University of Reading

Chess endgame news: an Endgame challenge for neural nets

Article

Accepted Version

Haworth, G. ORCID: https://orcid.org/0000-0001-9896-1448
(2019) Chess endgame news: an Endgame challenge for neural nets. ICGA Journal, 41 (3). p. 176. ISSN 1389-6911 doi: https://doi.org/10.3233/ICG-190109 Available at https://centaur.reading.ac.uk/86555/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.
Published version at: https://content.iospress.com/articles/icga-journal/icg190109
To link to this article DOI: http://dx.doi.org/10.3233/ICG-190109

Publisher: The International Computer Games Association

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

CentAUR

Central Archive at the University of Reading

Reading's research outputs online

CHESS ENDGAMES AND NEURAL NETWORKS

G.M[c]C. Haworth[1] and M. Velliste[2]

Berkshire, England

ABSTRACT

The existence of endgame databases challenges us to extract higher-grade information and knowledge from their basic data content. Chess players, for example, would like simple and usable endgame theories if such holy grail exists: endgame experts would like to provide such insights and be inspired by computers to do so. Here, we investigate the use of artificial neural networks (NNs) to mine these databases and we report on a first use of NNs on KPK. The results encourage us to suggest further work on chess applications of neural networks and other data-mining techniques.

1. INTRODUCTION

Endgame databases, particularly those of Ken Thompson (1986), have made a significant contribution to chess. Their existence and commercial availability invite the chess community to:

- re-analyse exemplar positions and games played over the board;
- validate, or 'cook' and possibly repair, published problems and studies;
- adjust past perceptions of and correct publications on various endgames, such as KQKR;
- identify new concepts, patterns and specific positions, such as zugzwangs and studies;
- create complete theories of particular endgames, such as KRKN and KBBKN;
- prioritise and simplify endgame theories to make them more usable for over-the-board games.

Nunn's remarkable ground-breaking trilogy (1992, 1994, 1995) analysed over twenty 'Thompson' profiles of force. It is a prime example of what can be achieved and illustrates all of the above. In a closing summary, Nunn expressed confidence (1995, p. 272) that endgames would continue to be explained "in more human friendly terms".

Such progress will be facilitated by the best possible symbiosis of man and machine, i.e., a combination of their respective capabilities of synthesis and analysis. If the symbiosis is enriched with good information presentation it will aid human cognition considerably (Roycroft and Beal, 1991). This contribution describes an exploration of neural-network data-mining techniques prompted by the second author's short degree-project on this technology. The KPK endgame was chosen as the application domain. The task was to demonstrate how a number of NNs learns to classify positions as wins or draws.

The neural network (NN) is a re-emerging technology. Samuel (1959) referred to NNs but Fürnkranz (1996) found few uses of NNs in computer chess and none on endgame databases. However, NNs have made significant contributions in the field of backgammon, perhaps because it is a game of both skill and chance. Tesauro (1994, 1995) taught NEUROGAMMON from existing games and then created the NN-based TD-GAMMON which learned by playing itself. The latter is now (TD-GAMMON 3.0) the world's best backgammon program, rivals the human champions and has changed the theory of the game.

Section 2 describes the key aspects of the KPK domain and Section 3 those of NN technology. Section 4 describes the design of the NN experiment and Section 5 the training and testing of the NNs chosen. Finally, in Section 6 and 7, we suggest questions for further work and we put NNs in a broader data-mining context.

---

[1] International Computers Limited, Sutton's Park, Reading, UK RG6 1AZ. Email: guy.haworth@icl.com.
[2] Graduate, Cybernetics Department, The University of Reading, UK.

2. THE ENDGAME KPK

Chess has been referred to as the fruit fly of artificial intelligence, an ideal laboratory subject with which to investigate new ideas (Michie, 1980). It provides:

- a synthetic, discrete, finite domain, known worldwide and defined by a few *rules of the game;*
- an almost static domain, those rules being controlled by FIDE and unlikely to change (although the *rules of play* which do not concern us here are more mutable);
- a clear definition of *result* - win, draw, or loss - against which new strategies may be tested;
- a body of past AI work with which any new ideas may be compared;
- a successful history of exercising AI initiatives in the past.

