2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Comparison Of Artificial Neural Network Architectures And Training Algorithms For Solving The Knight's Tours

Robert G. Escalante and Heidar A. Malki, Senior Member
College of Technology
University of Houston
malki@uh.edu

***Abstract*--**This paper compares known architectures and training algorithms of multi-layered artificial neural networks (ANNs) and their ability to solve one specific type of chess problem (the Hamiltonian Cycles known as the Knight's Tours). The networks used here are trained to solve tours starting from the center of a five-by-five chessboard. The ANN architectures focused on are the function approximation feedforward architecture; the classification feedforward architecture; and two customized variations of the function approximation feedforward architecture. The training algorithms compared include Levenberg-Marquardt; variable learning (adaptive) gradient descent; resilient backpropagation; the Fletcher-Reeves conjugate gradient; and the Broyden-Fletcher-Goldfarb-Shanno Quasi-Newton algorithm. These five training algorithms are used for each of the four architectures. The real world application of this research is limited only by the functions given and the parameters used to define the environment. This can be expanded to include a variety of chess pieces (functions) in a variety of environments (any *m*-by-*n* sized board or even multidimensional problem solving). The practical limitation on the application is the software necessary to run extremely large neural networks.

I. INTRODUCTION

For the last three decades, chess has been the main focus of many artificial intelligence (AI) efforts. There are over 250 distinct Chess engines capable of playing master level chess. They do this by using advanced techniques such as: using the minimax algorithm for finding best moves; alpha-beta pruning to reduce the tree searches; and using databases of openings and endgames. In other words, the AI quest for better computer chess players has not led to superior chess engine architecture but to better sorting techniques and better position evaluation algorithms. They may play chess better, but they are not closer to human intelligence [1].

In 1997, Deep Blue achieved more wins than losses over then reigning World Chess Champion, Garry Kasparov. However, Deep Blue was an expert system. While it could play excellent chess and it could defeat the world champ, it was also a special "Kasparov-beating" machine that may have been unable to defeat some of the other Grandmaster chess players such as Vladimir Kramnik and Vishwanathan Anand. To date there is much debate over whether Deep Blue defeated Kasparov or whether Kasparov, attempting to get tricky, defeated himself. In post game interviews Kasparov said that he prepared with the wrong strategy for defeating Deep Blue. There were also accusations that the IBM team cheated by using a human. They never presented evidence that they did not (sealed printouts) and Deep Blue was immediately dismantled after winning the match [2, 3].

As a learning system example, in 1999 Chellapilla and Fogel made evolved neural networks compete for survival using checkers. Without any of the endgame databases or opening book tables used by expert systems, these evolved neural networks competed for 250 generations with winners given the rights to bare offspring networks. The end result, two weeks later, was a network that had learned by evolution algorithm of its parent networks how to play checkers good enough to beat two experts and draw a master [4].

II. LITERATURE REVIEW

These are the ways in which the Knight's tour problem has been solved in the past: batch move technique; batch path technique; and pure mathematical methods.

In the batch move technique a random bunch of Knight moves is selected with the initial condition that none of the moves is repeated. A neural network is fed the moves as inputs and it checks to see if they connect in a continuous pathway. If the pathway is not continuous then the tour is not complete and the Knight moves are adjusted. This adjustment consists of erasing some of the Knight moves and replacing them with other random moves. As weights are adjusted in the neural network some of Knight moves become "more desirable" than others. Eventually a series of Knight moves that complete a tour is created [5, 6].

The batch move technique does not solve the tour from start to finish but does the entire pathway at once. Random parts of the path are solved first and then the moves required to link all the paths are solved as the network goes through each iteration. The batch move technique is done as an applet which will automatically begin when the webpage loads. The source code for this applet is also available at the website. This applet can only solve tours on an 8x8 sized chessboard [7].

