Complex Decision Making With Neural Networks: Learning Chess

Jack Sigan

Dr. Aleksander Malinowski, Advisor

Bradley University Department of Electrical and Computer Engineering

May 9, 2005

Abstract

The advancement of computing technology has allowed machines to defeat even the best human practitioners of chess. Still, typical computer algorithms differ enormously (in function) from human playing techniques. This research project examines the use of artificial neural networks (ANNs) as an alternative mechanism for playing the game.

Following an introduction to this topic, the discussion covers the ANN system design, and the procedures and tools needed for its development. Preprocessing concerns, training procedures, and considerations for the system evaluation are detailed. An analysis of initial results and a feasibility statement of using ANNs in this application are also provided.

Table of Contents

Introduction:

The real question to be asked; can chess really be implemented with artificial neural networks (ANNs) as opposed to the traditional approach of exhaustively searching game trees?

Applied correctly and appropriately designed, artificial neural networks have extraordinary potential for solving problems involving generalization, trend recognition, and pattern matching. Game play, which often involves non-linear strategies or decision making, is a particularly good area to demonstrate the ANN as a way of approximating otherwise inexpressible functions [1]. To date, the promise and lofty expectations of this "artificial intelligence" approach have yet to be fully realized, which creates demand for further research. Work on complex problem solving, such as that required in classical board games such as chess, has been limited, although many of the available research results [1-4] are tantalizing.

Numerous published studies serve as motivation and a starting point for this research. Chekkapilla and Fogel, in developing an ANN to play checkers, indicated that there is feasibility in teaching ANNs games of some complexity [1, 2]. Although chess is clearly a leap forward from checkers, it seems a logical next step in the evolution and development of the ANN and its applications, as chess is one of the most widely studied and researched games. A demonstration of strategy in such a game is also recognized as a direct measure of logical decision making ability [1]. Chess as an ANN problem is not a new idea; in fact, ANNs have already been proven to be highly effective in playing chess endgames, as seen in "Neural Network Learning In a Chess Endgame," although it is ironic that the author of this specific study also states that chess is too difficult for ANNs to learn in its entirety [3]. The "Distributed Chess Project," when considering the full game of chess, reported approximately 75% accuracy in choosing the "proper" move when confronted with a chess problem external to the training domain. While not fully successful as implemented, the study does appear to indicate that chess schema may be learned by ANNs [4].

The published studies therefore seem to give a mixed opinion of whether a solution is possible in this problem. However, three points must be emphasized here. Technology, both in terms of hardware and software capability, has improved exponentially since [3] was published in 1994, which allows vastly more complex networks to be implemented. Also, breaking the game down as proposed in this project has not, in the author's knowledge or opinion, been considered or researched before. Finally, the success (and failures) of [4] can only be seen as relevant to the approach used, which is derivation of ANNs through genetic algorithms. A vast training set was apparently not utilized, and external rule control was not applied. Considering the above facts, proposed improvements, and possible implications of this kind of research to the field of artificial intelligence, the project was pursued.

Underlying Theory of Learning and Move Assumptions:

At this time, it is appropriate to describe the general goal of the project and state a few vital assumptions. In this project, the game of chess is to be broken down into simple units—specifically moves consisting of an initial position and a final position without regard to the piece to be moved. It is also possible to break the game down by piece as will be described shortly. A neural network is to be created for each "unit," and trained to either make or not make the move in question.

In order to train the networks, a massive quantity of data is required. The source of the data is ChessBase Version 9.0, which is simply a database of millions of saved chess games. Each game can be considered as a series of board positions. During training, random sets of

board positions will be chosen as the input vectors, and the desired output for ANN training will be the next move made. In other words, input vectors will not be held within the context of the game from which they originally came. In choosing the training vectors, *only games where black won the game are to be considered*. It is assumed that the next move (which was made by a human player) was the "proper" move to make in the given situation—otherwise the result would not be a "black win." All training data originates from games with an ELO rating of 1400+.

In an attempt to propose the actual mechanism of play the ANN system will develop, it is useful to look at the methods employed by human players—especially the top performers. Human chess players do not perform a numerical evaluation of the board from the current position to the n^{th} position into the future as most computer chess programs do. They do not possess the ability to solve a complex evaluation function with hundreds of position considerations. Instead, human players are believed to play using the concept of a "schema." This is merely a particular arrangement of pieces or a specific position which the player recognizes. In knowing a schema, a player should have a basis help choose their move.

This project is effectively proposing that schema knowledge may be learned by the ANN system. Through training with hundreds of thousands of board positions, the ANN should begin to assign the key components of input patterns additional weight—which is thought to be the same result as schema "generalization."

Standards and Patents:

Standards apply to this project insofar as the rules of chess and also the file formats which are to be implemented and manipulated in the data preprocessing stage. The rules of chess which are to be considered the "standard" applied in deciding move legality in the final system implementation will be the 2001 edition of the World Chess Federation's Laws of Chess [7]. This standard is freely available. There is some possibility than games in the database may contain moves which are illegal according to this standard. In such cases, the standard is to have priority, and any move which is "taught" to a neural network deemed illegal based on these laws will be disabled in the decision finalization block shown in Fig. 2 and Fig. 3.

The two file formats to be utilized in this project, PGN and EPD, are also recognized as "standards." The PGN file standard [8] which will be consulted in case of questions is the 1994 document entitled "Standard Portable Game Notation Specification and Implementation Guide," which was compiled by members of the newsgroup "rec.games.chess." EPD, although considered a "standard," does not have a single defining document. This is not of concern however, as only one source of EPD strings is to be used in the project to create temporary data. Any new software will be designed to work with the format provided by the existing PGN to EPD conversion application.

As far as patents are concerned, only one can be found which describes an idea with a fairly strong resemblance to the stated goals of this project. US patent 6,738,753 [9] describes a "modular, hierarchically organized artificial intelligence entity," which in many ways describes the concept of the "parallel" approach to the chess problem described in this paper. This patent refers to a system of modular components which are each specialized in a specific task, although the structure of them is reproduced. The task specialization is developed by individual "learning." These "golems" are controlled by another unit, which is somewhat along the lines of the "rule logic" to be implemented in this project. This is where the similarities end however, as the patent describes the learning process of one golem to be directed by another (superior) golem. In the end, the system resembles a military chain of command. It is doubtful if this patent

will have a profound impact on the development of this research, as it does not adequately describe the internal design of the golems to be of any practical usefulness. Given the fundamental difference in the described "chain of command" learning to the system described in this paper, it is also doubtful that any useful learning strategies can be adopted.

