Complex Problem Solving With Neural Networks: Learning Chess

Mr. Jack Sigan
Dr. Aleksander Malinowski, Advisor
Dept. of Electrical and Computer Engineering

BRADLEY UNIVERSITY

May 3, 2005
Outline

• Introduction to project and neural networks
• Neural networks and chess
• Neural network system design
• Tools and procedures
  – Preprocessing
  – Training
  – Interface
• Results and conclusions
  – Initial results
  – Creation of the evaluation function
  – Final conclusions
System Block Diagram

Playing or Advisory Mode
Player Move

Interface (Generates Board Description)

Board Position
Chosen Moves

ANN Move(s)

ANN “System”

1792 Feedforward networks
2 layers of 128 nodes

Learning Mode

Game DataBase
Data Preprocessing
ANN “System”
Neural Networks

Node structure with hyperbolic tangent activation function

Simple partially connected neural network structure from Stuttgart Neural Network Simulator (SNNS)
Chess and Neural Networks?

- Demonstrates complex decision making
- Highly nonlinear problem
- Schemas
- Widely studied
- Massive amounts of available data
- Success with checkers
- Mixed results with chess in the past
Standards and research

• Numerous applicable “standards”
  – Chess “laws”
    • FIDE (Fédération Internationale des Échecs)
    • http://www.fide.com
  – PGN file standard
    • rec.games.chess, 1994
  – EPD file standard
    • Format supplied by “EPD_Position.exe” (standard?)
  – Chess engine standard command reference

• Most influential research
  – K. Chellapilla and D.B. Fogel
  – C. Posthoff, S. Schawelski and M. Schlosser
Parallel Network Designs

• Decrease learning cycle time
• Less destructive learning process
• Multiple “suggested” moves may be returned
• Simpler network architectures
• 2 paradigms for deconstructing chess
  – Geographical “move based”
  – Functional “piece based”
• External logic is applied to both paradigms to filter out illegal moves or impossible moves
Parallel Network Designs

Mapping the legal moves in chess, using overlay consisting of queen + knight moves

Excluding castling, there are 1792 moves possible —ignoring the piece type which is moved

Example: Moving from d3 to d4 is considered ONE possible move, whether the piece is a pawn, queen, king, etc.
Parallel Network Designs

- starting position \(i\)
- \(m_{if} = f(b_i)\) represents all legal moves for a board position \(b_i\)
- game \(g\) of \(n\) moves may be expressed as a set of board positions \(b_i, b_i \in g\), where \(i\) is the position number 0 to \(n\).

\[
L = \sum_{i=0}^{n} f(b_i)
\]

Legal moves for a game of \(n\) positions

\[
M = \sum_{i=0}^{\infty} (L_i)
\]

Legal moves for chess (all games)

- In this design, it is required to create an individual ANN structure for all moves \(t, t \in M\). \(M = 1792\), ignoring castling
Approach A: Geographical

Design with “move specific” neural networks—the geographical approach
Parallel Network Designs

- starting position $i$
- piece $p$
- $m_{if} = f(p, b_i)$ represents all legal moves for a piece in $b_i$
- game $g$ of $n$ moves may be expressed as a set of board positions $b_i$, $b_i \in g$, where $i$ is the move number 0 to $n$.

\[
L = \sum_{i=0}^{n} f(p, b_i) \quad \text{Legal moves for a game of } n \text{ moves}
\]

\[
M = \sum_{i=0}^{\infty} (L_i) \quad \text{Legal moves for a piece (all games)}
\]

- In this design, it is required to create an individual ANN structure for all pieces $p$, $p = 1$ to 16
Approach B: Functional

Design with “piece specific” neural networks—the functional approach
Final ANN System

Current Board Position

ANN Position A8-A7

ANN Position A8-B8

ANN Position H2-H1

Evaluation Function and Rule Logic

Decision Finalization
(Pick The Strongest Output From The Output Set)

"1" Will Close "Switch"
"0" Disables Move

Output (Move)

Design with “move specific” neural networks—the geographical approach
Data Representations

1. e4 d6      2. d4 Nf6  3. Nc3 g6    4. Nf3 Bg7    5. Be2 O-O    6. O-O Bg4
13. f3 Bd7    14. f4Bg4  15. Rb1 c6   16. fxe5 dxe5 17. Bc5 cxd5  18. Qg5 dxe4
25. Qxf5 Qf2  26. Rd1e3 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