The batch path technique consists of stringing together a series of Knight moves into small paths of several Knight moves each. The paths are connected together in a similar method to the batch move technique. A neural network uses the paths as inputs and checks to see if they connect in a continuous path to make a tour. If they do not make a continuous path then the tour is incomplete. Random Knight

0-7803-9490-9/06/$20.00/©2006 IEEE

moves are inserted to connect the paths at each iteration and weights are adjusted based on how closely the new path is to being a complete Knight's tour. Other small paths of more than one Knight move may also be inserted to attempt completion. Eventually a series of paths and Knight moves is found that will complete a tour. In the batch path technique, every time the applet is right clicked a few of the moves will change but most of the path remains intact. The end result is a new Knight's tour with every right click.

The batch move technique doesn't solve a Knight's tour from first move to last move but merely connects predetermined paths. Since the major paths are preexisting the majority number of paths which can make Knight's tours is never solved. Only those tours which include the preexisting paths can be solved.

The pure mathematical methods for the Knight's tour using neural networks are based on modeling all aspects of the system as mathematical problems. For example, in the many papers written by Ian Parberry, the neural network is simulated using math to represent the various parts of the neurons. Math is also used for the transfer functions and training functions of the networks. Also, the Knight's ability to move and the tours themselves are presented in mathematical terms.

The biggest advantage of using this method is that it can be done in so many ways. Gordon and Slocum used backtracking similar to the 18th century and early 19th century solutions were used to improve on Warnsdorff's rule. They combined it new evolution algorithms. Many others, such as Chellapilla and Fogel, have used evolutionary algorithms as well. Lately the focus has been on using genetic algorithm models [4, 8].

In this work multilayered architectures and algorithms are used to solve the Knight's tours. In addition the performance of Levenberg-Marquardt; variable learning (adaptive) gradient descent; resilient backpropagation; the Fletcher-Reeves conjugate gradient; and the Broyden-Fletcher-Goldfarb-Shanno Quasi-Newton algorithm, training algorithms were compared. All of the networks in this work are modeled using MATLAB neural network toolbox.

III. DEFINITION OF A KNIGHT'S TOUR

A Knight is a chess piece capable of moving in one of eight possible directions as shown in Figure 1. Throughout this paper each of the eight distinct Knight moves will always be referred to using the numbers one through eight as shown in figure 1.

A closed Knight's Tour is a type of Hamilton Cycle (also known as a Hamiltonian circuit). A Knight visits each square on a chessboard (an 8x8 square grid) once and only once until it has visited each of the 64 squares. Since a single Knight move would take the piece from the last square back to the first square, this is a closed tour.

An opened Knight's tour while not a completely closed Hamilton cycle is known as a Hamiltonian path. The only difference between an open tour and a closed tour is that open tours do not require getting back the original starting square.

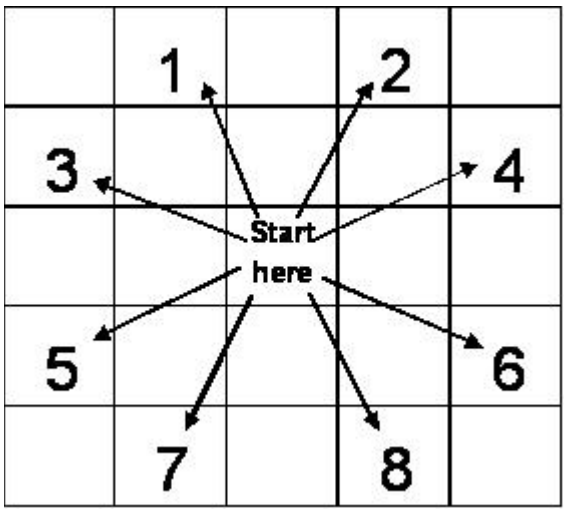

Figure 1: The Eight Knight Moves.

This paper will focus on Knight's tours on a 5x5 sized board. There are a total of 1728 distinct tours on a 5x5 board [6]. This is still to large a training set to be conveniently used with MATLAB. Therefore the set of Knight's tours will be limited to those tours which can be made starting only from the center square of a 5x5 sized board. There are only 64 distinct Knight's tours in this training set.