Objective and System Description:

Although mainly a research endeavor, the end goal of this project was to produce a system based on artificial neural networks (ANNs) to play chess (as the dark side only) effectively against a human opponent. The dark side was chosen to prevent the system from having to make the first move. To meet the objective, research centered on artificial neural network (ANN) topology, with the purpose of creating a topology appropriate for complex decision making in a massively dimensional problem. Functionality of the system may be broken into two parts: the learning mode is an automated process where the ANNs learn how to play from an extensive external database of games; while the playing/advisory mode of operation accepts user interaction, taking move inputs from the player and providing the move(s) chosen by the system. The playing mode's interface is also capable of playing a chess engine (through command line interaction). Fig. 1 illustrates the operating modes. The training mode must precede the playing mode. The performance of the playing and advisory modes is clearly a direct reflection of the learning mode, as well as the internal structure of the ANN module.

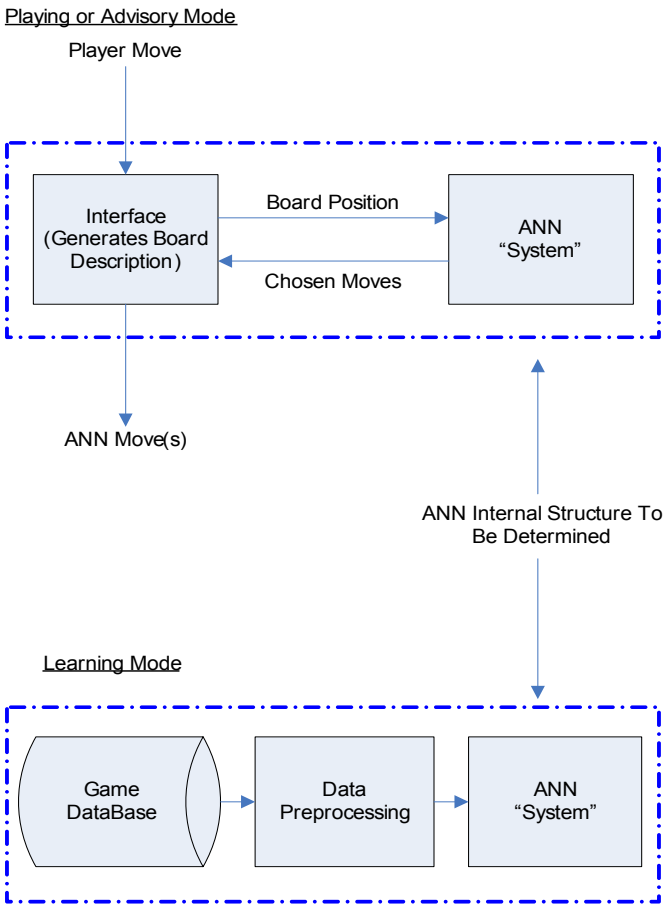

Figure 1.
System Block Diagrams for Operating Modes

Input and Output Descriptions:

As shown in Fig. 1, four main components will make up the system; the interface module, the ANN module, and the preprocessing and database systems. The ANN module is shared by both the learning and the playing/advisory modes. Table 1 displays the connections and IO paths which will comprise the modes of operation. This same information is shown graphically in Fig. 1. Note that this IO plan is very high level, and that far more complexity is found within the individual modules, which themselves will be broken down into components. The ANN module in particular will be discussed shortly in much more detail.

Learning mode:

Module	External Input	External Output	Inter-Modular Input	Inter-Modular Output
Game Database/ Preprocessing	NA	NA	NA	Game records to ANN module
ANN Module	NA	NA	Game records from database/preprocessing	NA

Playing/Advisory mode:

Module	External Input	External Output	Inter-Modular Input	Inter-Modular Output
Interface Module	Player's move	ANN's move(s)	ANN move(s) choice	Board description to ANN
ANN Module	NA	NA	Board description from interface	Move choices to interface

Table 1. Input/Output Descriptions by Module

Structure of the ANN Modules:

Fig. 1 places the ANN module in both the learning and playing/advisory mode, and it must be emphasized that the design of this component was the end goal of the project's research. In the beginning, two major design paradigms were considered. The first paradigm is depicted in Fig. 2.

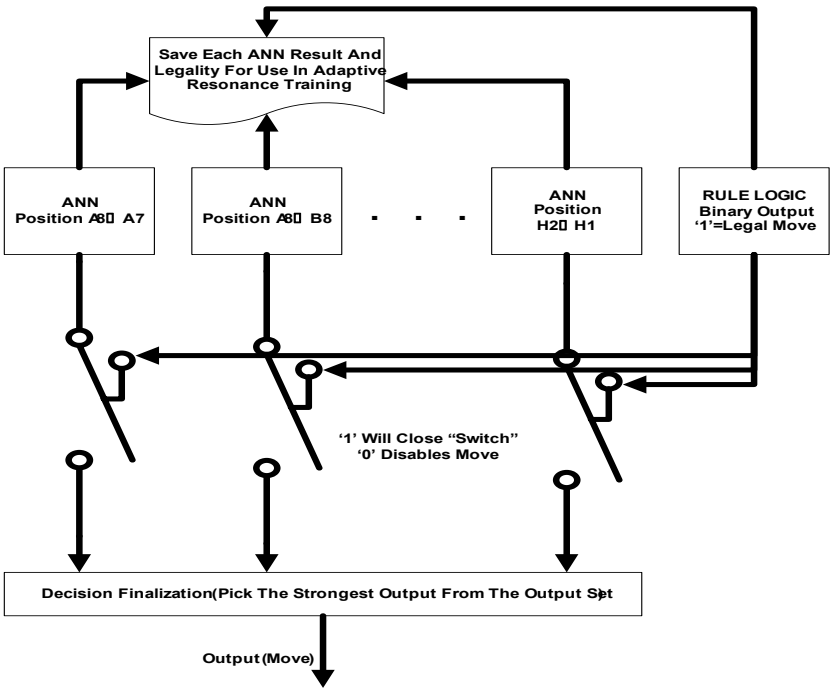

Figure 2. Geographically Derived ANN System Design

The first approach involves a geographical breakdown of all possible moves in the game based on the starting position *i* and the final position *f*. An entire game *g* of *n* moves may be expressed as a set of board positions b_i where *i* is the move number from 0 to *n*. Of course, each $b_i \in g$ has a set of possible legal moves based on the piece *p* located at all *i* positions held by the game participant's side. Therefore, it is possible to define the set of legal moves as $m_{if}=f(b_i)$ since complete knowledge of *p, i,* and *f* may be obtained from any b_i in addition to the rules of chess. *For now it is required to consider the castling moves to be special cases, as they involve more than 1 piece*. If considering a whole game of *n* moves, then all legal moves made throughout the game may be expressed as $L = \sum_{i=0}^{n} f(b_i)$ All legal moves in chess may therefore be found as $M = \sum_{i=0}^{\infty} (L_i)$. It is not difficult to determine m_{if} through simple logic for any $b_i \in g$. *M* is finite, obviously having a value less than 64x63 moves. This idea is fundamental to justifying the design in Fig. 2. In this design, it is required to create an individual ANN structure for all moves *t*, $t \in M$. Thus, each ANN will be trained to make a 'yes' or 'no' decision for its own move *t* based only on b_i. It is possible that some networks may say 'yes' for making a move, even if the move is illegal based on the rules of chess for the given b_i ,even though $m_{if} \in M$! For example, the move

