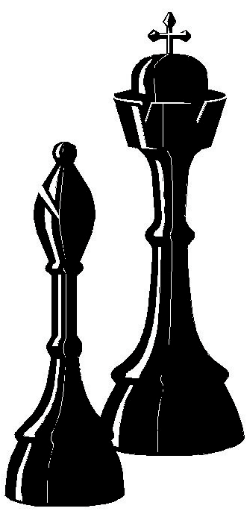
Complex Problem Solving With Neural Networks: Learning Chess

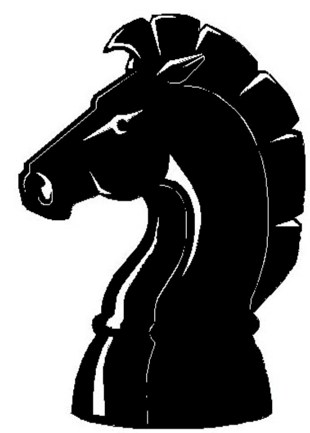
Mr. Jack Sigan

Dr. Aleksander Malinowski, Advisor

Dept. of Electrical and Computer Engineering

BRADLEY UNIVERSITY



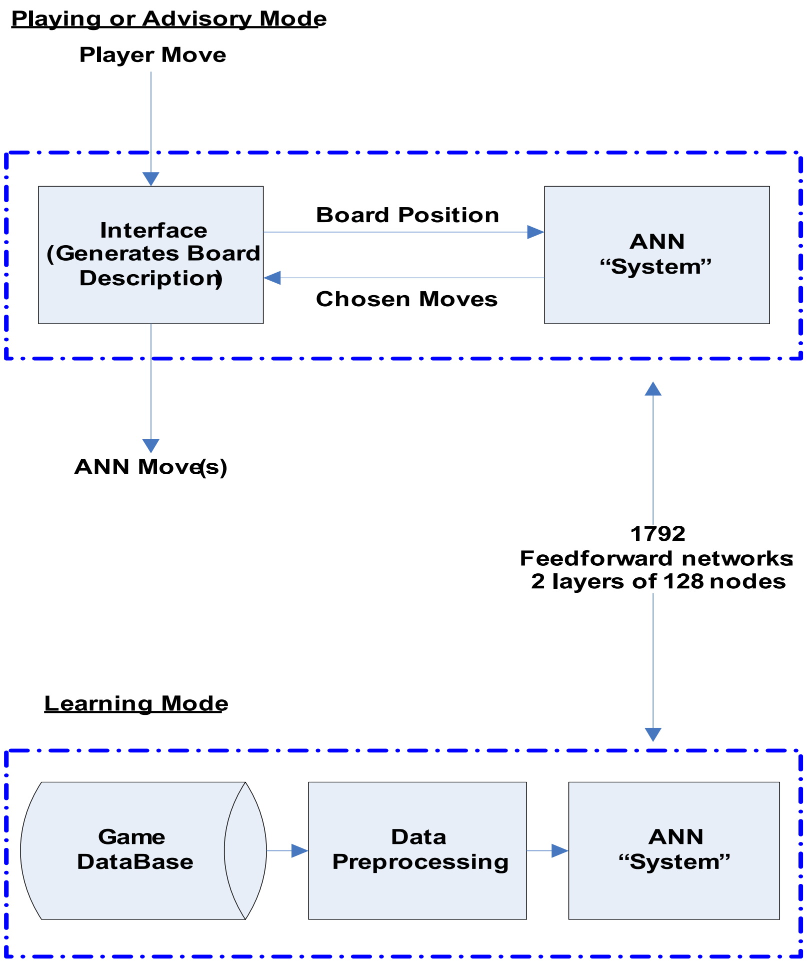


May 3, 2005

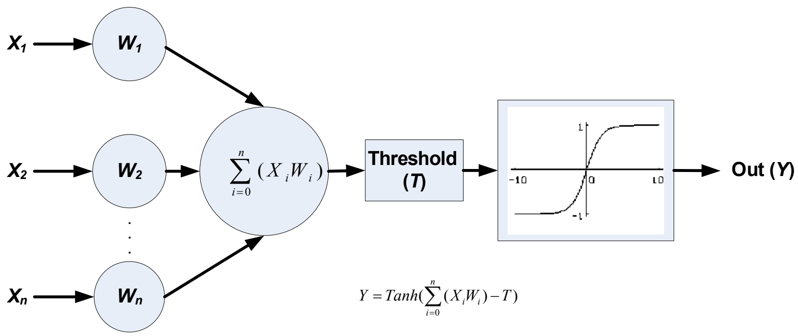
Outline

- 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

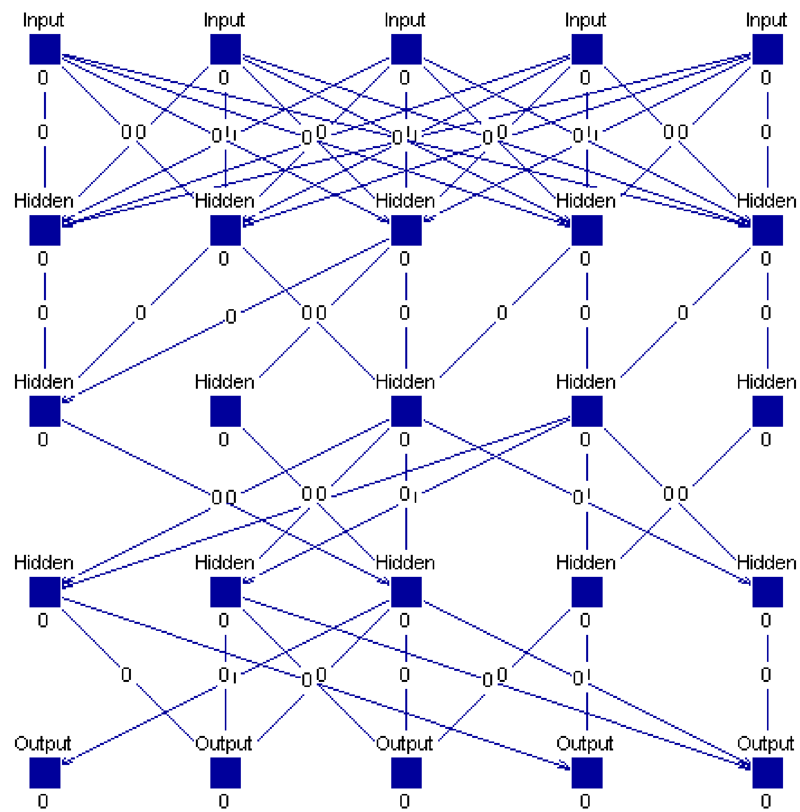
System Block Diagram



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

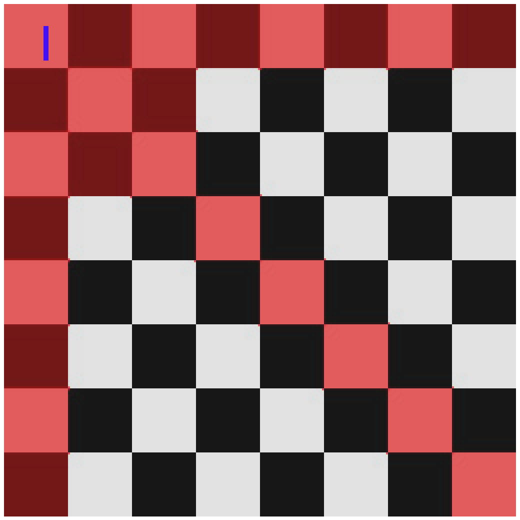
- Numerous applicable “standards”
  - Chess “laws”
    - FIDE (*Fédération Internationale des Échecs*)
