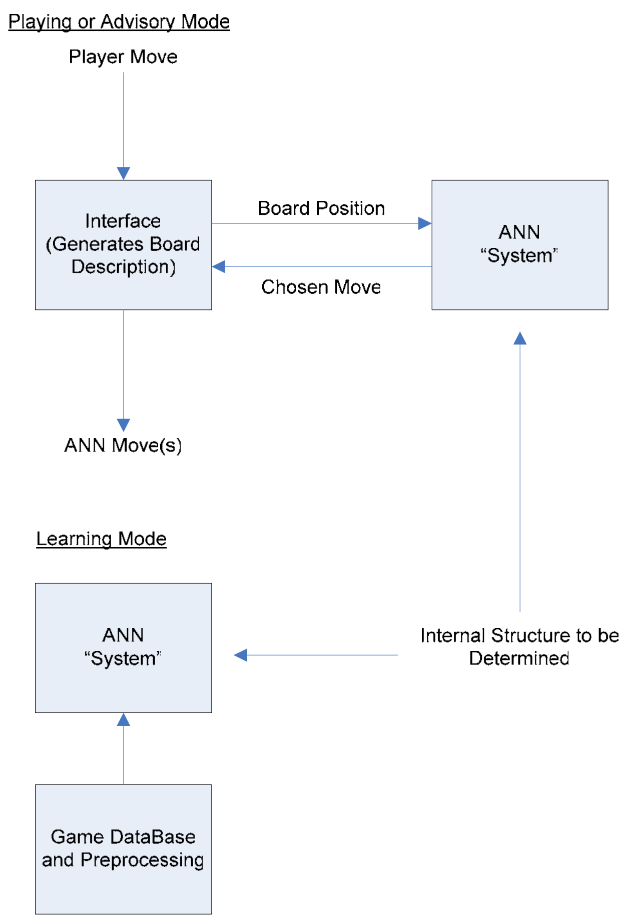
Complex Decision Making With Neural Networks: Learning Chess
Functional Description

Jack Sigan
Bradley University ECE Dept
Dr. Aleksander Malinowski, Advisor
October 6, 2004

Objective and System Description:

Although mainly a research endeavor, the end goal of this project is to produce a system based on artificial neural networks (ANNs) which will play chess (as the dark side only) effectively against a human opponent. The dark side is chosen simply because it prevents the system from having to make the first move. To meet the objective, research will be centered on artificial neural network (ANN) topology, specifically for the purpose of creating a topology appropriate for complex problem solving. Functionality of the system may be broken into three parts; the learning mode will be an automated process where the ANNs learn how to play from an extensive external database of games, while the playing and advisory modes of operation accept user interaction, taking move inputs from the player and providing the move(s) chosen by the system. Figure 1 illustrates the two operating modes. The training mode must precede the playing and advisory modes. 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 2 Operating Modes



Input and Output Descriptions:

As shown in figure 1, three main components will make up the system; the interface module, the ANN module, and the preprocessing/database system. The ANN module is shared by both the learning and the playing/advisory mode. The only difference in the output between regular playing mode and "advisory mode" is that a list of "good" moves will be returned in advisory mode, while only one will be returned in play mode. Table 1 displays the connections and IO paths which will comprise the modes of operation. This information is shown in figure 1 as a system block diagram. Note that this IO plan is very high level, and that far more complexity is to be 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.

Table 1:
Input/Output Descriptions by Module

Learning mode:

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

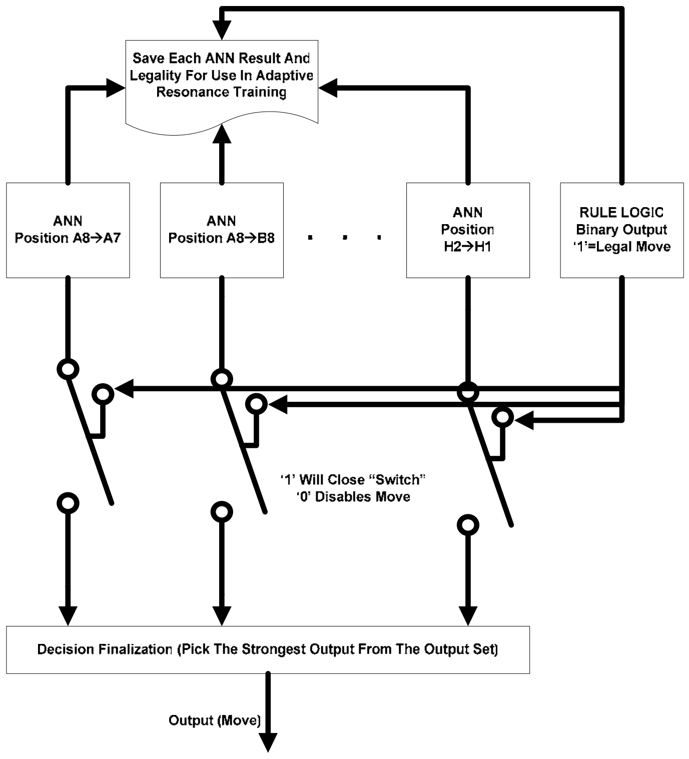
Playing/Advisory mode:

