Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi[1]

Yu Nasu
Ziosoft Computer Shogi Club
April 28, 2018

Abstract

Sankoma-Kankei (lit. „Three-Piece-Relationship“) which is used in todays computer Shogi programs as an evaluation function can be calculated very fast, but is unable to capture non-linear relationships.

This paper describes a neural-network based evaluation function dubbed „Efficiently Updatable Neural-Network-based evaluation functions“ (NNUE) which runs fast even on a CPU.

The NNUE evaluation function achieves similar performance compared to the Sankoma-Kankei evaluation function through optimization to run on CPUs as well as acceleration by using a difference-based computation.

„the end of genesis T.N.K.evolution turbo type D“ ist he first Shogi program that implements this evaluation function and will take part in the 28. Shogi computer Shogi chess championship.

Preface

This paper describes the software program „the end of genesis T.N.K.evolution turbo type D“, which will take part in the 28. Shogi computer championship. Described ist the used evaluation function; experimental evaluations are not contained in this paper. The abilities or

[1] Translated from Japanese by Dominik Klein (First . LastName at outlook . com)

general strength of the program is also not the topic of this paper; here we refer to the results of the championship.

Introduction

Choosing a move in computer shogi is mainly determined by the evaluation function which evaluates the current board position as well as a search function. To get a strong program, it is required to use an evaluation function that is delivers a good evaluation but is also fast to calculate. The more precise the evaluation, the stronger the program plays, because if the evaluation function is precise, a better judgement oft he current board position is given.

If the evaluation value is precise then the current state of the game can be judged precisely, and if the evaluation is fast, it is possible to search deeper. The reason ist that the evaluation will be better both if the evaluation of a position is more precise but also if the calcuation is faster.

As of March 2018, the current state of the art for the evaluation function used in computer Shogi is a linear model, that is dubbed „Sankoma-Kankei“ (lit. “Three-Piece-Relationship”). The idea was first presented in 2003 [2], and since the source code of „Bonanza“ was published in 2009, which used it for piece combinations(?), it has been widely adopted. After some extensions, like the move-evaluation that was introduced in 2014 in "NineDayFever" [4], it is still the commonly used evaluation function of the strongest computer Shogi software programs.

The advantage of the Sankoma-Kankei evaluation function ist hat the function is determined by a large number of parameters, yet the evaluation value can be calculated very quickly. During the calculation of a evaluation value for a position which is just one move away from another position, and for which the evaluation value is already known, we can – instead of recalculating the evaluation value from scratch – more efficiently compute the value. Namely, by calculating the difference between the evaluation value of the original position and the other position by addition or subtraction oft he piece values oft he moved piece(s). This simplicity of the difference calculation is a major advantage when using such kind of linear model as the evaluation function.

The fact that this is a linear model however makes it difficult to increase the expressiveness of the evaluation function. Non-Linear relationships, like the piece-relationships of certain pieces, which in some situations are more of an advantage, and in some other positions not, cannot be expressed directly through the Sankoma-Kankei relationship. In order to increase the expressiveness while keeping a linear model, one would require to use different evaluation values, but as of March 2018 no better linear model than Sankoma-Kankei has been found.

Using non-linear models, especially neural-networks has been tried as an attempt to realize evaluation function with higher expressiveness. A pionier of this approach was "Naruso" [5], a Shogi software with a non-linear evaluation function which weights the value of a piece w.r.t. the own's and enemy's King safety. In the last years, a very large CNN-based evaluation function was very successful for computer Go [6], and several attempts have been made to also use it for Shogi. Most Shogi programs that use CNNs employ GPUs which are good at parallel computing and batch processing, in order to evaluate several positions at once to increase speed.

Still though, until March 2018 – with the exception of "AlphaZero" [7], which used specially hardware optimized of neural networks, there is no reported example for a software implementation which has a strength that is comparable to the best (traditional) Shogi software.

In this paper we propose an evaluation function based on neural networks, which can be executed on a CPU with high speed. This is since CPU based optimization as well as differential computations enable to construct the evaluation function as a neural network with several hidden layers, yet having a similar execution speed as the Sankoma-Kankei evaluation function. We call this evaluation function “NNUE” (Efficiently Updateable Neural Network based Evaluation Function).