    - http://www.fide.com
  - PGN file standard
    - rec.games.chess, 1994
  - EPD file standard
    - Format supplied by “EPD_Position.exe” (standard?)
  - Chess engine standard command reference

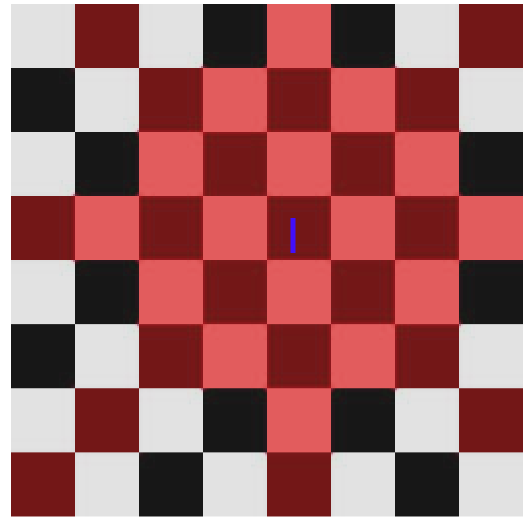
- Most influential research
  - K. Chellapilla and D.B. Fogel
  - C. Posthoff, S. Schawelski and M. Schlosser

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

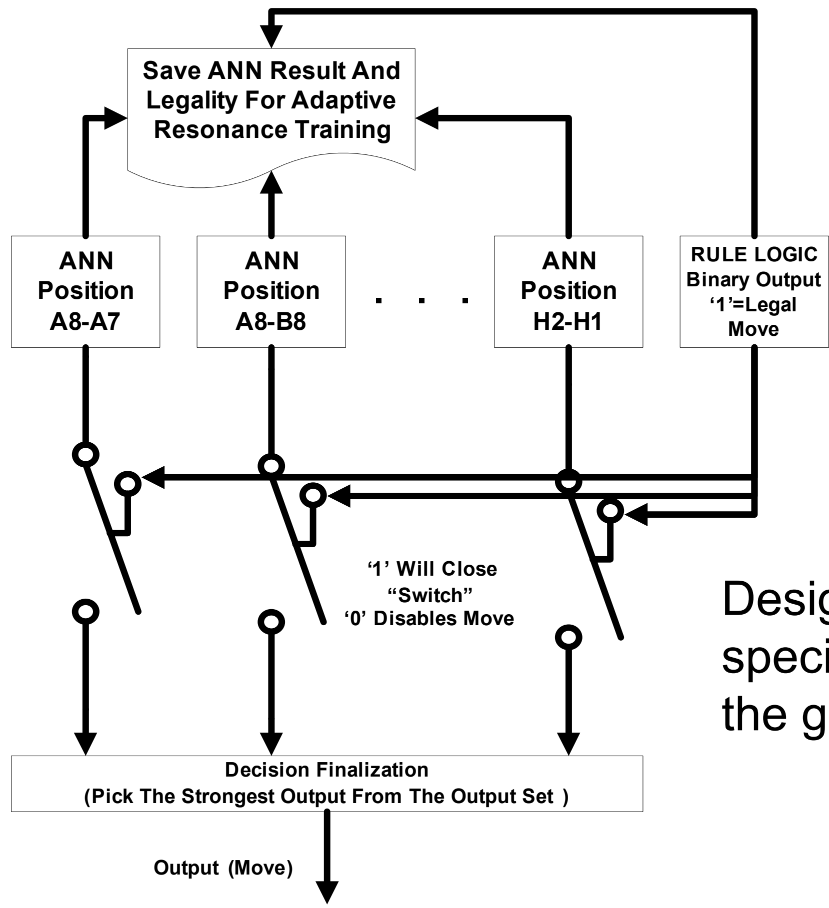
- starting position *i*
- $m_{if}=f(b_i)$ represents all legal moves for a board position b_i
- game *g* of *n* moves may be expressed as a set of board positions *bi, biЄg,* where *i* is the position number 0 to *n*.

