Comparison Training of N-Tuple Networks for Chess

Wen-Jie Tseng[†1] Jr-Chang Chen[†2] I-Chen Wu[†1]

Abstract: This paper applies a modified comparison training to a chess program. We implement the training method in the open-source UCI chess engine Stockfish. Feature weights are tuned based on the best moves searched by Stockfish from game records of grandmasters. The update formulas of feature weights are changed suitably for the evaluation functions which use tapered eval. With the method, the evaluation functions using tapered eval can be trained with the same training set without partitioning. The experiments include 1- to 4-ply training with a quiescence extension. The experimental results show that the trained version with 4-ply training outperforms the original version with only linear evaluation functions by a win rate of 65.25%.

Keywords: computer chess, comparison training, machine learning

1. Introduction

Evaluation functions are used for game-tree search in many computer games. It is critical to evaluate the winning chance of game positions for the player to move accurately. Positions are usually evaluated from the weights of designated features, such as pieces, locations, mobility and king safety for chess-like games. Many researchers have shown the difficulty of constructing an effective evaluation function, which includes (1) choosing appropriate features and (2) manually tuning their weights together with experts. In addition, the more features there are, the more difficult and time-consuming this work becomes.

In the past, feature selection required both domain knowledge and programming skills. A structured evaluation function representation [2] was proposed for exploring the feature space to discover new features. A technique called n-tuple networks provided a knowledge-free way of extracting game-playing strategies. It was applied to Othello [2][8], shogi [6], Connect4 [14], Chinese chess [15] and 2048 [10][17] successfully.

To improve the playing strength of game-playing programs, machine learning methods were used to tune the evaluation functions automatically. Comparison training, one of the successful methods, was employed in backgammon [11][12], shogi [5][6], chess [13][16], and Chinese chess [15] programs.

The stage-dependent features [2] is one of the issues of tuning feature weights. The features chosen for evaluating positions may depend on game stages. It is crucial to evaluate positions smoothly while crossing stages and to avoid big evaluation jumps in game-tree search. However, it may be necessary to partition a training set according to game stages and then tune the weights for each game stage separately. This process requires a large number of training positions, especially when the number of stages is large. Tapered eval [3] is a common technique used to make the evaluations become smooth across adjacent stages. Most chess programs use this technique, such as Fruit, Crafty and Stockfish. For computer chess, it is worth investigating whether comparison training can be used to tune the weights of evaluation functions which use tapered eval.

Previously, Tseng, et al. [15] obtained significant improvement on their Chinese chess program by using comparison training with tapered eval and an n-tuple network that take into consideration the relationship of positional information from individual features. In this paper, we apply their method to a chess program and also obtain improvement.

2. Related Work

This section first briefly reviews related research on stage-dependent features, comparison training and n-tuple networks. Then, we describe the chess program, Stockfish, where we implemented the training method.

2.1 Stage-dependent Features

Selecting the features and tuning their weights in evaluation functions depend on the game stages in many games. For example, in the endgame stage of Chinese chess, the material combination of a knight and a cannon is better than that of two knights or two cannons. In Chinese dark chess, the weights of the king depend on the number of pawns and hence need to be tuned as the game stages progress. In chess, tapered eval was widely used to measure a stage based on the pieces on the board and then to obtain the weight of the stage by interpolation. The technique was implemented by most chess programs.

It is common to formulate a linear evaluation function as follows.

$$\text{evaluation}(w, s) = w^{\mathrm{T}}\varphi(s), \qquad (1)$$

where w is a weight vector corresponding to a feature vector $\varphi(s)$ which indicates the features in a position s. In the implementation of tapered eval, two weights, w_o and w_e, are used for each feature to represent the weights at the opening and the endgame respectively. Then, the feature weights are calculated by a linear interpolation of the two weights according to many sub-stages divided by tapered eval. The following linear interpolation replaces the weight vector w in Formula (1) for an evaluation function using tapered eval,.

$$w = \alpha(s)w_o + \big(1 - \alpha(s)\big)w_e, \qquad (2)$$

where the game stage index $\alpha(s)$ denotes how close to the opening a position s is and $0 \le \alpha(s) \le 1$. Hence, it is

†1 Dept. of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. E-mail: wenjie0723@gmail.com and icwu@csie.nctu.edu.tw

†2 Dept. of Computer Science and Information Engineering, National Taipei University, New Taipei City, Taiwan. Email: jcchen@mail.ntpu.edu.tw

necessary to use different weights for the same feature in each stage.