In addition, some 92 of the 145 three- to five-man chess endgames are now documented in basic data terms in databases which are:

- complete and essentially correct;
- of various known sizes, structured and exhibiting within them a range of problem-complexity
- populated by positions each with a defined *value* ('+', '=' or '–') and *depth* indications (+*n* = White wins in *n* ply, 0 = draw, –*n* = Black wins in *n* ply).

Of these 92 endgames, the KPK endgame has the following useful features. It:

- involves fewer positions than 4/5-man endgames by factors of ≈16/950 and ≈60/3600;
- comes with a useful set of concepts used to define principles of successful play;
- has been most used of all endgames to demonstrate a variety of approaches to endgame computer-play;
- features a significant proportion of draws as well as wins for White;
- includes positions with a range of complexity from trivial to deep and non-intuitive;
- is 'd-e axis' symmetric, halving the number of different positions.

Set	Legal?	Piece positions	#	+	=	Notes
1	Yes	{wKg2/h2/h3, wPb2, bKg5: wtm}	3	3	0	Deepest wtm wins: 1. Kg3! 37 ply conv.
2	Yes	{wKh3, wPb2, bKh5/h6: btm}	2	2	0	Deepest btm wins: 1. ... Kg5! 38 ply conv.
3	Yes	{wKd1, wPc3, bKf8: wtm}	1	1	0	1. Kc2! 1. Kd2 draws
4	Yes	{wKd6, wPb2, bKd3: wtm}	1	1	0	1. Kd5! 1. Kc5 is sub-optimal
5	Yes	{wKc4, wPb2, bKc8: wtm}	1	1	0	1. Kb5! not 1. Kb4/Kc5 (sub-optimal)
6a	no	{wKa6/b6/c6/c7/c8, wPa7, bKa8: wtm}	5	0	5	Not reachable from the initial array
6b	no	{wKx6, wPx7, bKx8: x = b/c/d: wtm}	3	3	0	Not reachable from the initial array
6c	no	{wKc1/c2, wPa2, bKa1: wtm}	2	2	0	Not reachable from the initial array
6d	no	{wKc7/c8, wPb6, bKa8: wtm}	2	2	0	Not reachable from the initial array
6e	no	{wKa6/b6, wPc7, bKa8: wtm}	2	2	0	Not reachable from the initial array
7a	no	{wK?, w'P'a8...d8, bK?: btm}	13,980	12,985	995	Post-conversion positions not in KPK
7b	no	{wK?, wPa2...d2, bKx3 in check: btm}	388	6	382	not reachable from the initial array
7c	no	{wKa1, wPa2, bKc1/c2: btm}	2	1	1	not reachable from the initial array

Table 1: A selection of legal and illegal KPK positions.

Like any endgame, KPK can be characterised, for both wtm- and btm-positions in terms of its size, structure and profile. The total size of the domain and the %-split of wtm/btm wins ('+') is clear (see Table 2) despite Clarke's (1977) apparent 'one out' discrepancy. Some 76.5% of wtm-, 58.4% of btm- and 67.3% of all KPK positions are won for White. Table 3 shows the number N_p of 'won' wtm-positions convertible with best play in exactly *p* ply.

The counts usually include a few illegal positions, which are unreachable from the original array (see Table 1) but are difficult to detect. For the record, we have subtracted the ones we know from Edwards' (1994) figures in Table 2 and Clarke's (1977) figures in Table 3. In Table 2, the figures presented are taken from Clarke (1977), Bramer (1980), Bramer (1982), Shapiro and Niblett (1982), Thompson (1986) and Edwards (1994). In the final column we have added our own findings.