$$L = \sum_{i=0}^{n} f(b_i)$$ 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*, *tЄM. M*=1792, ignoring castling

Approach A: Geographical



Design with "move specific" neural networks— the geographical approach

Parallel Network Designs

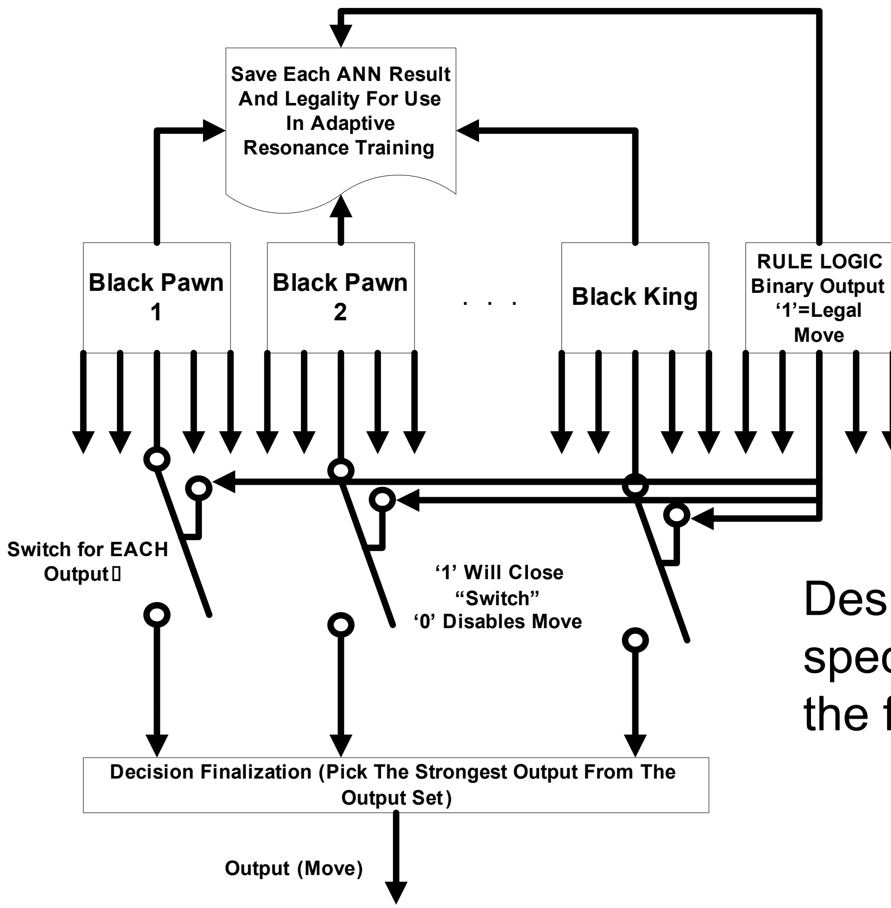
- starting position *i*
- piece *p*
- $m_{if}=f(p,b_i)$ represents all legal moves for a piece in b_i
- game *g* of *n* moves may be expressed as a set of board positions *bi, biЄg,* where *i* is the move number 0 to *n*.