2.2 Comparison Training

Comparison training belongs to a kind of supervised learning method and is used to train evaluation functions [11]. The objective is to tune the weights in an evaluation function so that the results after searching positions match the desired moves of the positions. For example, grandmaster game records are commonly used as training data for learning chess.

Let s denote a training position and let s_{best} denote the best child position of s. For all child positions of s, they are compared with s_{best}. In [15], s_{best} is assumed to be the child position reached by an expert's move. The information involved in the comparison results is extracted to tune feature weights of an evaluation function. The aim is to make the evaluation value calculated by the tuned evaluation function be as close to that by making the move to s_{best} as possible. Averaged perceptron [4] is an online training method described below. An update for feature weights is made for each training position. Let $w^{(t)}$ denote the weight vector in the t-th update, and assume that $w^{(t-1)}$ is used to evaluate all of the child positions of s during the t-th update. For a position s, the update formula is as follows.

$$w^{(t)} = w^{(t-1)} + \frac{1}{\left|S^{(t)}\right|} \sum_{s_i \in S^{(t)}} \left(\varphi(s_{best}) - \varphi(s_i)\right), \quad (3)$$

where $S^{(t)}$ is the set of child positions of s which has a better evaluation value than that of s_{best}, $\left|S^{(t)}\right|$ is the number of these better child positions, and $\varphi(s_i)$ is the feature vector of s_i. Assuming that there are T training positions, there are T updates in each iteration. After N iteration, the final feature weight is the average weights of $w^{(0)}$ to $w^{(N \cdot T)}$. When the evaluation function using the final feature weight does not improve on choosing the desired move, the training process stops.

A correct evaluation value improves the quality of tuning in comparison training. One method is to replace the evaluation values of child positions with those after making a shallow tree search. In [13], Tesauro proposed d-ply comparison training by replacing the evaluation for s_i with the leaf on the principal variation (PV), denoted by l_i, in the minimax search with depth d, as shown in Fig. 1.

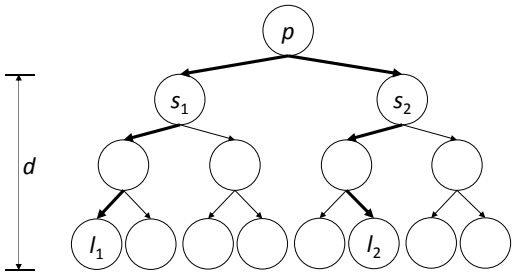

Fig. 1 An extension of comparison training ($d = 3$).

As described in Subsection 2.1, the feature weight w is obtained from w_o and w_e. In [15], a modified comparison training that incorporates tapered eval was proposed. To update w_o and w_e, the second term in Formula (3) is multiplied by the two factors, $\frac{\alpha(s)}{\alpha(s)^2+(1-\alpha(s))^2}$ and $\frac{1-\alpha(s)}{\alpha(s)^2+(1-\alpha(s))^2}$, respectively.

Comparison training was successful applied in backgammon, shogi, chess and Chinese chess. The backgammon program, called Neurogammon, was trained with comparison training from about 8000 positions chosen from 400 games [12]. Comparison training was also applied to tuning the evaluation function in the chess program, called SCP, and some weights in Deep Blue's evaluation function with a slight modification [13]. The tuning of the king safety weights had made a difference in Deep Blue's choice on the Kasparov-Deep Blue rematch. For the shogi program, called Bonanza, a similar machine learning method was used to tune the evaluation function [7]. These programs reached great achievements in international competitions including the Computer Olympiad and World Computer Shogi Championship.

2.3 N-tuple Networks

In chess-like games, a large number of features are designed in evaluation functions. Some of them are highly related and the relations are hard to be extracted. N-tuple networks are a good implementation for this issue with less requirement of domain-knowledge. For example, one 6-tuple covers six designated squares on the Othello board [8] and includes 3^6 features, where each square is empty or occupied by a black or white piece.