Standard algebraic chess notation will be replaced by a Cartesian coordinate system. Point (0, 0) is the lower left corner and (5, 5) is the upper right corner. As a reference example the 24 moves for the first tour are {1, 6, 8, 5, 3, 2, 4, 8, 7, 3, 1, 4, 6, 7, 5, 1, 4, 4, 7, 8, 3, 5, 2, 1} and the coordinates are {(3, 3), (2, 5), (4, 4), (5, 2), (3, 1), (1, 2), (2, 4), (4, 5), (5, 3), (4, 1), (2, 2),(1, 4), (3, 5), (5, 4), (4, 2), (2, 1), (1, 3), (3, 4), (5, 5), (4, 3), (5, 1), (3, 2), (1, 1), (2, 3), (1, 5)}.

Many of the Knight's paths lead to dead ends before they can complete a tour. These paths stop at the first move which forces error. For example, a path following the Knight's moves {1, 6, 8, 5, 3, 2, 4, 8, 7, 3, 1, 4, 6, 7, 3} can never complete a Knight's tour. There are several different paths that can be followed after the last move but all of them leave the knight trapped with no more moves to make while leaving some squares unvisited.

To represent these dead end paths in MATLAB he rest of the moves are filled in with negative tens thusly: {1, 6, 8, 5, 3, 2, 4, 8, 7, 3, 1, 4, 6, 7, 3, -10, -10, -10, -10, -10, -10, -10, -10, -10}. This makes it easier for the networks to distinguish failed paths from completed tours.

IV. PROPOSED TECHNIQUES

Four main approaches are examined: function approximation; classification; and two custom networks: SquareNet and MassiveFA. For the purposes of this paper the concentration will be on the classification approach.

IV.1 ARCHITECTURE AND METHODOLOGY

The Knight's Tours Artificial Neural Network (KTANN) program uses the data set of 288 Knight's tours. 270 tours are used for training and 18 tours are used for testing.

Knight's Tours Artificial Neural Network (KTANN) uses 24 inputs. One input for each of the 24 Knight moves required to make the 5*x*5 tour. There are 40 neurons in the hidden layer. The number of neurons used for the hidden layer was decided by trial and error. KTANN has two outputs; one represented the class of pathways that complete Knight's tours and another for the class of pathways that do not complete the tours. The architecture of the network is shown in Figure 2.

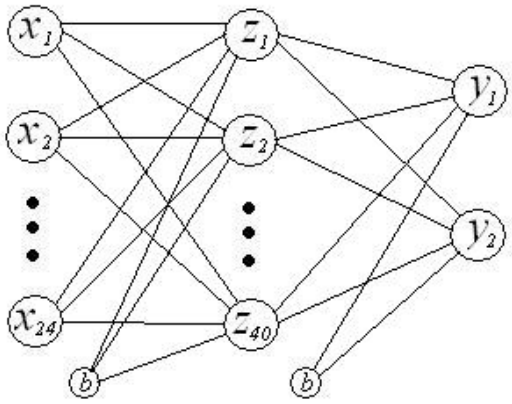

Figure 2: Knight's Tours Artificial Neural Network.

As each training algorithm is used the networks parameters (learning rate, momentum, etc.) are adjusted for quickest convergence. Only the GDX algorithm uses the learning rate increment parameter. Only the CGF and BFG algorithms require alterations to step size. Rapid convergence is desired since it will take some of algorithms several minutes to converge.

Changes made to the program are not just the training algorithm and the goal. Parameters such as learning rate, learning rate increment and momentum are also changed using trial and error to obtain the quickest convergence for each algorithm [9, 10, 11, 12, 13]. No post processing is done and all initial weights are set by the default initialization program. After each algorithm has run 20 times with a goal of 1E-1 the goal is decreased to 1E-2, 1E-3, etc. until 1E-5 is achieved.

IV.2 TRAINING AND TESTING

Training the KTANN program is done using most of the data set, 270 out of the 288 paths. Four out every eighteen paths is a tour. The remaining 18 paths include 4 complete tour pathways and 14 incomplete tour pathways. These 18 paths are used for testing the network after it is trained.

