Complex Decision Making With Neural Networks: Learning Chess

Jack Sigan
Dr. Aleksander Malinowski, Advisor
Bradley University Department of Electrical and Computer Engineering

Objectives:

This research project asks if an artificial neural network (ANN) can be trained to play a full game of chess. Prior to reaching this ultimate goal, it is required not only to develop the neural network system, but also the process to derive this system.

Neural Networks:

An ANN can be thought of as a parallel processing structure. It is a mathematical attempt to model the biological neural networks found in animal brains—with a special focus on mimicking the learning process. While it is obvious that ANNs are incredibly crude in comparison to even simple biological networks, they have been found useful in solving certain classes of problems involving non-linear mapping, function estimation, and prediction where traditional mathematic models are unknown or non-existent.

Figure 1 shows a generic structure of an ANN. Each square represents a node or neuron—the simple computation element of the ANN. It is in the parallel architecture of these nodes that the ANN gains its power. A close up of the node structure is shown in Figure 2. The node performs a summing operation of all inputs multiplied by their corresponding weights. The output of a single neuron is computed by the activation function, which takes the total node input as its argument.

Neural networks are trained by adjusting the weights of each individual node. The weights are adjusted to allow the outputs of the network to match the desired outputs for a specific input pattern. During training, many input patterns are presented, and weights are modified each time to obtain the proper outputs. Eventually, the outputs will be correct (or almost correct) for all patterns. The network should provide correct answers for patterns not found in the original training set after training is complete.

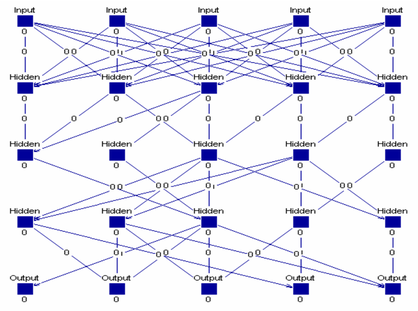
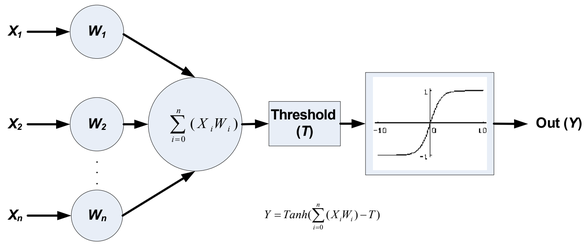


Figure 1 (left):
Neural network structure showing the parallel arrangement of the nodes. The weights are shown between neurons, with initial values of zero. During training, these values will be changed.

Figure 2 (below):
Individual node structure, showing the node output as a function of the inputs and input weights. Thresholds are subtracted from the argument of the activation function.

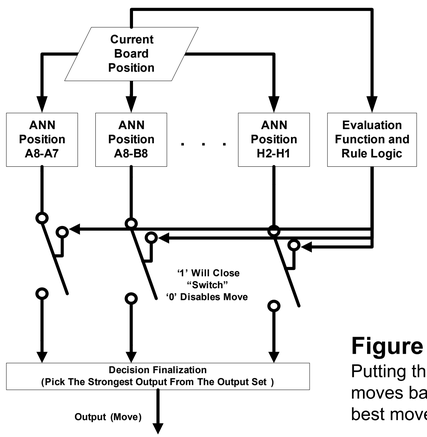


Project Specific Network Features:

The ANNs used in this project utilize the hyperbolic tangent as the activation function. This function is chosen because it provides a range of -1.0 to +1.0, which is very convenient for the purpose of each network in this project. As will be seen shortly, the ANN system is really composed of over 1800 individual neural networks of about 300 nodes—trained to decide 'yes' or 'no' with regards to making a given move in chess. An individual network exists for every legal move, which is defined by an initial and a final position without regards to the piece in question.

Defining the Legal Moves:

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. Figure 3 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. Future logic determines if each "potentially" legal move is actually legal given the board condition at the time. Figure 4 shows how the individual ANN components are linked together. When a move is to be made, all network outputs are considered.



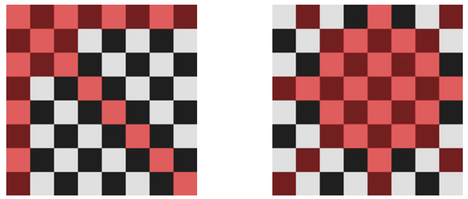


Figure 3 (above):
Defining the legal moves in chess by making a new piece composed of queen moves plus knight moves. There are 1856 such moves.