	Illegal positions			Clarke (1977)	Bramer (1980)	Bramer (1982)	Sha/Nib (1982)	Thompson (1986)	Edwards (1994)	Haw/Vel (1998)
	#	+	=	database				database	database	legal?
wtm				81,664	----	----	----	81,664	81,664	81,650
wtm, +				62,479	----	----	----	62,480	62,480	62,471
wtm, =				19,185	----	----	----	19,184	19,184	19,179
Pos. sets 6a-e	14	9	5	**included**					**included**	not inc.
btm				97,992	84,012	97,992	83,622		84,012	83,622
btm, +				61,787	48,802				48,802	48,795
btm, + %				63.1%	58.1%				58.1%	58.4%
btm, =				36,205	35,210				35,210	34,827
Pos. set 7a	13,980	12,985	995	**included**	not inc.	**included**	not inc.		not inc.	not inc.
Pos. sets 7b-c	390	7	383	**included**	**included**	**included**	not inc.		**included**	not inc.
btm, P can run							36,958			

Table 2: Sources of numerical data on the KPK endgame.

The structure and profile of the KPK domain is indicated by the existence of subdomains of positions with specific characteristics. Examples relevant here include:

S = {positions p | p $\in$ S $\Rightarrow$ 'best play' successors of p (before conversion) $\in$ S} ...
let such subsets S be defined as 'closed'; S_1 closed & S_2 closed $\Rightarrow$ $S_1 \cap S_2$ closed;
two examples of such closed sets are used in this paper.

C_p = {won wtm positions | won by conversion (P-promotion or mate) with best play in $\leq p$ ply}
Clearly $C_1 \subset C_3 \subset ... \subset C_{37}$: C_p is closed.

F_x = {wtm positions p | Pawn on file x}
{F_x} is a partition of KPK and F_x is closed.

P_q = {positions p | Pawn on file x $\geq$ q}, regularly used in the KPK texts
also used in other endgames with a Pawn (Roycroft and Thompson, 1986; Nunn, 1992, 1995)
P_q is closed and $P_7 \subset P_6 \subset ... \subset P_2$. Therefore $C_p \cap F_x \cap P_q$ is closed.

F_x is specific to KPK and generalises only to endgames featuring Pawns but the nested subsets C_p are more interesting as they can be identified from any endgame database. Suppose an 'agent' (human or silicon) is to achieve a goal, such as W(C_9), 'win any position in C_9'. It need only be able to effect a $C_9 \rightarrow C_7$ transition and achieve W(C_7). Further, the achievement of W(C_7) might inform the achievement of the $C_9 \rightarrow C_7$ transition and therefore of W(C_9). W(C_1) and W(C_3) are clearly the first two of a sequence of ever more inclusive goals of escalating difficulty en route to W(C_{37}).

This suggests that in general, if we want to train an agent to achieve goal G on some domain, it may be better to train it to walk rather than run first. The idea of training first on smaller closed subdomains and ascending to more difficult goals is revisited in Section 7.

Agents which secure wins over C_p may be combined with a standard minimax program searching a tree to depth q to create a hybrid agent which wins over C_{p+q}. The effectiveness of such an approach in KPK can be judged from Table 3 which shows the percentage R_p of wtm wins not in C_p.

The complexity of KPK can be judged from both the chess and technology perspectives:

- 75.7% of White wins in btm-positions (Shapiro and Niblett , 1982, p. 81) involve only running the Pawn;
- Bramer (1980, p. 86) notes that even a USCF-2342 master missed optimal moves and a win;
- Table 1's positions 1-2 give the deepest wtm/btm wins with 37/38 ply to conversion respectively;
- Table 1's positions 3-5 illustrate other difficulties of securing the win and playing optimally:
  in position 3, White's Kc2 rather than Kd2 advances the K and gains *the opposition*;
  in position 4, White's Kd5 stops a bK advance both on the P and to c8; Kc5 does not;
  in position 5, White should not postpone the wKb5/bKb7 confrontation; Kb4/Kc5 are sub-optimal;
- the successful completion of computer programs has proved remarkably difficult:
  a residue of positions repeatedly escaping classification by a growing set of rules.