$$L = \sum_{i=0}^{n} f(p, b_i)$$ Legal moves for a game of *n* moves

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

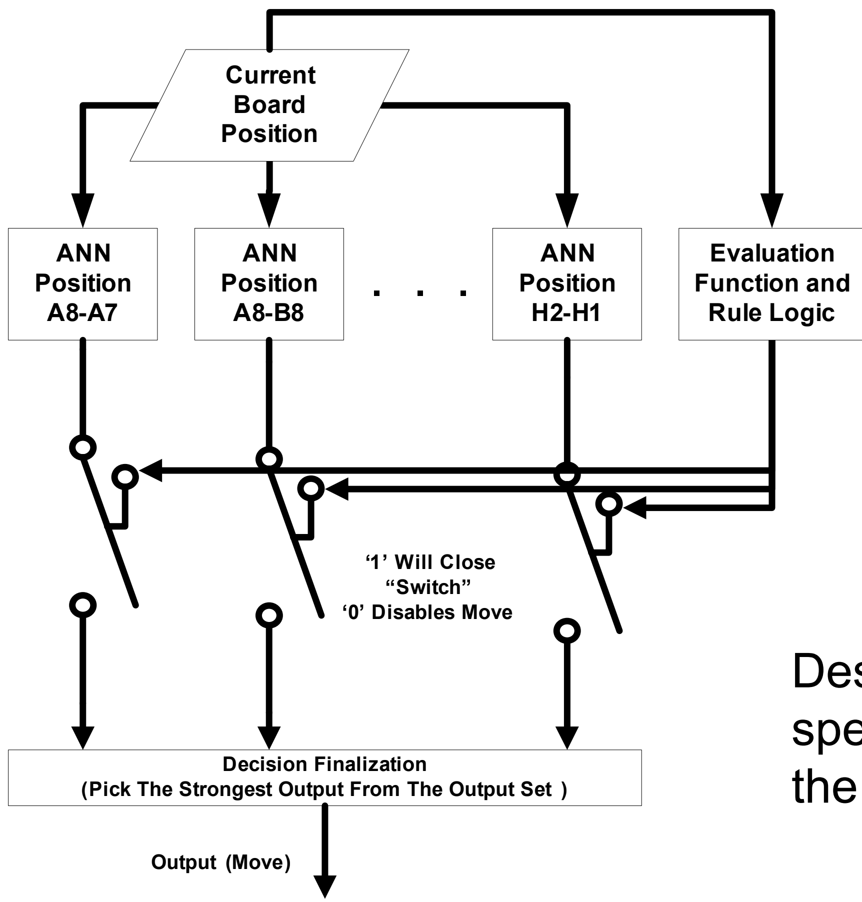
- In this design, it is required to create an individual ANN structure for all pieces *p*, *p*=1 to 16

Approach B: Functional



Design with "piece specific" neural networks—the functional approach

Final ANN System



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
 7. Be3 Nc6     8. Qd2 e5  9. d5 Ne7    10. Rad1 Bd7  11. Ne1 Ng4   12. Bxg4 Bxg4
13. f3 Bd7     14. f4Bg4  15. Rb1 c6    16. fxe5 dxe5 17. Bc5 cxd5  18. Qg5 dxe4
19. Bxe7 Qd4+ 20. Kh1f5   21. Bxf8 Rxf8 22. h3 Bf6    23. Qh6 Bh5   24. Rxf5 gxf5
25. Qxh5 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/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;
```

Example of the EPD (string) format

Input Vector Creation

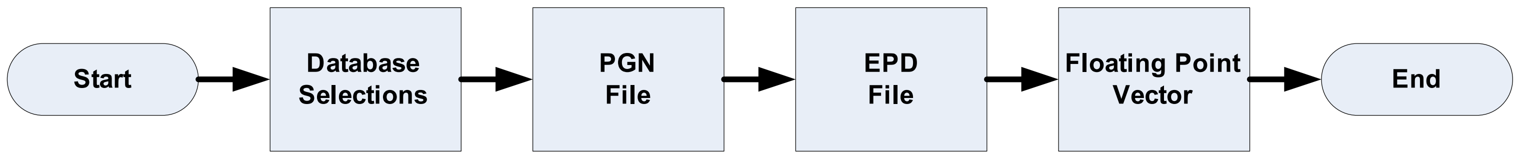
Piece	**EPD Character** **Black, white**	**Weight** **Black, white**
King	k,K	1.0, -1.0
Queen	q,Q	0.9, -0.9
Rook	r,R	0.5, -0.5
Knight	n,N	0.4, -0.4
Bishop	b.B	0.3, -0.3
Pawn	p,P	0.1, -0.1

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