2. NNUE Evaluation Function

The NNUE evaluation function is a neural network based function, which evaluates one game position on a CPU without the need of a graphics processor or batch processing. Similar to the Sankoma-Kankei Evaluation Function the calculation time for evaluating one position is rather small, and such it can be combined easily with minimax-search and branch pruning. Figure 1 shows a schematic diagram of the NNUE evaluation function used in "the end of genesis T.N.K.evolution turbo type D". This structure is suited for high-speed calculation ona CPU, and differs from neural network structure used in“AlphaZero”. In the following, we will describe the design of the NNUE evaluation function as well as acceleration techniques.

2.1 Fully-Connected Neural Network

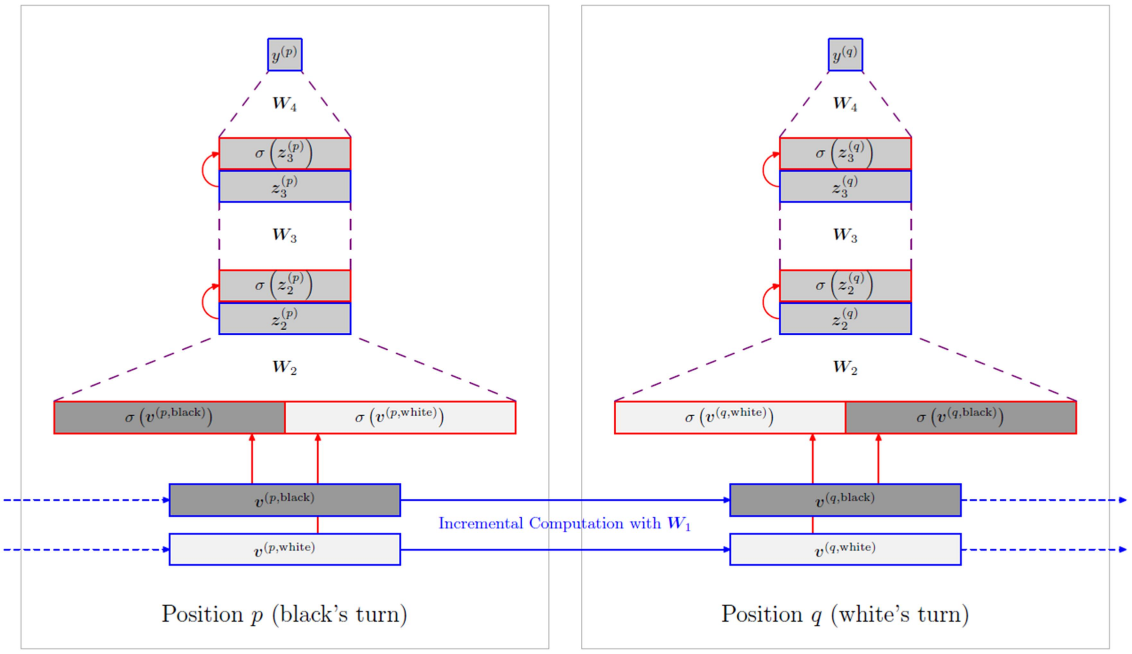
Used here is a fully connected neural network instead of a convolutional neural network (CNN). The network uses the two-dimensional structure of a Shogi board. The main advantage is that memory access patterns are easy to accelerate, and the difference computation, that is described below, is easy to realize.

The following equation shows the calculation of the evaluation value by the fully connected neural network. Let the input vector be denoted by x, the amount of hidden layers by L, the weight matrix at layer l by W_l, and the bias vector of layer l by b_l, and the activation function by σ. Then the evaluation value y is defined as

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

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

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



$y^{(p)}$
W_4
$\sigma\left(z_3^{(p)}\right)$
$z_3^{(p)}$
W_3
$\sigma\left(z_2^{(p)}\right)$
$z_2^{(p)}$
W_2
$\sigma\left(v^{(p,\text{black})}\right)$
$\sigma\left(v^{(p,\text{white})}\right)$
$v^{(p,\text{black})}$
$v^{(p,\text{white})}$
Position p (black's turn)
Incremental Computation with W_1
$y^{(q)}$
W_4
$\sigma\left(z_3^{(q)}\right)$
$z_3^{(q)}$
W_3
$\sigma\left(z_2^{(q)}\right)$
$z_2^{(q)}$
W_2
$\sigma\left(v^{(q,\text{white})}\right)$
$\sigma\left(v^{(q,\text{black})}\right)$
$v^{(q,\text{black})}$
$v^{(q,\text{white})}$
Position q (white's turn)

Figure 1: Evaluation function of "the end of genesis T.N.K.evolution turbo type D". The evaluation value for game position q is calculated, where q differs from p by one move. For a part of the calculation a difference calculation is used. The move number is considered during evaluation.