Example of the PGN (algebraic) standard

rnbqkbnr/pppppppp/8/8/8/8/8/8 w KQkq - pm d4;
rnbqkbnr/pppppppp/8/8/3P4/8/PPP1PPP/8 RNBQKB1R b KQkq d3 pm Nf6;
rnbqkb1r/pppppppp/5n2/8/3P4/8/PPP1PPP/8 RNBQKB1R w KQkq - pm Nf3;
rnbqkb1r/pppppppp/5n2/8/3P4/5N2/PPP1PPP/RNBQKB1R b KQkq - pm b6;

Example of the EPD (string) format
## Input Vector Creation

<table>
<thead>
<tr>
<th>Piece</th>
<th>EPD Character</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>k,K</td>
<td>1.0, -1.0</td>
</tr>
<tr>
<td>Queen</td>
<td>q,Q</td>
<td>0.9, -0.9</td>
</tr>
<tr>
<td>Rook</td>
<td>r,R</td>
<td>0.5, -0.5</td>
</tr>
<tr>
<td>Knight</td>
<td>n,N</td>
<td>0.4, -0.4</td>
</tr>
<tr>
<td>Bishop</td>
<td>b,B</td>
<td>0.3, -0.3</td>
</tr>
<tr>
<td>Pawn</td>
<td>p,P</td>
<td>0.1, -0.1</td>
</tr>
</tbody>
</table>

“Standard” values for pieces used in the floating point input vector creation

```plaintext
# 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
0.0 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
```

Floating point input vector for the initial board position
The game data preprocessing procedure

• Board positional data, plus a ‘yes’ or ‘no’ decision in regards to making the specific move must be in the floating point vector.

• A mix of ‘yes’ and ‘no’ samples will be used in all training sets

• Training sets are randomly chosen

• Do not differentiate between pieces to be moved, only the initial and final positions are important—rule logic is separate
Training Setup

GDANSK SERVER

PC1

PC2

PCn

Training data

Network file
Training Process

Start

1. Obtain network name from configuration file
2. Obtain required raw data files (download)
3. Initialize a new network
4. Generate training data set from raw data
5. Start SNNS, load network and data
6. Train 50 epochs
7. Send trained network to server (FTP)
8. More networks to train?

Setup (Shell Program)

- Check for updates and process them
- Generate scripts and batch files—Start batch execution
- Obtain network name from configuration file

Data Processing

- Obtain required raw data files (download)
- Generate training data set from raw data
- Initialize a new network

Training

- Start control script
- Start SNNS, load network and data
- Train 50 epochs

Exit Button (Kill all processes)

Yes
Training Problems

- 50 epochs of about 5 thousand patterns
- 1792 networks with 18 minutes per network
- SNNS is NOT the most elegant solution!
  - Java version lacks scripting ability
  - Slow training
  - Variable execution time
- Problems with .NET / lab machines
Interface Module
Initial Findings and Results

- Backprop does not work
- Using resilient backprop instead
- No significant difference between 2 or more hidden layers (in training speed)
- Close output proximity—how to decide?
- An evaluation function is required
Evaluation Function
Evaluation Function 1

• Only top NN outputs will be considered
• Rate the moves, ignore NN output
• Consider
  – Material (?M)
  – Threats (?T)
  – Mobility (?O)
  – Vulnerabilities (?V)
• Score = a*?M + b*?T + c*?O + d*?V
• a,b,c,d are weights (experimentally determined)
Evaluation Function 2

- Only top NN outputs will be considered
- Rate the moves with NN output as a factor
- Consider
  - Material (M)
  - Threats (T)
  - Mobility (O)
  - Vulnerabilities (V)
  - NN output (Y)
- Score = a*M + b*T + c*O + d*V + Y
- a, b, c, d are weights (experimentally determined)
Final Evaluation Function

• A chess engine is utilized
  – Provides an experimental “baseline”
  – Provides a board evaluation “E” only
• White side chooses move based on “E”
• Black uses $E + aY$
• $Y$ is the neural network output
• The “a” coefficient is found experimentally
Findings and Results

- Neural network output can contribute to the evaluation of a move
- Performance of the NN system is better than the chess engine alone
- The system performs very poorly in end-game scenarios, possibly indicating a need for piece-specific knowledge
- Modified training may lead to improvements in performance…
Findings and Results

Average Score VS Ply
(Not Corrected for NN Term Weights)
Findings and Results

Average Gaviota Engine Score VS Ply
(Corrected for NN Term Weights)
Training Vector Coding

• May need to evaluate other formats
  – Binary representation (used in end-game research)
  – Multiple spatial relationships?

• Training must be started to know!

Possible spatial relationships
Questions?

Additional questions and comments are invited. Please contact:
Jack Sigan, jsigan@bradley.edu

Project Website:
http://cegt201.bradley.edu/projects/proj2005/nnchess/