for the left black rook may say 'yes' to move from square D4 to D6 even though square D5 is held by an enemy. D4 to D5 may also report a 'yes' decision, so how will an action be chosen? This is where knowledge of m_{if} as defined above for the specific board position is applied. It is possible to deactivate the ANNs which want to make illegal moves. Once this is done, the ANN with the highest output level (from a hyperbolic tangent function, range +/- 1) is chosen as the move to make. The total number of neural networks which exist in the geographical approach is 1792.

Originally, it was proposed that over time it should in theory be possible to modify the ANNs so they do not make illegal moves anymore by using an implementation of adaptive resonance. In short, the input b_i will be saved in a file along with a 'no' output for any illegal moves. A special training session will then be held to improve the network.

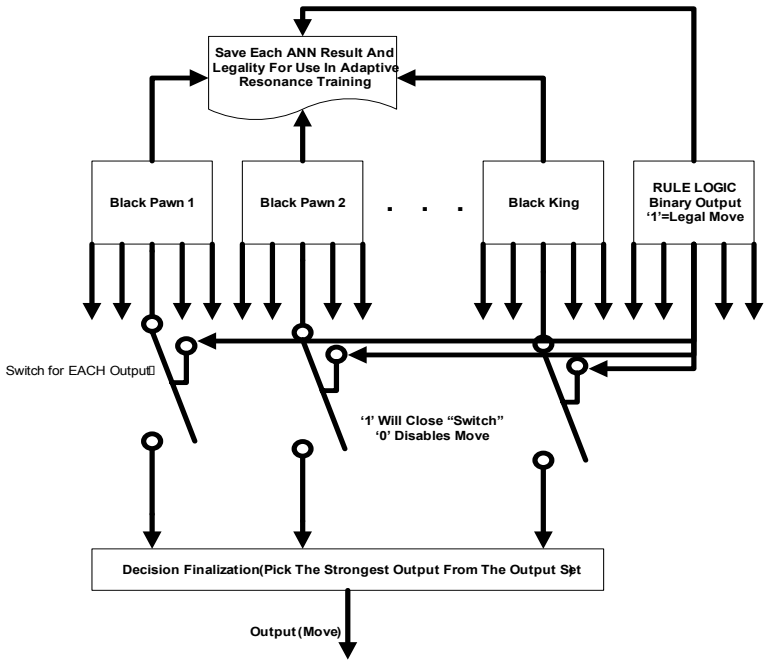

Figure 3. Functionally Derived ANN System Design

The second approach initially proposed was to break the game of chess apart by piece, thereby allowing a more "functional" approach to the same problem. 16 networks are defined which represent all pieces for the dark side. Each network will have *m* outputs, which are defined by $m = \sum_{i=0}^{\infty}(l_i)$ where $l = \sum_{i=0}^{n} f(p, b_i)$. Defining *l* is only slightly different from how *L* was defined above, as it must have emphasis placed on *p* (the piece) in addition to b_i. Once again *n* is the number of turns in a single game. The total number of outputs (*M*) remains the same. Outputs are still deactivated in the same fashion as described above. Adaptive resonance was still proposed to improve the network decisions over time. Fig. 3 shows a block diagram of the functionally derived approach.

In general, both design approaches may be viewed as parallel approaches to complex problem solving. Many benefits are achieved in a parallel approach, including shortened training

time and simplified internal network design. Although it was not known in advance how large the networks should be for either approach, it seemed logical that the networks for the geographical approach would be significantly smaller (in terms of node count) than those in the functional design, simply because they have only one output, thus the simulated function was assumed to be less complex, thereby requiring fewer nodes to achieve accurate approximation results. It was expected that all networks will be the same size for the geographic approach, although it may also seem valid that the moves involving edge positions as initial positions will not need as many nodes as they simply do not have as many legal moves to make, while moves in the center could require a greater number of nodes. Unfortunately, theory does not exist to accurately predict ANN size for problems such as this, so it was left largely to trial and error. More experimentation would be needed to optimize network size.

Another proposed benefit of the parallel approach is that learning was anticipated to be less destructive than it would be for a single network having to process all moves. In such a situation, involving a highly dimensional problem and dataset, adjusting the weights and thresholds to make one output converge (to a desired value) could cause divergence of the other outputs. Not only would learning take much longer, but it would seem that accurate results may be far more difficult to come by.

Either approach will be trained using the same dataset, which is a database of several million recorded games of varying skill levels (above 1400 ELO). An interesting characteristic is that the level of play the ANN is capable of may be related to the skill level of the games used in training. If so, then it would be possible to pre-specify the quality of performance when training the network. Although this was not considered to date, it could be an interesting area to examine once the network design is improved to a point where accurate performance rating becomes possible.

The Final ANN System Design:

The final ANN system design which was implemented resembles the geographical approach shown in Fig. 2. It was eventually decided that the implementation of "adaptive resonance"—the retraining of erroneous moves, would be unnecessary, as it would be both highly impractical and probably have little benefit considering the quantity of original training data verses the number of games which could be played in the project's timeframe. The final neural network system design is shown in Fig. 4.