2.2 Activation Function

For the activation function let's denote the number of nodes in layer l by N_l , and we use the clipped ReLU function, which is expressed as

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

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

The advantage here is that the range of values is finite, and such can be realized by integer arithmetic and can be efficiently accelarated by the SIMD operations described below.

2.3 Feature-Vector (Input Vector)

For the feature vector we use vector of binaries where each entry is either 0 or 1. This restriction also supports the simplification of the difference calculation. In order to execute the difference calculation with high speed, it is desirable to make sure that the amount of entries that which change bevor and after a move is rather small.

One example of a feature vector that has these desired properties is the KP (King-Piece) encoding, which uses the relations of pieces between the own king or the enemy king and other pieces different from the king. The precise encoding used in "the end of genesisT.N.K. evolution turbo type D" is based on a modification of KP. Details are described later.

2.4 Affine Transformations and Memory Layout

In order to execute an affine transformation like the on in equation (2), it is usually efficient to calculate the inner product of one matrix row and the vector; the i-th row of $\boldsymbol{W}_l$ is denoted here by $\boldsymbol{W}_l(i,:)$.

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

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

Since the memory access pattern is continually centered around elements of lines of the matrix, we assume a row-major memory layout.

If the vector a_{l-1} however is parse (i.e. most entries are zero), then it's faster to calculate by just using some columns of the matrix like this: Calculate $\boldsymbol{W}_l(:,j)$ for the j-th column of $\boldsymbol{W}_l$.

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

In this case the memory access pattern relates to continually accessing elements of the matrix rows, and it is more efficient to use a column-major memory layout.

For the NNUE-evaluation function we use the affine transform of the feature vector **x** according to the method of equation (8).

$$\boldsymbol{z}_1 = \boldsymbol{v} \qquad (9)$$

$$\boldsymbol{v} = \boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{x} \qquad (10)$$

As mentioned, **x** is a binary vector, and if **x** is sparse, the affine transformation can be efficiently calculated by equation (8)[2]. By using the fact that **x** is a binary vector, we can also simplify equation (8) by $a_{l-1}, j = x_j \in \{0,1\}$ to:

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

2.5 Difference Calculation

In the NNUE evaluation function the vector, which is the result of the affine transformation of the feature vector $\boldsymbol{x}^{(q)}$ in position q, i.e.

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

calculated by using $\boldsymbol{v}^{(p)}$, which has been calculated already for the previous position p. Specifically, $\boldsymbol{v}^{(q)}$ is calculated by the following equation:

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

This can be interpreted as an expression, which extends the difference calculation of the linear model to a vector. The difference calculation is only used for the affine transformation of feature vectors (i.e. input vectors of the network), not in other parts.

2.6 Enemy/Pair-based Calculation

[2] It is important that changes of the feature vector (hamming distance) bevor and after a move is small, since the difference calculation has to be executed and we cannot assume that these vectors are sparse in that case.

In Shogi the position of the pieces depends on which player's turn it is, even though the arrangement of the pieces is symmetric. The Sankoma-Kankei evaluation function the evaluation values is calculated by using this linearity and split into two terms, and expressed as the sum of one part which is independent on whose turn it is, and one part which depends on whose turn it is [4].

For the NNUE evaluation function the evaluation is realized by always evaluating the position from the perspective of the player whose turn it is. First we illustrate a simple method. Suppose active(p) is the player whose turn it is in position p and $\boldsymbol{x}^{(p,c)}$ is the vector in position p taking the perspective oft the player c. Then the evaluation of the position

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

can be realized from the perspective of the player whose turn it is.

Since the feature vector changes depending on whose turn it is, the difference calculation has to be done both from the perspective of the first (black) and the perspective of the second (white) player.

For each move and c ∈ {Black, White } the difference for each is calculated as

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

For the calculation of the evaluation value (in a position) we use the value of the player whose turn it is:

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

Next we explain how this method is extended to use the vector of the player who doesn't have the current turn, for the computation of the evaluation value. Assume opponent(p) is the player who does not have the current turn, then for the non-playing side of position p we can use the following equation instead of equation (16):

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

This means that the feature vectors of the player who has the current turn and the one who does not are both transformed by affine transformations, and the combined vector is $\boldsymbol{z}_1^p$. The parameters $\boldsymbol{b}_1$ and $\boldsymbol{W}_1$ of the affine transformation are the same, so that this calculation can be interpreted as a folding in the direction of the player.