Assume there are m n_i-tuples in an n-tuple network, where $i = 1, 2, \ldots, m$. Each n_i-tuple consists of n_i features, denoted by f_{ij}, where $j = 1, 2, \ldots, n_i$. The feature weight w_i of an n_i-tuple indicates the importance of these features f_{ij} as a whole.

Each n_i-tuple is implemented by a look-up table LUT_i where the weights are stored. Let $\mathrm{idx}(v_{i1}, v_{i2}, \ldots, v_{in_i})$ be a function that calculates an index of the feature value v_{ij} of each feature f_{ij} in n_i-tuple. First, v_{ij} is extracted from a given position s. Then, each weight w_i is queried from LUT_i using idx. Finally, the n-tuple network sums up all of the weights and obtains the evaluation value of s.

2.4 Stockfish

We use Stockfish for analyzing the training method since it is one of the strongest open-source chess engines in the world. Moreover, Stockfish uses the UCI protocol [9] to communicate with a GUI, which benefits testing and analyzing training results. Stockfish uses techniques including principal variation search, quiescence search, transposition tables, static exchange evaluation, killer and history heuristics, null move pruning, futility pruning, and late move reductions. Stockfish also uses tapered eval in the evaluation functions. The game stage for a position s is calculated as follows.

$$\alpha(s) = \frac{max\{3915, min\{15258, NPM(s)\}\} - 3915}{15258 - 3915},$$

where $NPM(s)$ is summed opening weights of the non-pawn material in s for both white and black sides, and 15258 and 3915 are the boundaries of opening and endgame.

Some of the features in the evaluation function of Stockfish are linear and the others are non-linear. The evaluation function includes the following sub-functions.

- psq_score() evaluates material and piece squares.
- imbalance() evaluates the material imbalance.
- pieces() evaluates pieces of a given color and type according to their mobility or specific patterns.
- king() evaluates bonuses and penalties to a king of a given color.
- threats() evaluates bonuses according to the types of the attacking and the attacked pieces.
- passed() evaluates the passed pawns and candidate passed pawns of a given color.
- space() evaluates a bonus based on the number of safe squares available for minor pieces on the central four files on ranks two to four.
- initiative() evaluates the initiative correction value based on the known attacking/defending status of the players.
- pawn_score() evaluates the pawns of a given color.
- Specialized sub-functions evaluate positions with particular material configurations.

Among the above sub-functions, psq_score(), pieces(), threats(), and pawn_score() include only linear features, whereas imbalance(), king(), passed(), space(), initiative(), and specialized sub-functions include linear and non-linear features. Note that when a position is a type of specialized endgames, the evaluated value of the position is exactly the value evaluated by the specialized sub-functions since the specialized sub-functions are absolutely correct.

3. Method

We use comparison training, incorporating tapered eval, to tune the weights of linear features and use n-tuple networks to implement new features that are not in Stockfish.

3.1 Training and Tapered Eval

In the second term of formula (3), the difference of two feature vectors from two positions is computed. More specifically, for a training position s, s_{best} is from the desired move and s_i is from other moves. Since s, s_{best} and s_i may be in different game stages, it is reasonable to further change formulas (3) for updating w_o and w_e, as follows.

$$w_o{}^{(t)} = w_o{}^{(t-1)} + \frac{1}{|S^{(t)}|}\sum_{s_i \in S^{(t)}} \left(\varphi_o(s_{best}) - \varphi_o(s_i)\right)$$

$$w_e{}^{(t)} = w_e{}^{(t-1)} + \frac{1}{|S^{(t)}|}\sum_{s_i \in S^{(t)}} \left(\varphi_e(s_{best}) - \varphi_e(s_i)\right)$$

, where

$$\varphi_o(s) = \frac{\alpha(s)}{\alpha(s)^2 + \left(1-\alpha(s)\right)^2}\varphi(s)$$

$$\varphi_e(s) = \frac{\left(1-\alpha(s)\right)}{\alpha(s)^2 + \left(1-\alpha(s)\right)^2}\varphi(s)$$

The main idea is similar to [15]. The intention is to make the update formulas more suitable for the opening and endgame weights since $\alpha(s)$, $\alpha(s_{best})$ and $\alpha(s_i)$ may be different.