```
# 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
```

Floating point input vector for the initial board position

Preprocessing



The game data preprocessing procedure

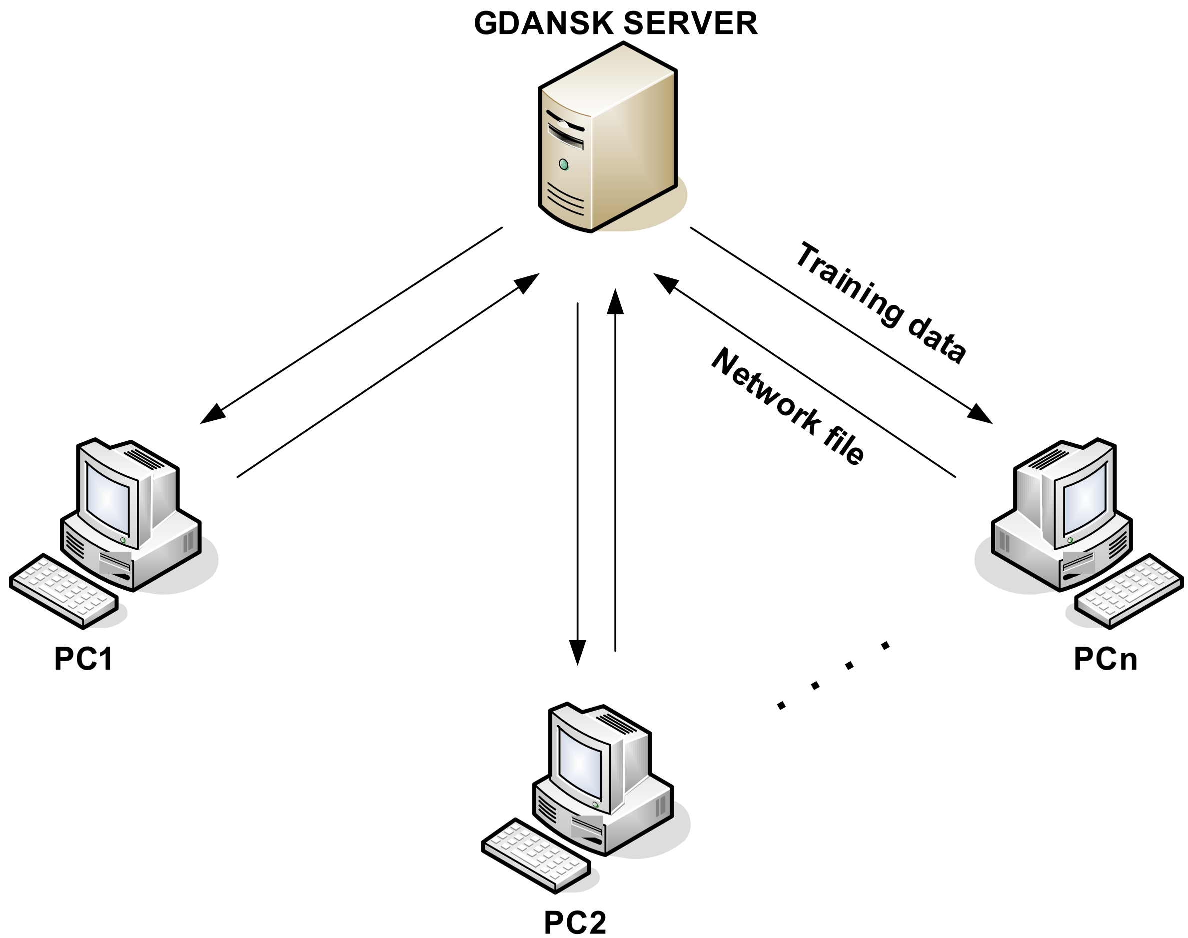
•Board positional data, plus a ‘yes’ or ‘no’ decision in regards to making the specific move must be in the floating point vector.