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

Structure of the ANN Modules:

Figure 1 depicts the ANN module in both the learning and playing/advisory mode, and it must be emphasized that the design of this component is the primary focus of the project's research. Presently, two major design paradigms are being examined.

Figure 2:
Geographically Derived ANN System Design

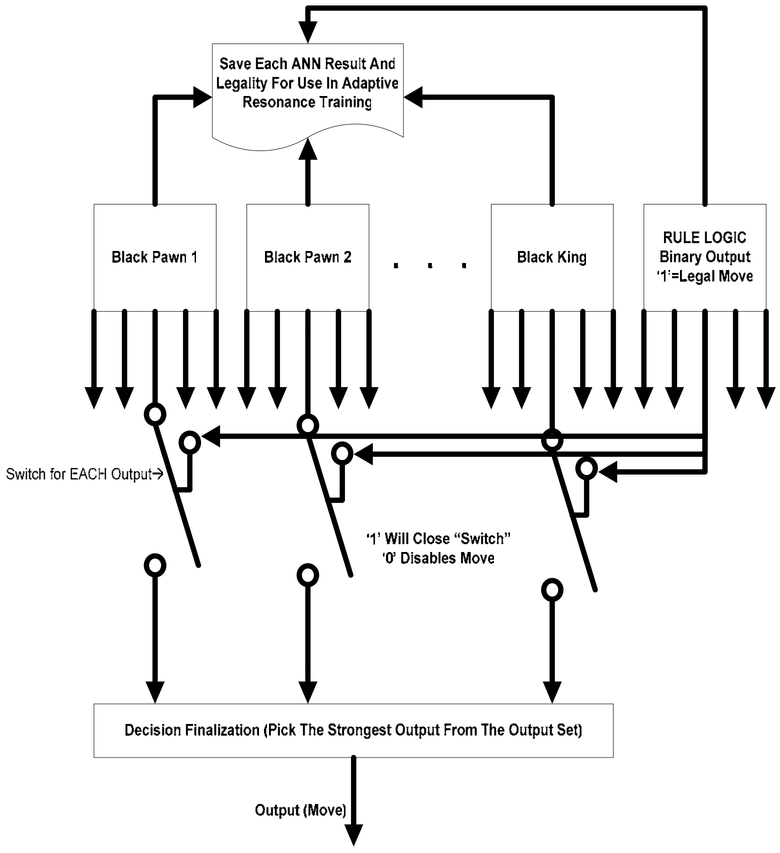


Save Each ANN Result And Legality For Use In Adaptive Resonance Training
ANN Position A8→A7
ANN Position A8→B8
. . .
ANN Position H2→H1
RULE LOGIC Binary Output '1'=Legal Move
'1' Will Close "Switch" '0' Disables Move
Decision Finalization (Pick The Strongest Output From The Output Set)
Output (Move)

The first will involve 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 $b_i Є g$ 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, we can 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 we must consider the castling moves to be special cases, as they involve more than 1 piece. If we consider a whole game of *n* moves, then all legal moves made throughout 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 $b_i€g$. M is finite, obviously having a value less than 64x63 moves. This idea is fundamental to justifying the design in figure 2. In this design, we must create an individual ANN structure for all moves t, $t€M$. Thus, each ANN will be trained to make a yes or no decision for its own move 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 $m_{if}€M$! 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 do we determine what to do? This is where we apply our knowledge of m_{if} as defined above for the specific board position in question. We can hereby deactivate the ANNs which want to make illegal moves. Once this is done, we simply choose the ANN with the highest output level (from a hyperbolic tangent function) as the move to make.