3.2 Weight Initialization

At the beginning of training, the opening and endgame weights of material are both initialized as in Table I, and the other weights are initialized to zero. The weights are chosen based on the values in the source code of Stockfish. We chose the values close to the average values of the opening and endgame weights for each piece. The concept is like the shogi program Gekisashi [16] which initialized the weights for pieces to some heuristic values and the others to zero.

Table I. Initial weights of material for training.

Queen	Rook	Bishop	Knight	Pawn
2600	1300	850	800	200

3.3 Material Constraint

To prevent weights from diverging during training, we add a constraint when using evaluation functions in tree search. We use floating-point weights in the training and the rounded integer weights in the game-tree search. A transition from floating-point weights to integer weights is computed by $w_{integer} = round(\gamma w_{floating-point})$, where $\gamma = 16488/(2w_o[queen] + 4w_o[rook] + 4w_o[bishop] + 4w_o[knight])$ is a scaling factor and 16488 is obtained from the source code of Stockfish by the summation of total non-pawn material for both sides. The intention is to make the summed opening weights of two queen, four rook, four bishop and four knight be a constant. Note that we do not have to change the boundaries of opening and endgame (15258 and 3915) that are already used in Stockfish since the summation of non-pawn material are fixed.

3.4 N-tuple Features

We design 2-tuple networks for chess like in [15]. The feature set extracts the locational relation of two pieces, which may include attack or defense strategies. One 2-tuple of size 32 × 63 can be used to represent the location relation for two specific pieces when the left-right symmetry of piece locations is considered. Thus, one 2-tuple is used for each combination of two pieces and there are in total 72 2-tuples. The total number of features for 2-tuples is 145,152. Then we follow the formulas proposed in [15] to perform the comparison training.

4. Experimental Result

The experiments described in this section used Stockfish (version date 2018/7/30), whose source code was available on GitHub. The training data included 94,167 game records of expert players whose Elo ratings exceeded 2500 (Grandmaster level). Among these game records, five million positions were selected for training and five hundred thousand were for testing. For benchmarking, one thousand positions were selected for self-play based on the frequencies played by experts. A total of two thousand games, from the perspective of both players, were played in each experiment. Each move took one hundred thousand nodes.

4.1 Evaluation Functions for Training

We used ten versions of Stockfish to analyze the training method described in Subsection 3.1. All versions are listed in Table II. For the original versions ORI1 and ORI2 from GitHub, the former is the version without any changes and the latter is the version that removes non-linear sub-functions from ORI1. For analysis, we removed the non-linear sub-functions (except the specialized sub-functions) of the evaluation function and reserved the linear sub-functions. We reserved the specialized sub-functions in ORI2 since their evaluations are absolutely correct.

CT1-CT4 and CT5-CT8 are the trained versions based on ORI1 and ORI2 respectively. Additionally, the trained versions incorporated the 2-tuple networks as described in Subsection 3.4. The experiments included 1- to 4-ply training with a quiescence extension. Note that the weights in non-linear sub-functions were not trained in our experiments.

In order to improve the quality of training data, we used the original version ORI1 to search the best moves of the training positions and used the searched moves as the desired moves in the training. Each move took two hundred thousand nodes.

Table II. The versions of Stockfish for experiments.

Versions	Explanation
ORI1	Original version without any changes
ORI2	Remove non-linear sub-functions from ORI1
CT1	ORI1+2-tuple trained with 1-ply training
CT2	ORI1+2-tuple trained with 2-ply training
CT3	ORI1+2-tuple trained with 3-ply training
CT4	ORI1+2-tuple trained with 4-ply training
CT5	ORI2+2-tuple trained with 1-ply training
CT6	ORI2+2-tuple trained with 2-ply training
CT7	ORI2+2-tuple trained with 3-ply training
CT8	ORI2+2-tuple trained with 4-ply training

4.2 ORI vs. ORI

First of all, for establishing a baseline for experiments, we compared the two original versions to see how different between the versions with and without the non-linear sub-functions. The result is shown in Table III. ORI1 outperforms ORI2 by a win rate of 82%. That means the non-linear sub-functions play an important role in the evaluation function. The most playing strength are contributed by the non-linear sub-functions in Stockfish.