In addition to using part of the data set to test the network a special test is done using several phony paths. The purpose of the phony paths is to test how the network will respond to noisy data and illegitimate data.

Noisy data is represented in two ways. In one case, legitimate pathways used with just one of the Knight moves replaced. In the other case legitimate pathways used with just one of the Knight moves missing.

Illegitimate data is represented two ways as well. First, it is represented by using reversed order of moves. Then it is represented by using some random order of moves.

The training and testing phases are built into the program and they happen in this order. First, KTANN loads all the training and testing files. Secondly, the program creates the network. Next, the program runs the network using the training data (270 out of 288 pathways) to adjust the weights of the network. Then the first test is run using the remaining 18 of 288 pathways. Finally, the last test is run using variations of noisy and illicit data.

V. RESULTS AND DISCUSSIONS

The result of various training algorithms with and the number of epochs for each algorithm is shown in Table 1. The stopping condition is the mean squared error. First the goal is set to 1E-1 and each training algorithm is run through the program. There are 20 runs for each algorithm and the average number of epochs needed to converge is calculated. If a run of the program fails to converge another run is done. Only runs that successfully converge are used to calculate the epochs and the time. However, due to the optimization of the algorithms there were no failures to converge for any of the five algorithms while running the KTANN program.

Table 1: Epoch Results from the KTANN Program.

Training Algorithms	1E-01	1E-02	1E-03	1E-04	1E-05
Levenberg-Marquardt (LM)	2	3	4	7	9
Adaptive Gradient Descent (GDX)	58	112	1858	15413	199039
Resilient Backpropagation (RP)	15	32	133	496	2736
Conjugate Gradient (CGF)	9	33	118	374	946
Quasi-Newton (BFG)	10	20	31	54	95

Table 2 compares the algorithms running the KTANN program with regards to convergence time. This time BFG is one of the slowest algorithms. GDX starts as one of the fastest at high goal values of 1E-1 and 1E-2 but it becomes the slowest as the goal is decreased. Indeed, on some runs GDX required over 2½ hours to converge at goal values of 1E-5. LM has the most consistent results and proves to be the fastest algorithm at goal values of 1E-5. Overall, RP and CGF prove to be the quickest at converging. They perform better than the other algorithms for most of the goal values.

Table 2: Time Results from the KTANN Program.

Training Algorithms	1E-01	1E-02	1E-03	1E-04	1E-05
Levenberg-Marquardt (LM)	6.516	9.206	12.277	19.288	23.652
Adaptive Gradient Descent (GDX)	1.724	2.716	37.364	299.938	5402.900
Resilient Backpropagation (RP)	1.009	1.312	3.441	11.027	58.394
Conjugate Gradient (CGF)	1.024	1.878	5.100	16.742	47.705
Quasi-Newton (BFG)	12.852	24.209	39.186	63.155	111.388

Each algorithm had some optimization done with regards to raising and lowering parameter values such as momentum and learning rate. The maximum epochs are changed from 1000 to 1,000,000. Thus there is no failure to converge due to an algorithm requiring additional iterations to converge.

The LM, and RP algorithms have narrow differences between the minimum and maximum time needed to converge. The CGF and BFG algorithms have maximum convergence times four times greater than the minimum convergence time but only a couple of the runs are close to the maximum. The GDX algorithm has a maximum convergence time which is four times greater than the minimum convergence times. The GDX algorithm has the largest spread with some runs being half the average time while others are twice the average time.

Looking at an algorithms convergence curve gives an excellent indication of how good it is and what type of performance it might have at other goal values. Figure 3 shows that the convergence curve for the BFG is somewhat linear in shape which suggests that it will perform better at very low goal values. The GDX convergence curve is extremely steep at first which shows excellent learning but it quickly flattens out to an almost horizontal line as shown in Figure 4. Most of the algorithms look more like the GDX curve than the BFG curve [14].