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

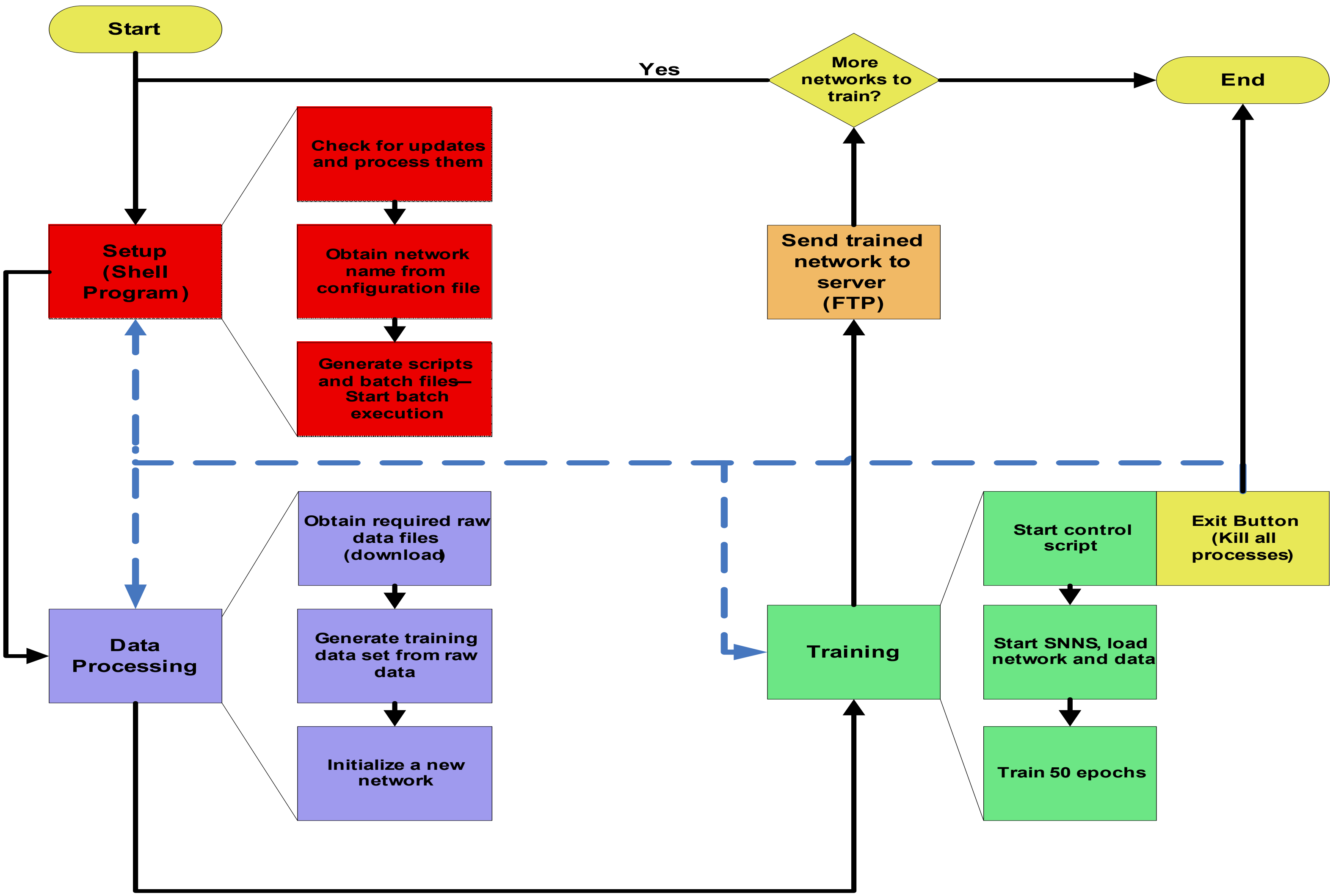
•Training sets are randomly chosen

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

Training Setup



Training Process



Training Problems

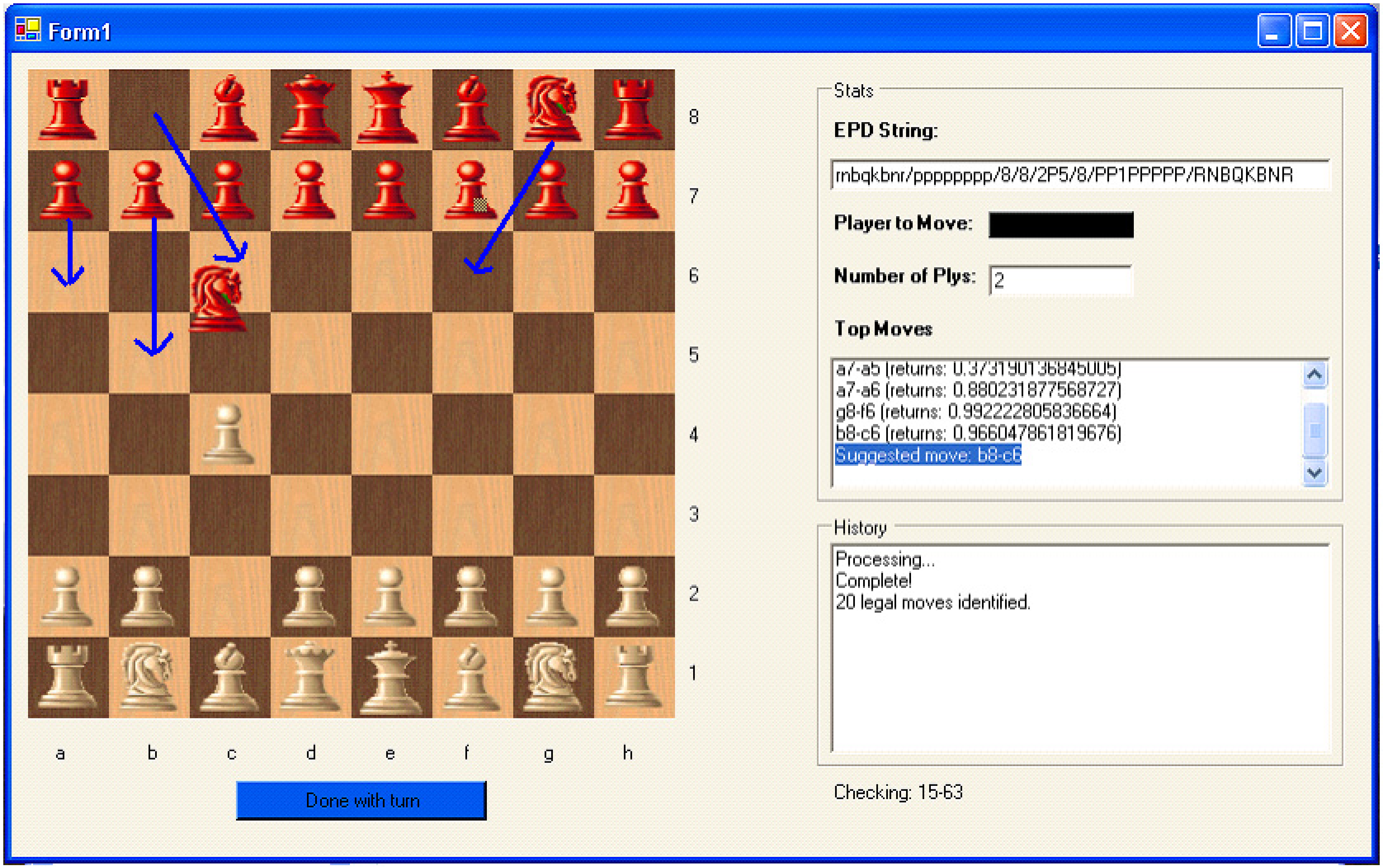
- 50 epochs of about 5 thousand patterns