Table III. Experiment results of ORI1 vs. ORI2.

Versions	Win rate
ORI1 vs. ORI2	82.00%

4.3 CT vs. ORI

For testing the power of each trained version, we let each CT1-CT8 compete against the baselines ORI1 and ORI2. Table IV shows the match results of CT1-CT4 against ORI1. The win rate is about 40% for CT2-CT4, where CT1 is the weakest one. From the results, the linear sub-functions were not suitably trained to cooperate with the non-linear sub-functions.

Table IV. Experiment results of CT1-4 vs. ORI1.

Versions	Win rate
CT1 vs. ORI1	33.83%
CT2 vs. ORI1	40.10%
CT3 vs. ORI1	38.77%
CT4 vs. ORI1	41.30%

Table V shows the match results of CT5-CT8 against ORI1. As mentioned above, without the non-linear sub-functions, the playing strength of the versions becomes very weak. However, CT5-CT8 played better than ORI1. It seems that the versions with only linear evaluation functions were trained well.

Table V. Experiment results of CT5- CT8 vs. ORI1.

Versions	Win rate
CT5 vs. ORI1	21.70%
CT6 vs. ORI1	24.30%
CT7 vs. ORI1	26.13%
CT8 vs. ORI1	28.15%

Table VI shows the match results of CT1-CT4 against ORI2. The trained version clearly outperforms ORI2 by win rates over than 72%. We think that is because the non-linear sub-functions still work with the linear sub-functions but not as good as the original version ORI1.

Table VI. Experiment results of CT1-CT4 vs. ORI2.

Versions	Win rate
CT1 vs. ORI2	72.33%
CT2 vs. ORI2	77.05%
CT3 vs. ORI2	75.40%
CT4 vs. ORI2	78.28%

Table VII shows the match results of CT5-CT8 against ORI2. From the results, it shows that the 2-tuple networks clearly contributed to improving the playing strength of computer chess. Especially for CT7 and CT8, they outperforms ORI2 by win rates over than 65%.

Table VII. Experiment results of CT5-8 vs. ORI2.

Versions	Win rate
CT5 vs. ORI2	58.92%
CT6 vs. ORI2	62.25%
CT7 vs. ORI2	65.03%
CT8 vs. ORI2	65.25%

4.4 TRAINED vs. TRAINED

For testing the power of each version trained with different search depth, we let each CT2-CT4 and CT6-CT8 compete against the versions trained with lower search depth. The results are listed in Table VIII and Table IX. It is clear that 2-ply training outperforms 1-ply training. Nevertheless, 3-ply training is slightly better than 2-ply training and 4-ply training performs nearly as 3-ply training. The results is quite different from the experimental results in [13]. From the results, it is better that the search depth of training has to be at least 2-ply.

Table VIII. Experiment results of CT2-CT4 vs. CT1-CT3.

Versions	Win rate
CT2 vs. CT1	57.70%
CT3 vs. CT2	50.95%
CT4 vs. CT3	51.65%

Table IX. Experiment results of CT6-CT8 vs. CT5-CT7.

Versions	Win rate
CT6 vs. CT5	57.65%
CT7 vs. CT6	52.90%
CT8 vs. CT7	50.90%

4.5 Analysis of Material Weights

We compared the material weights of ORI1 and CT4 as shown in Table X. After training, the opening weights changed slightly; however, the endgame weights changed a lot. We think the reason is that Stockfish uses the opening weights of non-pawn material to compute the game stage and we used the material constraint described in Subsection 3.3. Another reason may be that Stockfish uses some specialized sub-functions to evaluate endgame positions. The sub-functions make the material features in opening become less important than in endgame.

Table X. The trained material weights of ORI1 and CT4.

Features	ORI1		CT4	
	w_o	w_e	w_o	w_e
Queen	2500	2670	2432	1671
Rook	1282	1373	1189	812
Bishop	826	891	898	499
Knight	764	848	819	452
Pawn	171	240	107	202

4.6 Discussion

For chess, many significant features, including king safety, were designed in non-linear evaluation functions. It is crucial to tune the weights of such features. Minimax Tree Optimization [6] is a type of comparison training which is suitable for tuning non-linear evaluation functions. It is worth investigating how the method can be modified to tune the evaluation functions which use tapered eval.

