wedge x_k^{(q)}=0 \right\}} \boldsymbol{W}_1(:,j) \\ &+ \sum_{j \in \left\{ k \middle| x_k^{(p)}=0 \wedge x_k^{(q)}=1 \right\}} \boldsymbol{W}_1(:,j) \end{aligned} \tag{13}$$

これは，線形モデルの差分計算をベクトルに拡張した式であると解釈することができる．

[*1] 実際には差分計算を行うので，1 手指す前後の特徴ベクトルの変化（ハミング距離）が小さいことが重要であり，スパースである必要はない．

差分計算を利用するのは特徴ベクトルのアフィン変換だけで，それ以外の部分では利用しない．

2.6 手番評価

将棋においては，駒の配置が同じであっても，手番がどちらにあるかによって形勢は異なるので，評価関数も手番を考慮する方が高精度になる．三駒関係では，線形性を利用して評価値を 2 つの項に分解し，手番に依存しない項と手番に依存する項の和で表現している [4]．

NNUE 評価関数では，常に手番側の視点から見た局面を評価することによって手番評価を実現する．まず，単純な方法について説明する．局面 p の手番側プレイヤーを active(p)，プレイヤー c から見た局面 p の特徴ベクトルを $\boldsymbol{x}^{(p,c)}$ として，

$$\boldsymbol{z}_1^{(p)} = \boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{x}^{(p,\mathrm{active}(p))} \tag{14}$$

とすると，手番側の視点から見た局面の評価が実現できる．

手番がどちらにあるかによって特徴ベクトルが変わるため，差分計算は，先手（black）および後手（white）の両方の視点について行う必要がある．1 手指すごとに，$c \in \{\mathrm{black}, \mathrm{white}\}$ に対して，それぞれ

$$\boldsymbol{v}^{(p,c)} = \boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{x}^{(p,c)} \tag{15}$$

を差分計算しておく．評価値の計算には手番側を使用し，

$$\boldsymbol{z}_1^{(p)} = \boldsymbol{v}^{(p,\mathrm{active}(p))} \tag{16}$$

とする．

次に，この方法を拡張して，非手番側のベクトルも評価値の計算に活用する方法を説明する．局面 p の非手番側のプレイヤーを opponent(p) として，式 (16) の代わりに次式を用いる．

$$\begin{aligned} \boldsymbol{z}_1^{(p)} &= \begin{bmatrix} \boldsymbol{v}^{(p,\mathrm{active}(p))} \\ \boldsymbol{v}^{(p,\mathrm{opponent}(p))} \end{bmatrix}_{2N_1 \times 1} \\ &= \begin{cases} \begin{bmatrix} \boldsymbol{v}^{(p,\mathrm{black})} \\ \boldsymbol{v}^{(p,\mathrm{white})} \end{bmatrix}_{2N_1 \times 1} & (\text{if active}(p) = \text{black}) \\ \begin{bmatrix} \boldsymbol{v}^{(p,\mathrm{white})} \\ \boldsymbol{v}^{(p,\mathrm{black})} \end{bmatrix}_{2N_1 \times 1} & (\text{if active}(p) = \text{white}) \end{cases} \end{aligned} \tag{17}$$

すなわち，手番側と非手番側の特徴ベクトルをそれぞれアフィン変換し，結合したベクトルを $\boldsymbol{z}_1^{(p)}$ とする．アフィン変換のパラメータ $\boldsymbol{b}_1, \boldsymbol{W}_1$ は共有されるので，こ

の計算は手番方向の畳み込みであると解釈できる．

図 1 に示した『the end of genesis T.N.K.evolution turbo type D』の評価関数は，式 (17) の方式を採用している．

2.7 HalfKP 特徴量

前述の通り，『the end of genesis T.N.K.evolution turbo type D』では，二駒関係の一種である KP を改変した特徴量を採用している．KP が「自玉または敵玉と，玉以外の駒 1 つとの位置関係」を表す特徴量であるのに対して，敵玉を含む位置関係を使用せず，自玉を含む位置関係だけを使用する．以下では，この特徴量を「HalfKP」と呼ぶことにする．

HalfKP 特徴量は，前項で説明した，手番側と非手番側の特徴ベクトルを両方とも使用する手法と組み合わせるための特徴量である．HalfKP 特徴量を使用すると，式 (17) において， $\boldsymbol{v}^{(p,\mathrm{active}(p))}$ は「手番側から見た，手番側の玉に関する KP」に相当する情報を持ち，$\boldsymbol{v}^{(p,\mathrm{opponent}(p))}$ は「非手番側から見た，非手番側の玉に関する KP」に相当する情報を持つことになる．従って，特徴量そのものには敵玉の位置に関する情報を含まないが，手番側から見た特徴量と，非手番側から見た特徴量の両方を使用することによって，局面全体の情報を表現することができる．通常の KP を特徴量として使用する場合と比較して，同等の情報を表現するために必要な計算コストが小さくなることが利点である．