- 1792 networks with 18 minutes per network
- SNNS is NOT the most elegant solution!
  - Java version lacks scripting ability
  - Slow training
  - Variable execution time
- Problems with .NET / lab machines

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)

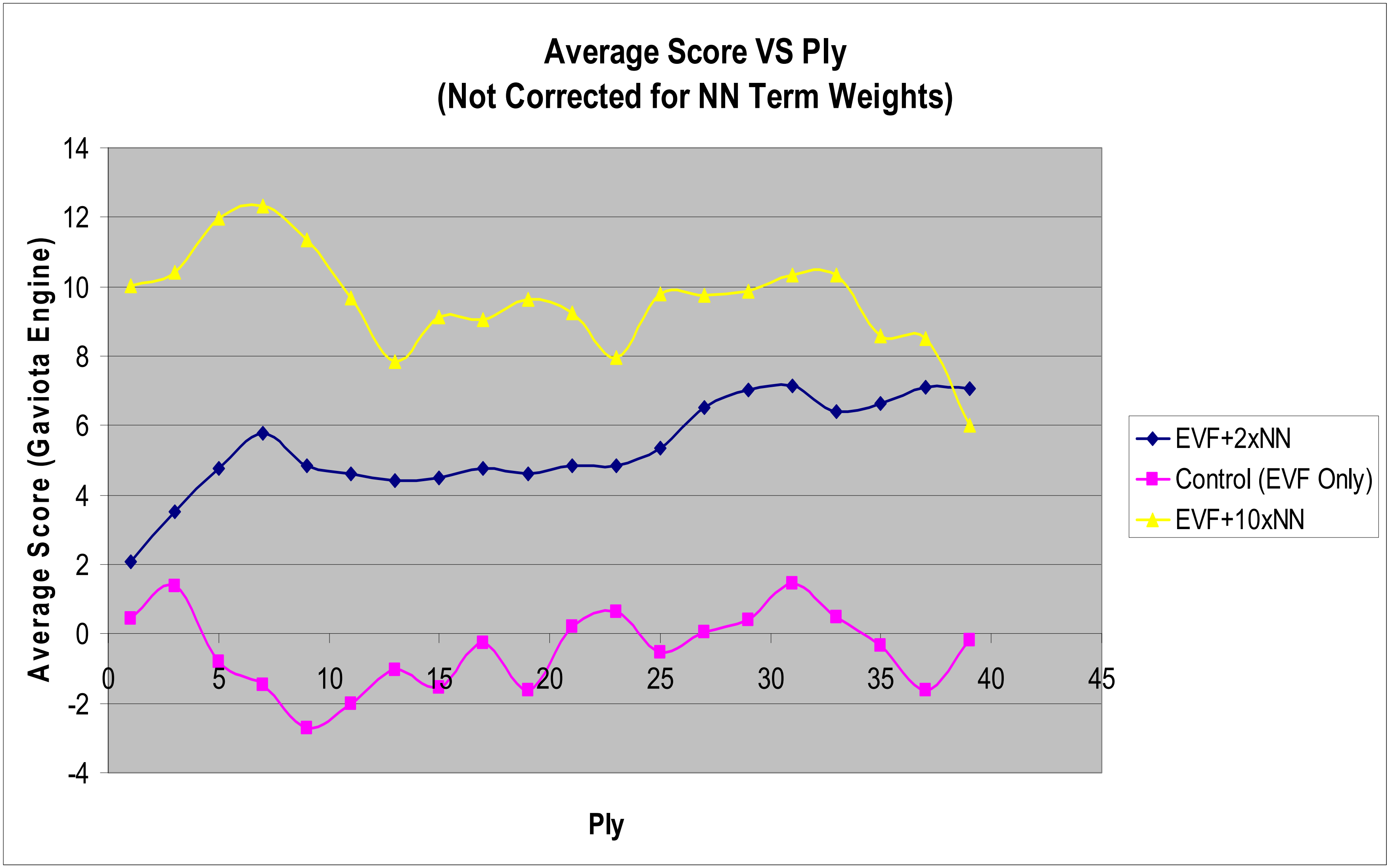
Final Evaluation Function

- 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+a*Y