5. Conclusion

This paper applies a modified comparison training, incorporating tapered eval, for computer chess and additionally designs new features with n-tuple networks for chess. We change the update formulas for the opening and endgame weights in the evaluation functions which use tapered eval. The new formulas are also useful for tuning feature weights. The experiments show that the training is efficient and effective to tune the linear evaluation functions and 2-tuple networks are also helpful for computer chess. With the method, the evaluation functions using tapered eval can be trained with the same training set without partitioning. Possible future work would be the training of non-linear sub-functions in Stockfish.

Acknowledgement

The authors would like to thank Ministry of Science and Technology of Taiwan for financial support of this research under the contract numbers MOST 107-2634-F-009-011, 107-2634-F-259-001, 106-2221-E-305-016-MY2, 106-2221-E-305-017-MY2 and 106-2221-E-009-139-MY2.

Reference

[1] Buro, M., Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello, *Technical Report* 96, NEC Research Institute, 1997.

[2] Buro, M., From Simple Features to Sophisticated Evaluation Functions, In *Proceedings of the First International Conference on Computers and Games (CG'98)*, pp. 126–145, 1998.

[3] Chess Programming Wiki, Taper Eval, [Online], Available: https://www.chessprogramming.org/Tapered_Eval

[4] Collins, M., Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms, In *EMNLP'02*, pp. 1-8, 2002.

[5] Hoki, K. and Kaneko, T., The Global Landscape of Objective Functions for the Optimization of Shogi Piece Values with Game-Tree Search. *Advances in Computer Games 13*, LNCS 7168, pp. 184-195, 2012.

[6] Hoki, K. and Kaneko, T., Large-Scale Optimization for Evaluation Functions with Minimax Search, J. Artif. Intell. Res. (JAIR) 49, 527-568, 2014.

[7] Kaneko, T. and Hoki, K., Analysis of Evaluation-Function Learning by Comparison of Sibling Nodes, In *Advances in Computer Games 13*, *LNCS* 7168, 158-169, 2012.

[8] Lucas, S. M., Learning to Play Othello with N-tuple Systems, *Australian Journal of Intelligent Information Processing* 4, 1-20, 2007.

[9] Shredder Computer Chess , UCI protocol [Online]. Available: https://www.shredderchess.com/chess-info/features/uci-universal-chess-interface.html

[10] Szubert, M. and Jaśkowski, W., Temporal Difference Learning of N-tuple Networks for the Game 2048, In *2014 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1-8, 2014.

[11] Tesauro, G., Connectionist Learning of Expert Preferences by Comparison Training. *Advances in Neural Information Processing Systems* 1, 99-106, Morgan Kaufmann, 1989.

[12] Tesauro, G., Neurogammon: a Neural Network Backgammon Program, *IJCNN Proceedings* III, 33-39, 1990.

[13] Tesauro, G., Comparison Training of Chess Evaluation Functions. In: Machines that learn to play games, pp. 117-130, Nova Science Publishers, Inc., 2001.

[14] Thill , M., Koch, P. and Konen, W., Reinforcement Learning with N-tuples on the Game Connect-4, In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature - Volume Part I (PPSN'12)*, pp. 184-194, 2012.

[15] Tseng, W.-J., Chen, J.-C., Wu, I-C., Wei, T., Comparison Training for Computer Chinese Chess, arXiv:1801.07411, 2018.

[16] Ura, A., Miwa, M., Tsuruoka, Y., and Chikayama, T., Comparison Training of Shogi Evaluation Functions with Self-Generated Training Positions and Moves, *CG 2013*, 2013.

[17] Yeh, K.-H., Wu, I-C., Hsueh, C.-H., Chang, C.-C., Liang, C.-C. and Chiang, H., Multi-Stage Temporal Difference Learning for 2048-like Games, *IEEE Transactions on Computational Intelligence and AI in Games,* 9(4), 369-380, 2017.