The evaluation function "the end of genesis T.N.K. evolution turbo type D" that is shown in Figure 1 is based on equation (17).

2.7 Half-KP Encoding

As mentioned, "the end of genesis T.N.K.evolution turbo type D" uses a modified version of KP encoding, a kind of two-piece relationship. Here KP is an encoding which uses the "relationship of the position of the own king or the enemy king and another piece (which is not the king). Here we don't use the enemy king to express positional relationships. In the following we use this approach, and dub it "HalfKP".

HalfKP is an encoding which can be combined with the method described in the last section that uses feature-vectors w.r.t. the player whose turn it is, and who does not have the current turn.

If HalfKP is used, v(p,active(p)) of equation (17) contains the information which corresponds to “KP of the king whose turn it is from the perspective of the player whose turn it is” and v(p,opponent(p)) corresponds to “KP of the king who does not have the current turn from the perspective of the player who does not have the turn”.

Even though the features itself do not contain information of the position of the enemy pieces, we can express the information of the whole position by using both the features for the side whose current turn it is and the features for the side who does not have the current turn.

The advantage of this method is that the amount of calculation to express the same information is smaller than with the common KP encoding.

2.8 Integer SIMD Instructions

SIMD instructions are operations which can apply the same instruction on multiple data at once. The NNUE evaluation function is designed for fast calculations by using integer SIMD instructions. The affine transformation of feature vectors is done by addition and subtraction of vectors with and without the difference calculation. The weight matrix $\boldsymbol{W}_1$ is converted to 16-bit-integer values, and then instructions for addition/subtraction of signed16 bit integers are used (VPADDW, VPSUBW in AVX2).

For affine transformations which are not feature vectors, the activations a_{l-1} of the previous layer and the weight matrix $\boldsymbol{W}_l$ are expressed as 8-bit values, and the inner product is calculated with VPMADDUBSW. This instruction calculates the product of two 8-Bit integer values.

The activation function is calculated by instructions which transform the bit-width of a vector of integers (VPACKSSDW, VPACKSSWB), as well as instructions which take the maximal value of each element of two 8-bit integer vectors (VPMAXSB).

	# columns	# rows
W_1	125388	256
W_2	512	32
W_3	32	32
W_4	32	1

Table 1: Size of the weight matrix, which is used in the evaluation function of "the end of genesis T.N.K.evolution turbo type D".

2.9 Network Size

Table 1 shows the sizes of the weight matrices, which are used in the „end of Genesis T.N.K.evolution turbo Typ D“. The number of columns and rows corresponds to the dimensions of the input resp. output of the affine transformation. The size of the network was adjusted such that the amount of positions which can be evaluated per time unit is equal to the case where Sankoma-Kankei is used as the evaluation function.

The vast amount of parameters are used for the weight matrix $\boldsymbol{W}_1$, which is used for the affine transformation of the feature vectors. Even though the transformations for W_2, W_3, W_4 can not be calculated by difference calculations, they can be calculated quite fast, since the amount of parameters is chosen to be small and we use 8-Bit integer SIMD instructions.

3. Conclusion

In this paper we have introduced an evaluation function for Shogi based on neural networks, and described the design and accelaration techniques for high-speed execution on CPUs. „the end of genesis T.N.K.evolution turbo type D", is the first Shogi software which is based on this technology and will take part in the 28. Computer Shogi world championship [1].

We plan to publish the source code of this program as open source later. It is based on the open source software program "Yaneura-Oh" [8] and is implemented in C++.

On top of the features described in this document we mention:

- Adaptability of functions and network structure by using class templates
- Fast implementation due to compilation and compiler optimizations
- Learning combined with stationary earch, similar to the method for learning used in Sankoma-Kankei
- Parameter-Sharing during learning by feature decomposition

The parameters and network structure used in "the end of genesis T.N.K.evolution turbo type D" are not necessarily the best. We hope that other developers will extend the source code and use it. We hope that this technology will contribute to further advancements in computer Shogi.

References

[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 mini-max 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 et al.: Mastering the game of Go with deep neural networks and tree search. Nature, Vol. 529, No. 7587, pp. 484-489, 2016.
[7] David Silver et al.: 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/