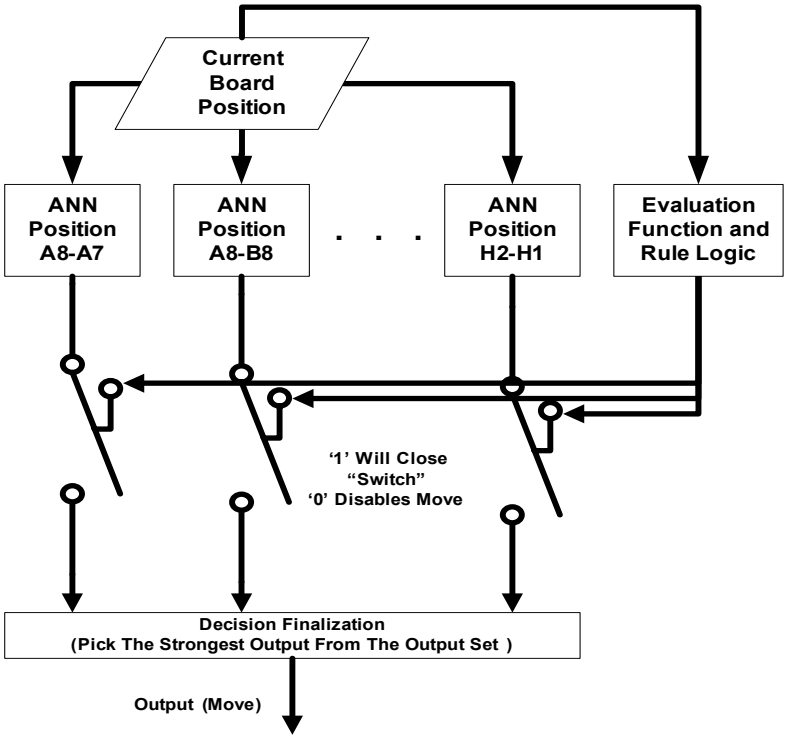

Figure 4. Final ANN System Design

In order to define the legal moves and therefore the required ANN subsystems, it was decided to define a new chess piece that has the mobility of the queen plus that of the knight. Fig. 5 shows how this piece could move by the red highlighted squares. If the piece is moved around the board from the top left to the bottom right, all legal moves will be defined. An ANN was then defined for each legal move. It is important to realize that the legal move concept *does not depend on the piece*, and also does not look at the rest of the board. Additional logic determines if each "potentially" legal move is actually legal given the board condition at the time. Note that the technique shown in Fig. 5 is the "practical" method of defining the networks described on page 8.

Time permitted the training and testing of only one network design: 2 hidden layers with 128 nodes in each and a single output node. The network connectivity was set to 75% for this network series. Future work should concentrate on finding an improved network design and studying the results of varying the network connectivity.

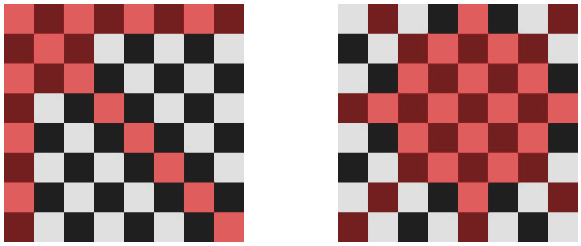

Figure 5. Finding all Legal Moves

Interface Module:

The interface module seen in Fig. 1 (in playing or advisory mode) is an application the user interacts with on the PC. It displays a chess board and the piece layout for the game, and the user makes their moves on this board. The interface module must be integrated with the artificial neural network system (ANN system) to exchange board position data and move choices. The interface is also capable of playing against a chess engine, as was done to obtain performance data for the ANN system. The interface application is shown in Fig. 6.

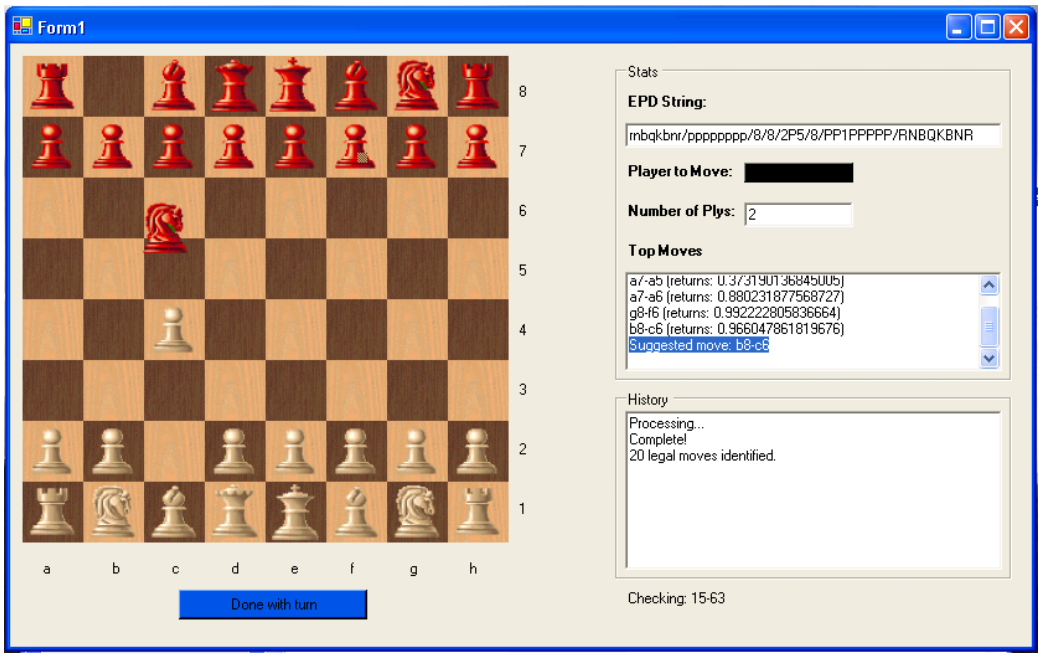

Figure 6. Interface Application Screenshot

Game Database:

The game database is a standalone application called ChessBase 9.0. This application allows a massive database of several million saved chess games to be sorted, filtered and searched by move. ChessBase 9.0 is used exclusively in the learning mode, at the start of the preprocessing stage. All game data must be preprocessed in order to convert it from the database format to a format usable for neural network training. Games from ChessBase 9.0 are extracted by searching for specific moves, which defines the first step in the preprocessing module to be described shortly. Many individual sets of games are made, which will then be passed to the remaining preprocessing stages.

Data Preprocessing:

Data preprocessing begins in the game database described above, and continues with a series of applications designed to convert the chosen game data into neural network training vectors. The data processing stage converts PGN files [8] (portable game notation—a popular algebraic game recording standard) which are extracted from the database, to EPD strings (extended position description). EPD files contain a series of strings representing the full board position at each move recorded in the PGN file. Fig. 7 shows a sample of PGN format, and Fig. 8

shows a sample of EPD format. At this time a detailed understanding of file content is not required.

```
1. e4 d6 2. d4 Nf6 3. Nc3 g6 4. Nf3 Bg7 5. Be2 O-O 6. O-O Bg4 7. Be3 Nc6
8. Qd2 e5 9. d5 Ne7 10. Rad1 Bd7 11. Ne1 Ng4 12. Bxg4 Bxg4 13. f3 Bd7 14. f4
Bg4 15. Rb1 c6 16. fxe5 dxe5 17. Bc5 cxd5 18. Qg5 dxe4 19. Bxe7 Qd4+ 20. Kh1
f5 21. Bxf8 Rxf8 22. h3 Bf6 23. Qh6 Bh5 24. Rxf5 gxf5 25. Qxh5 Qf2 26. Rd1
e3 27. Nd5 Bd8 28. Nd3 Qg3 29. Qf3 Qxf3 30. gxf3 e4 31. Rg1+ Kh8 32. fxe4
fxe4 33. N3f4 Bh4 34. Rg4 Bf2 35. Kg2 Rf5 36. Ne7 1-0
```