Over time, we should in theory be able to modify the ANNs so they do not make these 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



Save Each ANN Result And Legality For Use In Adaptive Resonance Training
Black Pawn 1
Black Pawn 2
. . .
Black King
RULE LOGIC Binary Output '1'=Legal Move
Switch for EACH Output→
'1' Will Close "Switch"
'0' Disables Move
Decision Finalization (Pick The Strongest Output From The Output Set)
Output (Move)

The second approach initially proposed is to break the game of chess apart by piece, thereby allowing a more "functional" approach to the same problem. It is required to define 32 networks 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 $l = \sum_{i=0}^{n} f(p, b_i)$. Once again n is the number of turns in a single game. Defining l is only slightly different from how L was defined above, as it must have extra emphasis placed on p. We will still have the same number of outputs in total (M). Outputs will still be deactivated in the same fashion as described above. Adaptive resonance will still be applied to improve the network decisions over time. Figure 3 shows the functionally derived approach.

In general, both design approaches may be viewed as parallel approaches to complex problem solving. Many benefits are achieved in such an approach, including shortened training time and simplified internal network design. Although it is not known in advance how large the networks must be for either approach, it seems that the networks for the geographical approach will 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 is assumed to be less complex, thereby requiring fewer nodes to achieve accurate results. It is 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 will be left largely to trial and error. Functional networks clearly will have varying sizes, simply because queen movement, for example, is more complex than that of the pawn.

Another benefit of the parallel approach is that learning is 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 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. 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 the initial goal is to simply train the system to play without regard to skill level, this could be a very interesting area to look into if time and success permits.

Motivation and Preliminary Proof of Concept:

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

Applied correctly and appropriately designed, artificial neural networks (ANNs) 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, demanding 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, 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, although it is ironic that the author 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 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 facts, proposed improvements, and possible implications of this kind of research to the field of artificial intelligence, the project is worth pursuing.

Implementation Considerations:

Work thus far has led to the project's description as described in this document. A major focus over the summer of 2004 was to develop a generic and versatile ANN framework, including all processing, learning, and management functionality. Some time has also been spent determining the best way to store training records, including the data representation, as well as normalization and standardization considerations. Although work on the ANN platform is estimated to be roughly 80% complete at this point, the choice was made recently to discontinue work on the ANN framework and instead utilize the existing Stuttgart Neural Network Simulator (SNNS) with Java interface [5]. Work is now focusing on network topology design, network size considerations, and integrating the rule logic with the ANN components.

It is required to have some form of data processing in addition to the ChessBase 9.0 software being utilized for the training data. The processing function will take the games stored in PGN files (portable game notation, a common algebraic chess notation standard) and convert them to extended position description (EPD) format. The result for one game would be a series of strings with one string for each board position in the game. These strings are then to be parsed by another application which will convert them into a series of 64 signed floating point values corresponding to the traditional weights given to chess pieces. The value of the move to be made will be concatenated with the resulting vector to produce a training record. Once a set of training records is generated, they will be presented randomly to the ANNs for training.

The networks themselves will be generated to work with SNNS, as will the training records. Multiple PCs will process the training files in the learning stage, and human supervision will not be required. It is expected that learning will take hours to a couple of days for the entire system. Rule logic is not considered in the initial training process.

A separate application will be created for the interface to be used in the playing mode. The application should present the user with a graphical view of the board, and be able to read the SNNS files and process the network after a new input vector is generated when the user makes a move. The resulting move must then be shown on the board.

The networks themselves are currently being considered with hyperbolic tangent activation functions, using classical back propagation learning. It may become clear in the near future that another learning style is more appropriate (perhaps back propagation with momentum, etc), but for now as many simplifications as possible are being sought, so learning will be kept simple for the time being. Positive elements on the input vectors will be scaled to the dark side (ANN) while negative elements will be scaled to the light (human) player. A zero means no piece is present on the square in question. The outputs of the ANN (tanh) are to be considered 'yes' if they are positive, and 'no' if they are negative. More positive moves (approaching +1.0) are the ones most likely to be made, while the most negative (-1.0) are illegal or very poor can never be made. As research and design continues, it will doubtlessly become apparent that numerous additions and modifications to the preliminary concepts will be required.

Bibliography:

[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].