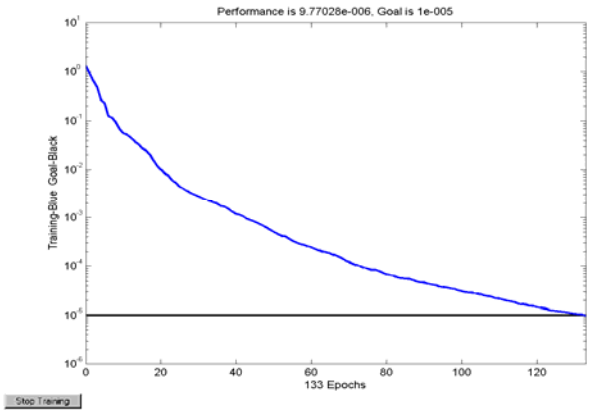

Figure 3: BFG Convergence Curve.

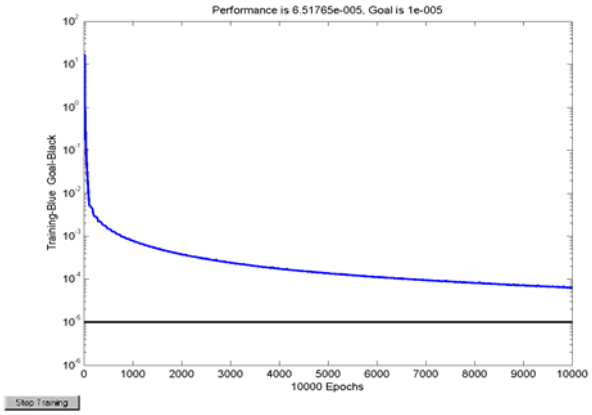

Figure 4: GDX Convergence Curve.

VI. CONCLUSIONS

The performance of the proposed training algorithms indicates that they all converged but with different rates of convergence. Some of the algorithms can converge with fewer iterations. KTANN is such a complex program (solving hundreds of pathways at once) that convergence in the fewest epochs means less when compared to. Some converged with more epochs but with less time than the others. The main measure of the quality of performance of an algorithm is the amount of real time it requires to converge. In this case CGF is the quickest and therefore the best algorithm for this problem.

References

[1] Brain, M. "How Chess Computers Work." http://stuffo.howstuffworks.com/chess1.htm.

[2] Greengard, M. "The Garry Kasparov ChessBase Interview, Part 1." http://www.chessbase.com/newsdetail.asp?newsid=2309.

[3] Sloan, S. "Kasparov Wants Deep Blue Computer Print-Outs Sealed."

[4] Chellapilla, K.; Fogel, D. B. "Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge." IEEE Transactions on Neural Networks, Vol. 10, No. 6, p1382-1391, (1999).

[5] Banerji, R. "Game Playing." Encyclopedia of Artificial Intelligence, Vol. 1, New York: Wiley & Sons, p312-319, (1987).

[6] Kraitchik, M. "The Problem of the Knights." Mathematical Recreations, New York: W. W. Norton, p. 257-266, (1942).

[7] http://www.neuro.sfc.keio.ac.jp/~saito/java/knight.html.

[8] Parberry, I. "An Efficient Algorithm for the Knight's Tour Problem," Discrete Applied Mathematics, Vol. 73, p. 251-260, (1997).

[9] Bishop, C. M. Neural Networks for Pattern Recognition, Oxford University Press, (1995).

[10] Fausett, L. V. Fundamentals of Neural Networks, Prentice Hall, (1994).

[11] Gurney, K. An Introduction to Neural Networks, London: Routledge, (1997).

[12] Haykin, S. Neural Networks: A Comprehensive Foundation, 2nd Ed., Prentice Hall, (1998).

[13] Demuth, H.; Beale, M. Neural Network Toolbox for Use with MATLAB: User's Guide Version 4, The MathWorks Inc. (2001). http://www.ishipress.com/printout.htm.

[14] Escalante, Robert G.; "Comparison of Artificial Neural Network Architectures and Training Algorithms for Solving the Knight's Tours" p. 65-70, (2005).