Figure 7. Sample of the PGN File Standard

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - pm d4;
rnbqkbnr/pppppppp/8/8/3P4/8/PPP1PPPP/RNBQKBNR b KQkq d3 pm Nf6;
rnbqkb1r/pppppppp/5n2/8/3P4/8/PPP1PPPP/RNBQKBNR w KQkq - pm Nf3;
rnbqkb1r/pppppppp/5n2/8/3P4/5N2/PPP1PPPP/RNBQKB1R b KQkq - pm b6;
rnbqkb1r/p1pppppp/1p3n2/8/3P4/5N2/PPP1PPPP/RNBQKB1R w KQkq - pm g3;
rnbqkb1r/p1pppppp/1p3n2/8/3P4/5NP1/PPP1PP1P/RNBQKB1R b KQkq - pm Bb7;
```

Figure 8. Sample of the EPD File Standard

Of course, neural networks require a numeric input vector for training. It is therefore required to convert the EPD stings into floating point input vectors by converting the EPD characters to floating point values. A sample geographical training vector is shown in Fig. 9. The output pattern is 1 or -1, corresponding to whether or not to make the move specific to the training data set and the neural network. The hope is that after learning enough patterns, the neural network will be able to effectively make move decisions for data samples which are not contained in the original training set.

```
# Input pattern 1:
0.5 0.4 0.3 0.9 1 0.3 0.4 0.5
0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
-0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1
-0.5 -0.4 -0.3 -0.9 -1 -0.3 -0.4 -0.5
# output pattern 1:
1
```

Figure 9. Sample Floating Point Training Sample

Playing/Advisory System:

The interface module and the ANN system are used in the playing/advisory system. Fig. 10 describes the operation of this entire system (or functional mode). In Fig. 10, the ANN board processing stage is expanded on the far right to give more insight into the ANN system. It is possible to treat the ANN processing as a single function as it has a clearly defined input and output, and its inner workings are of no consequence to the interface module so long as data is passed back and forth. In the final system implementation, a list of moves is returned in all cases. The "top" move is recommended in a message box. All other moves are listed with their corresponding "score" so that the user may view other possibilities. Thus, the playing and advisory modes of operation are actually combined in the final system.

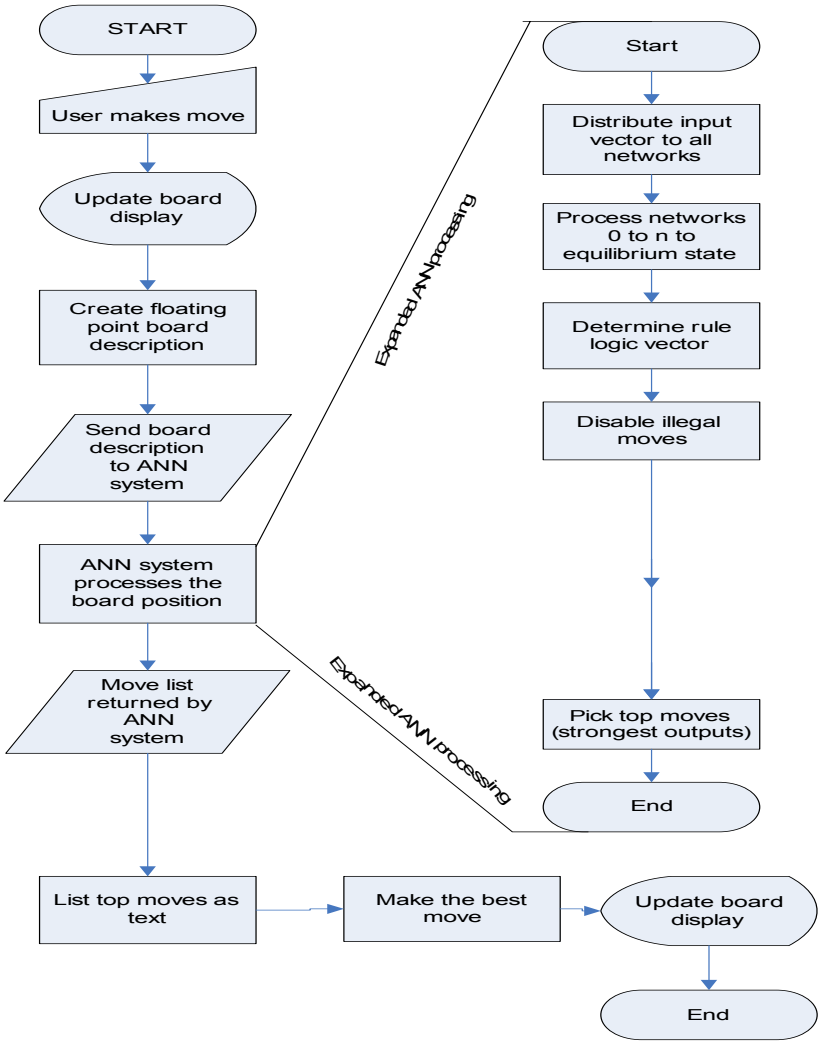

Figure 10. Playing/Advisory System Flowchart

Learning System:

The learning system is comprised of the ChessBase 9.0 database, the preprocessing software, and the ANN system. It is possible to divide learning into two distinct functions: data preprocessing and training.

The data preprocessing function is described in Fig. 11, and introduced above. The database specific operations and selections can be considered a unique function, and an expanded view is provided at the bottom of Fig. 11. When considering the geographical design paradigm, it is vital that the final output of the preprocessing stage is a set of training vectors specifically created for each possible legal move in chess, following the data format displayed in Fig. 9. A great deal of research and effort was devoted to the preprocessing stage, as the functionality required does not exist in any single known application. It is required to use numerous applications to convert between formats, the names of which are displayed in the appropriate functional locations in Fig. 11 below.

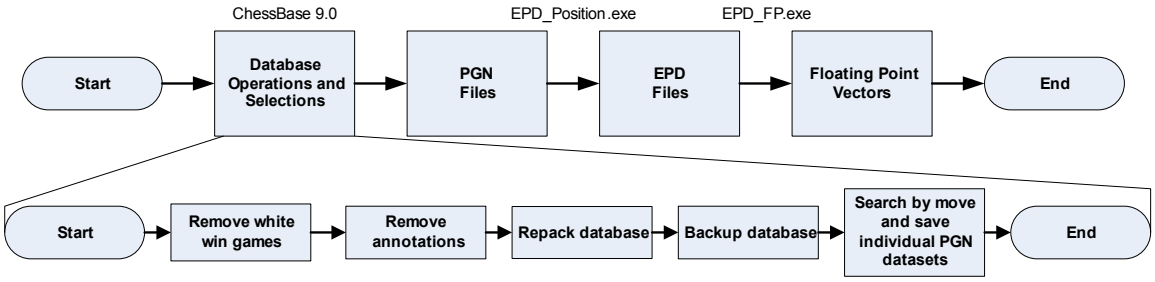

Figure 11. Data Preprocessing Subsystem Flowchart