- Y is the neural network output
- The “a” coefficient is found experimentally

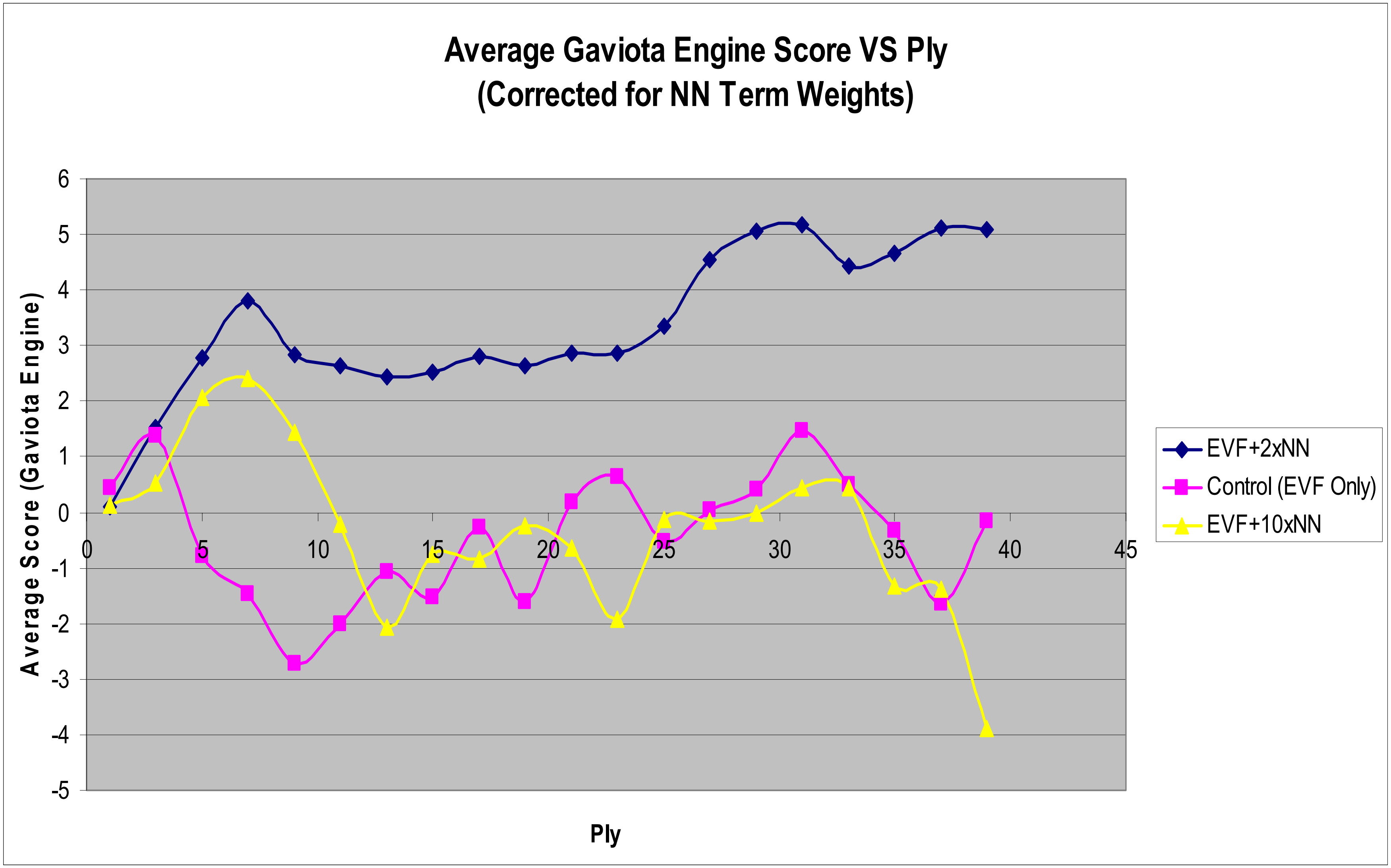
Findings and Results

- 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

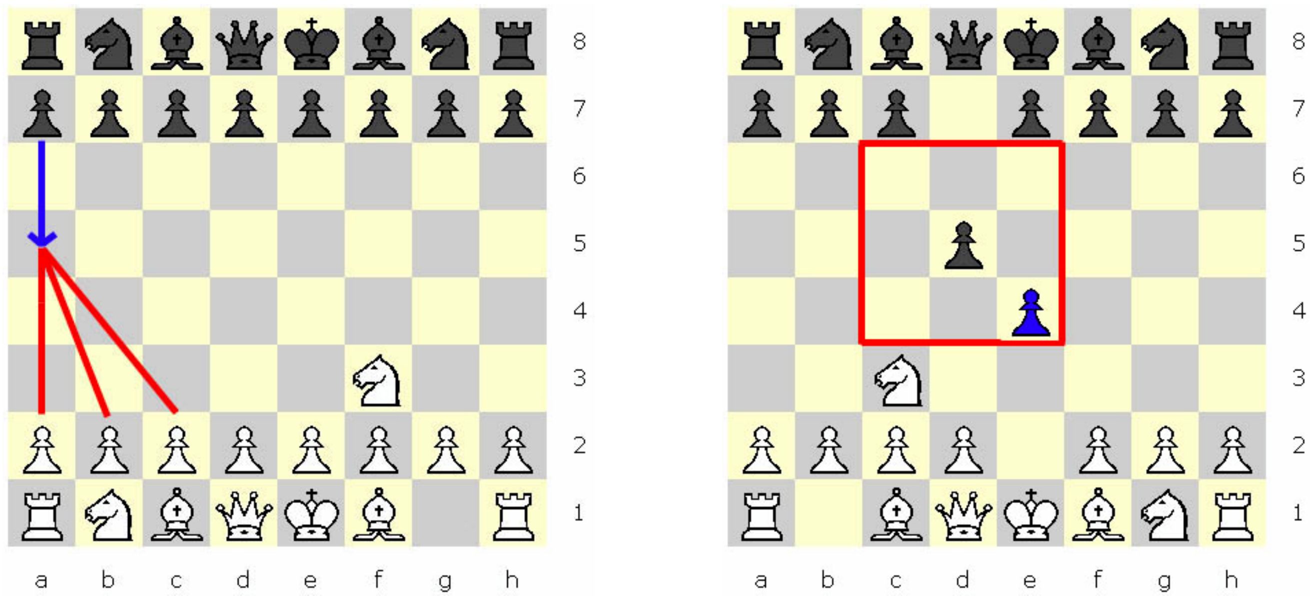


Findings and Results



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/