2.8 整数 SIMD 演算

SIMD (Single Instruction Multiple Data) 演算は，同一の処理を複数のデータに同時に適用する演算である．NNUE 評価関数は，各部分において整数の SIMD 演算を活用して高速に計算できるように設計されている．

特徴ベクトルのアフィン変換は，差分計算を行う場合も行わない場合も，ベクトルの加減算で求められる．重み行列 $\boldsymbol{W}_1$ を 16-bit の整数に変換して保持し， 16-bit の符号付き整数ベクトルの加減算を行う命令（AVX2 では `VPADDW`, `VPSUBW`）で計算する．

特徴ベクトル以外のアフィン変換では，直前の層のアクティベーション $\boldsymbol{a}_{l-1}$ と重み行列 $\boldsymbol{W}_l$ を 8-bit で表現し， 8-bit 整数ベクトル 2 つの積和を求める命令（`VPMADDUBSW`）などを用いて内積を計算する．

活性化関数は，整数のベクトルのビット幅を変換する命令（`VPACKSSDW`, `VPACKSSWB`）と， 8-bit 整数ベクトル 2 つの要素ごとの最大値を取る命令（`VPMAXSB`）などによって計算する．

表 1 『the end of genesis T.N.K.evolution turbo type D』の評価関数で使用する重み行列のサイズ．

	列数	行数
$\boldsymbol{W}_1$	125388	256
$\boldsymbol{W}_2$	512	32
$\boldsymbol{W}_3$	32	32
$\boldsymbol{W}_4$	32	1

2.9 モデルサイズ

表 1 に，『the end of genesis T.N.K.evolution turbo type D』の評価関数で使用する重み行列のサイズを示す．列数と行数が，それぞれアフィン変換の入力と出力の次元数に対応する．このモデルサイズは，単位時間あたりに読める局面数が，評価関数に三駒関係を使用した場合と同等になるように設定したものである．

全パラメータ数の大半を，特徴ベクトルのアフィン変換に使う重み行列 $\boldsymbol{W}_1$ が占めている．この部分は差分計算を行うため，実際の計算コストは大きくない．$\boldsymbol{W}_2, \boldsymbol{W}_3, \boldsymbol{W}_4$ を使うアフィン変換は差分計算ができないが，パラメータ数を比較的少なく設定し， 8-bit 整数の SIMD 演算によって高速な計算を可能にしている．

3. おわりに

本稿では，ニューラルネットワークを用いた将棋の評価関数を提案し， CPU で高速に動作させるための設計と高速化技術について述べた．本技術による評価関数を搭載する初めての将棋ソフト『the end of genesis T.N.K.evolution turbo type D』は，第 28 回世界コンピュータ将棋選手権 [1] に参加を予定している．

また，本技術を実装したソースコードを後日公開する予定である．オープンソースの将棋ソフト『やねうら王』[8] をベースとして C++ で実装したもので，本稿で述べた内容に加えて，以下のような特徴がある．

- クラステンプレートを利用した，特徴量やネットワーク構造のカスタマイズ性
- コンパイル時計算やコンパイラによる最適化を活用した高速な実装
- 三駒関係の学習で用いられる手法と同様の，
  - 静止探索と組み合わせた学習
  - 特徴量の分解による学習時のパラメータ共有

特徴量やネットワーク構造は，『the end of genesis T.N.K.evolution turbo type D』で採用したものが最適

であるとは限らない．より優れた構造の検討は今後の課題であり，ソースコードを拡張して利用される他の開発者にも期待したい．

本技術がコンピュータ将棋の更なる発展に繋がれば幸いである．

参考文献

[1] 第 28 回世界コンピュータ将棋選手権. http://www2.computer-shogi.org/wcsc28/.

[2] 金子知適, 田中哲朗, 山口和紀, 川合慧. 駒の関係を利用した将棋の評価関数. ゲームプログラミングワークショップ 2003 論文集, pp. 14–21, 2003.

[3] Kunihito Hoki and Tomoyuki Kaneko. Large-scale optimization for evaluation functions with minimax search. *Journal of Artificial Intelligence Research*, Vol. 49, No. 1, pp. 527–568, 2014.

[4] 金澤裕治. NineDayFever アピール文書. http://www.computer-shogi.org/wcsc24/appeal/NineDayFever/NDF.txt, 2014.

[5] 竹内章. コンピュータ将棋における大局観の実現を目指して. 人工知能学会誌, Vol. 27, No. 4, pp. 443–448, 2012.

[6] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, Vol. 529, No. 7587, pp. 484–489, 2016.

[7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv:1712.01815v1 [cs.AI], 2017.

[8] 磯崎元洋. やねうら王オープンソースプロジェクト. http://yaneuraou.yaneu.com/yaneuraou_mini/.