Training takes place after the data preprocessing is complete. The end result of preprocessing is a set of "pattern files" which are loaded by the Stuttgart Neural Network Simulator (SNNS) to perform the training [5]. Each network has its own pattern set, and training takes place one network at a time. Each training pattern is once again in the format described in Fig. 9. Training is performed entirely by SNNS, so the mathematical details of learning will not be described in this document. It is possible to instead treat the actual learning as a self contained process. Fig. 12 describes the training process at a high level.

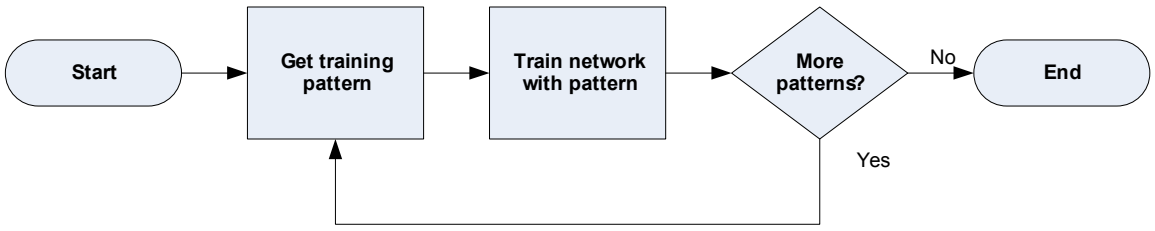

Figure 12. Training Process Flowchart, Using Stuttgart Neural Network Simulator

Training is actually repeated numerous times for each network. Each time the entire pattern set is trained is known as a cycle or epoch. 50 cycles are processed for each network.

A more complete picture of the training process is shown in Fig. 13, and can be considered as a set of 4 main procedures. Once the data is formatted as shown in Fig. 9, training

may start. It takes about 18 minutes to train each network. Because 1792 networks must be trained in all, the procedure must be distributed to several computers. A central server holds the training data (about 40 million patterns in all), and the individual clients will download this data as needed based on their training assignments. The completed network is uploaded to the server once training is completed.

The first component of the training procedure, called the "shell," is meant to create the scripts and batch files required to interact with the ANN simulator program "Stuttgart Neural Network Simulator" (SNNS). The next training function downloads the data from the server and initializes a new network. The training file is created by mixing training samples from the dataset. It is important that both 'yes' and 'no' patterns are trained for each network. Finally, SNNS is used to load the initialized network file, as well as the training file created in the last step. Training takes place by interacting with the SNNS interface through scripts. The actual training algorithm used is known as resilient back-propagation. This is an adaptive learning rate algorithm, and it was used after standard back-propagation failed to train the networks.

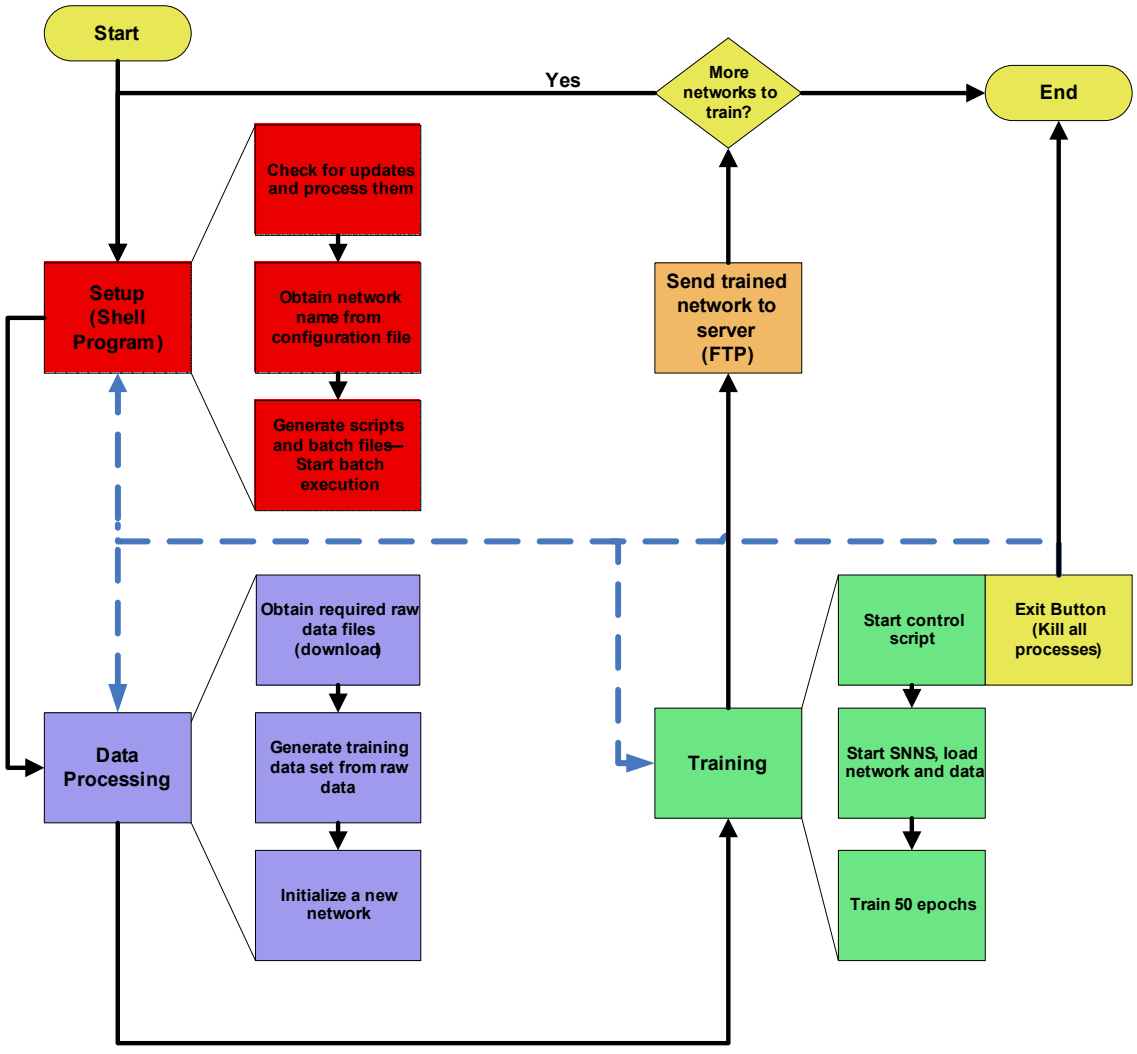

Figure 13. The Expanded Training Procedure

The Evaluation Function:

After the networks were completely trained, the system was found to lack the performance desired. After examining the moves returned by the ANN system (including the ANN score) it was found that several legal moves returned ANN outputs within very close proximity to one another. This may likely be a result of training with +1 and -1, which led to neuron saturation.

It quickly became apparent that several moves were returning outputs that matched up to 3 or more decimal places. In this case, there is absolutely no meaning in the variation, and both outputs are effectively equal. Thus it becomes difficult to say for certain which output is the "best" move to make.

The decision was made to create an evaluation function which would return a numerical score of a future board (which would exist if a given move was made). At first, an attempt was made to create an evaluation function which would look at four parameters, all of which are defined as the change from the current board to a possible future board:

- Material (ΔM)
- Threats (ΔT)
- Mobility (ΔO)
- Vulnerabilities (ΔV)

Although it can be left out of the final equation (by only looking at the top 10 or 20 returned moves), a fifth parameter, NN output (Y), could also be used:

$$\text{Score} = a*\Delta M + b*\Delta T + c*\Delta O + d*\Delta V + Y$$

It was determined that this approach was failing to produce an improved result, and the decision was made to utilize an existing evaluation function found within a chess engine. While several chess engines were tested, the most compatible engine found was "Gaviota."