Figure 4 (left):
Putting the networks together in the ANN system. An external logic block will disable illegal moves based on the current board position. The decision finalization block will choose the best move based on ANN output and a score from a board evaluation function.

Describing the System:

The system proposed by this project consists of two modes of operation. The first mode is the training mode, which is the process of deriving the individual neural networks for use in the playing mode. The fundamental idea behind the training is that an ANN can learn game strategy by the moves made in existing games. The playing mode allows the user or a computer chess engine to interact with the ANN system developed in the training stage. Figure 5 shows a system block diagram of both modes of operation.

The learning mode is by far the most complex and involved aspect of the project, due largely to the enormous quantity of raw chess data which must be preprocessed in order to create ANN training data. The training process itself is distributed to several computers, but still requires weeks of processing. The raw game data is obtained from a database, known as ChessBase 9.0. This database contains millions of saved chess games, stored in a compressed format. The preprocessing procedure must take the compressed data and expand it to training elements—or vectors— for the neural networks. The preprocessing procedure is shown in Figure 6. The desired format for ANN training data is shown in Figure 7. The last line of the training vector contains the desired output—whether or not to make a certain move. 1 means 'yes' and -1 means 'no.'

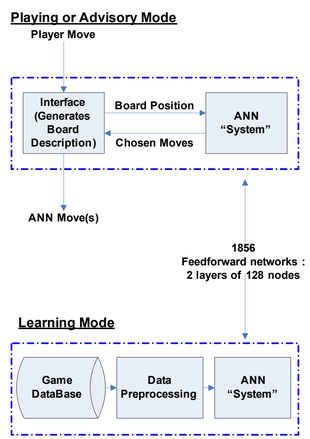


Figure 5 (left):
The two main modes of operation for the system. The learning mode must always precede the playing mode, and is in place to derive the networks used in figure 4.

Figure 6 (below):
The data preprocessing procedure. Game data is taken from the database and processed to match the format of a training vector, as shown in Figure 7.

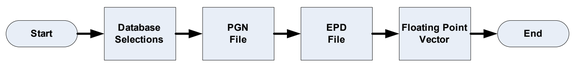


Figure 7 (right):
An example of the ANN training pattern. The top components are the inputs, and the bottom component is the desired output, which corresponds to 'yes' or 'no.'

```
# 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
```

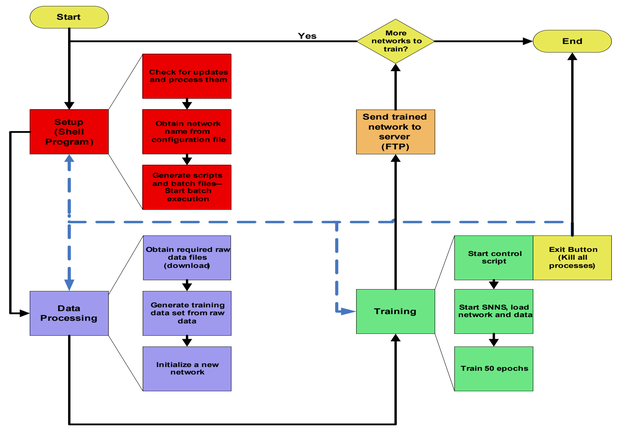
Training:

Once the data is formatted as shown in Figure 7, training may start. It takes about 18 minutes to train each network. Because 1856 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 training procedure can be considered as a set of 4 main procedures, as shown in Figure 8. The first part, 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 8 (below):

The training procedure can be viewed as a set of 4 individual processes.



Evaluation and Testing:

Testing the performance of the system requires an interface as shown in Figure 9. This interface is designed to allow a human player or a computer chess engine to play against the ANN system.

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. Experimentation revealed that the neural network system would also need to use this same evaluation function in conjunction with ANN outputs. In the end, the ANN system evaluates a move based on (ANN output + A*Evaluation score), where A is an experimentally determined constant.

Initial findings indicate that the ANN system performance is superior to that of the standalone evaluation function when 'A=2' is used. Other values of A are currently being tested.

Figure 9 (right):
The game interface which can work with a human player or a chess engine. It is possible to play games from the start or from an arbitrary start position. If playing a computer chess engine against the ANN system, the play if fully automated to ease the data collection process.