The integration of the chess engine meant that scores could be returned by entering a board position and the side to move. The result, a floating point value between -20 and 20, could then be used in determining the best move; either alone or in conjunction with the NN output. The final implementation uses the NN output as well as the evaluation score in a scaled summation:

$$\text{Score}=\text{Evaluation Output}+A*\text{NN\_Output}.$$

It was initially unknown what "A" should be set to. In fact, this became the subject of experimentation, as it was predicted that an optimum value existed.

Results and Conclusions:

In order to compare performance accurately, it was required to integrate an engine (Gaviota engine is used) which uses a search depth of 0—which means the evaluation function is used alone (lines are not created). The NN system will automatically play against the chess engine. Experimentation revealed that the neural network system would also need to use this same evaluation function in conjunction with ANN outputs (as described above). In the end, the ANN system evaluates a move based on (ANN output + A*Evaluation score), where A is an experimentally determined constant.

Three test groups were defined: A=0, A=2, and A=10. A series of board positions were extracted from the database, all occurring 5 plies into the game. This was done to ensure a variety of games could be played. The system is deterministic, so a single position (such as the ply 0 position) would always be played in exactly the same way. All three trial groups play games with the same set of starting positions.

Each game is played for 40 plies (or until a side wins). Each game is recorded with the current score for black (as returned by the NN System). After playing as many games as possible, the scores were averaged for each ply (within each trial group) in order to get a general idea of relative performance. 107 games were played for the A=2 case. A software bug has prevented more than 20 games from being played in the A=0 or A=10 case, although this should still be enough to get a rough estimate of performance.

Fig. 14 clearly shows the A=0 case (in pink) remaining close to a score of 0. This is exactly what is expected, as it is showing "evaluation function vs. evaluation function," and neither side should have an advantage. Prior to commenting on the A=2 and A=10 case, it is required to perform a correction on the plot seen in Fig. 14. Because the scores were taken from the NN system, the A=2 case will have approximately +2 added to the average and A=10 should have roughly +10 added to the average. These additions merely account for the NN score's contribution. However, it is required to subtract this factor as it does not originate from the engine, which is the only factor involved in determining the control group (A=0) score. This idea is best seen in looking at the scores at ply 1 in Fig. 14. At this point, all groups should have a score of nearly 0, as the game has just started. It is clear that each group has an offset equal to A. The correction is made, and Fig. 15 is the result.

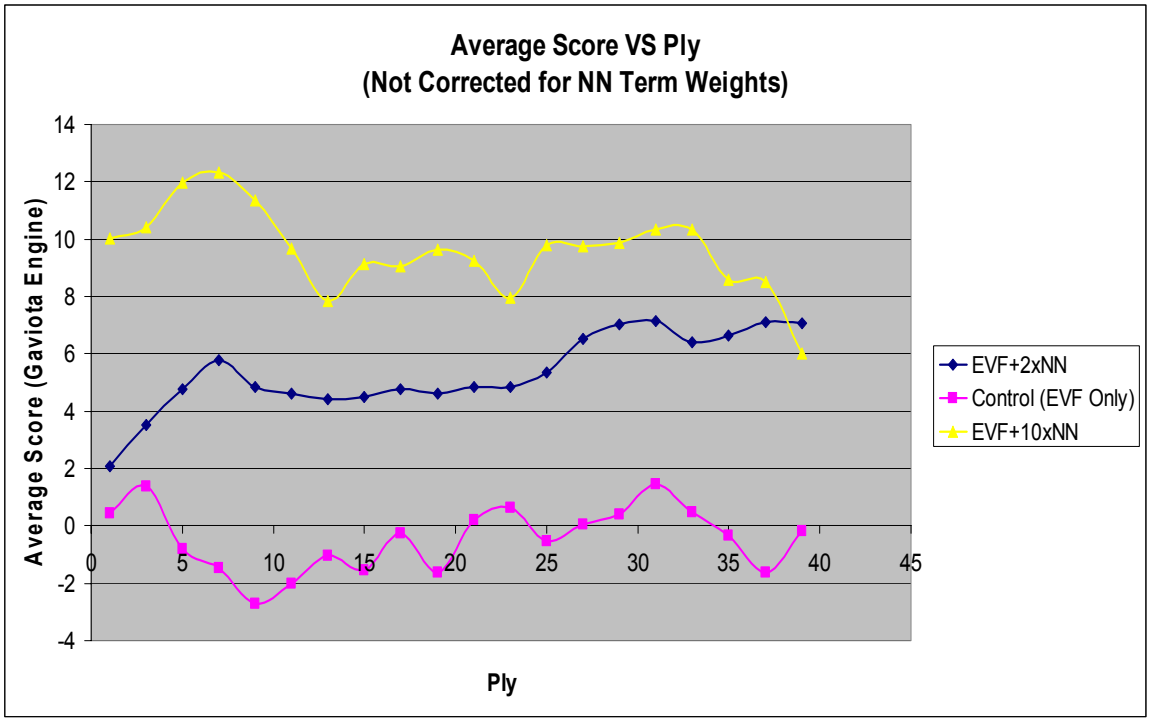

Figure 14. Average Scores by Ply (Non-Corrected)

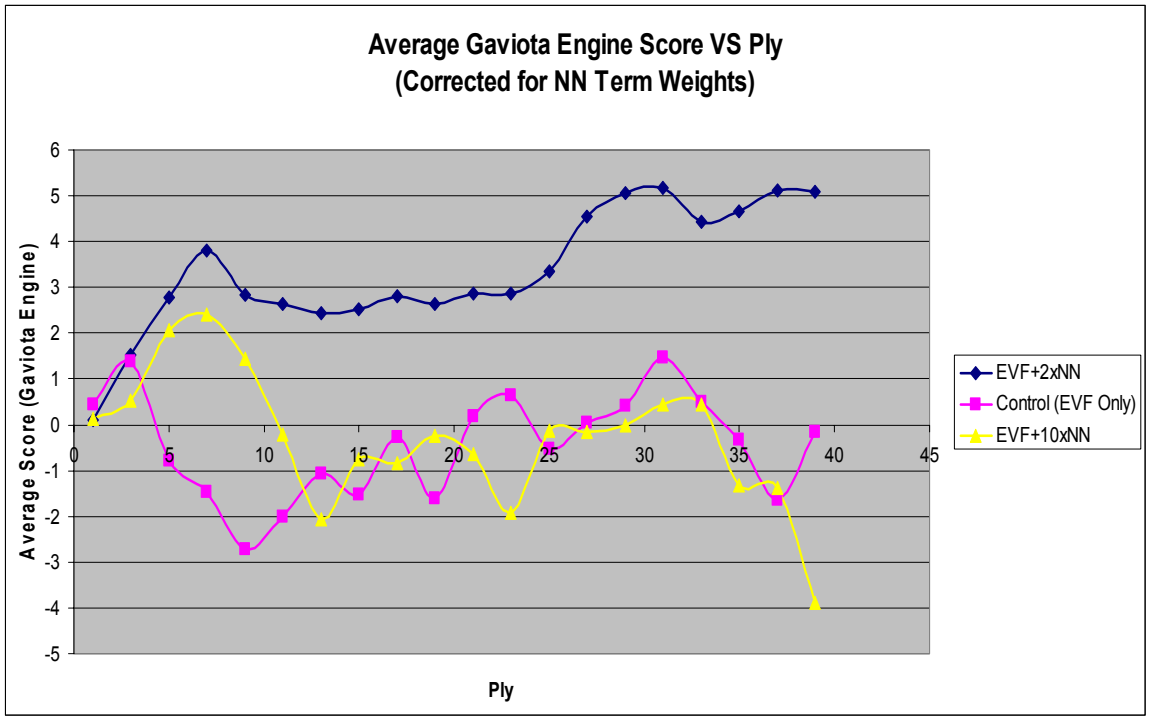

Figure 15. Average Scores by Ply (Corrected)

From Fig. 15, it seems apparent that the A=2 group has performed considerably better than the control group or the A=10 group. This seems to support the idea that there is an optimum value for A, although A=2 may not be it.

Most importantly, the fact that a sustained lead is observed in a group utilizing NN output indicates that the system is indeed contributing something to the evaluation of moves. Although the lead is modest (about equivalent to a knight's advantage), it does show some promise for further development of this system.

The A=10 case seems to carry an advantage within the first 5 or 7 plies, but it is apparent that the system is no better (and possibly worse) than using the evaluation function alone. The close output proximity of the top moves is possibly the cause of the poor performance. The error associated with the NN outputs is multiplied by ten, which will outweigh the more precise evaluation function outputs, which are very small (under 1) when neither side has a clear lead. In the A=2 case, the error due to output proximity is "corrected" in part by the evaluation function. It is important to notice that as the score (lead) increases, the NN output will actually make up a diminishing portion of the score as the evaluation function component will increase while "A" stays constant. An interesting experiment would look at the results of adjusting "A" to follow an increasing evaluation score.

Although the system is able to maintain a lead with A=2, a great deal of additional work is needed to develop the system to a point where it would play effectively against human players or fully enabled chess engines. In particular, the endgame would have to be strengthened. Admittedly, it does not seem that this system possesses the ability to play effectively in the endgame, where a high level of precision is required. It may be possible that further training

would improve the results, but it seems traditional approaches of using endgame tables would work best here as full knowledge of this specific problem already exists.

There does seem to be some feasibility in using ANNs as an alternative mechanism for playing chess, although the application may be somewhat limited to a contributing role in move evaluation. The results of this project indicate that ANNs can contribute to the evaluation of a specific move if they are given limited weight in conjunction with a traditional evaluation function. A great deal of further research would be needed to perfect this system to a point where it could reasonably play against a human opponent. It seems that a standalone ANN system, while obviously not proven to be impossible, will remain out of reach for quite some time.

References:

[1] K. Chellapilla and D.B. Fogel. “Evolution, Neural Networks, Games and Intelligence.” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1471-1496, Sept. 1999.

[2] K. Chellapilla and D.B. Fogel. "Anaconda Defeats Hoyle 6-0: A Case Study Competing an Evolved Checkers Program Against Commercially Available Software,” in *Proceedings of the 2000 Congress on Evolutionary Computation*, 2000, vol. 2, pp. 857-863.

[3] C. Posthoff, S. Schawelski and M. Schlosser. “Neural Network Learning In a Chess Endgame,” in *IEEE World Congress on Computational Intelligence*, 1994, vol. 5, pp. 3420-3425.

[4] R. Seliger. “The Distributed Chess Project.” Internet: http://neural-chess.netfirms.com/HTML/project.html, 2003 [Aug. 26, 2004].

[5] University of Tübingen. “Stuttgart Neural Network Simulator.” Internet: http://www-ra.informatik.uni-tuebingen.de/software/JavaNNS/welcome_e.html, 2003 [Aug 30, 2004].

[6] M. Chester. *Neural Networks*. Englewood Cliffs, NJ: PTR Prentice Hall, 1993, pp. 71-81.

[7] FIDE Handbook E.I.01A:2001, World Chess Federation Laws of Chess.

[8] S.J. Edwards, et al. Standard Portable Game Notation Specification and Implementation Guide, March 1994.

[9] Hogan, M.A., "Modular, Hierarchically Organized Artificial Intelligence Entity," U.S. Patent no 6,738,753, Aug. 2000.