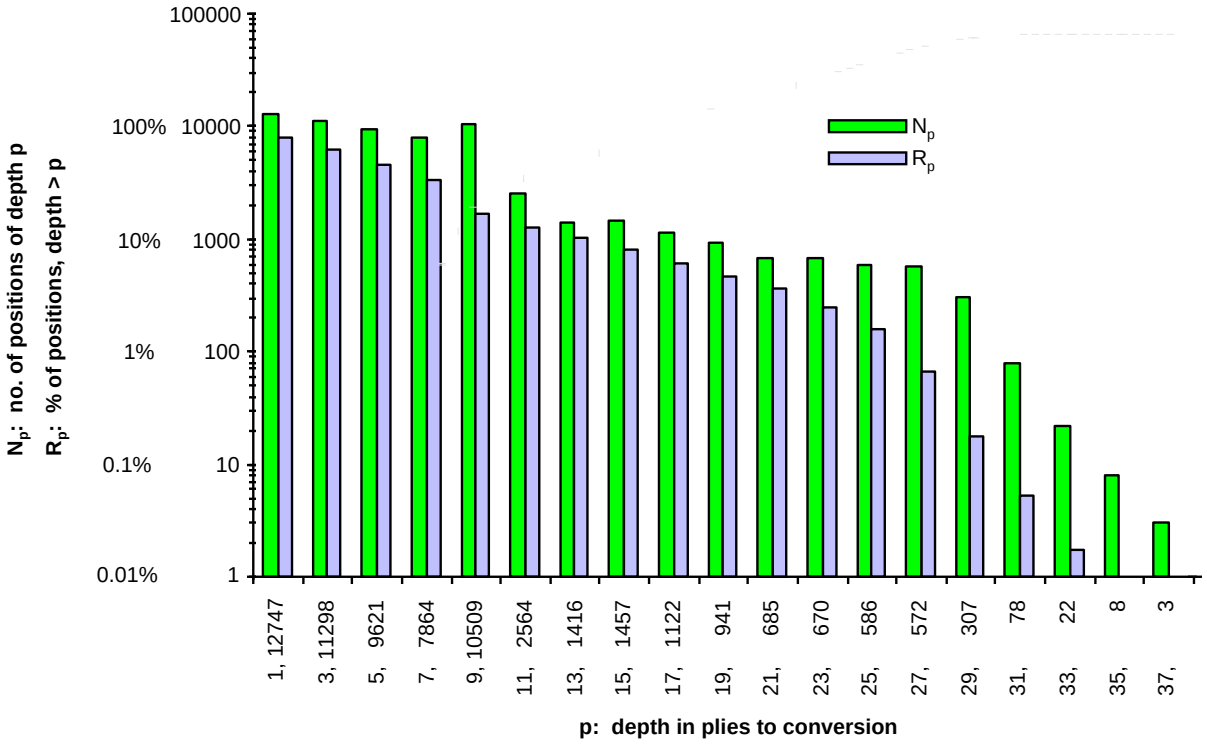

Table 3: wtm KPK wins: N_p and R_p for each (odd) ply-count *p*.

3. NEURAL-NETWORK BASICS

Figure 1 shows a multilayer perceptron (MLP), the particular type of NN chosen here. The MLP illustrated is a *three-layer* NN a-b-c-d as it has three layers of neurons. The general MLP is organised as *k* layers of N_x neurons each. To simplify nomenclature, the input nodes, which are not in fact neurons, are called *input neurons* in layer 0. Each layer also has a notional neuron N_{0x} outputting '1's (see the second row of Figure 1). Their weights $-t_{i,x+1}$ are used to define the input level at which the neurons *threshold*, that is, make their most rapid transition from 'output low' to 'output high'. The neurons in layers 1, ... , *k*-1 are *hidden neurons* and those in the final layer '*k*' are *output neurons*. Hence, Figure 1 distinguishes the input neurons, (N_{0x},), the hidden neurons and the output neurons from each other.

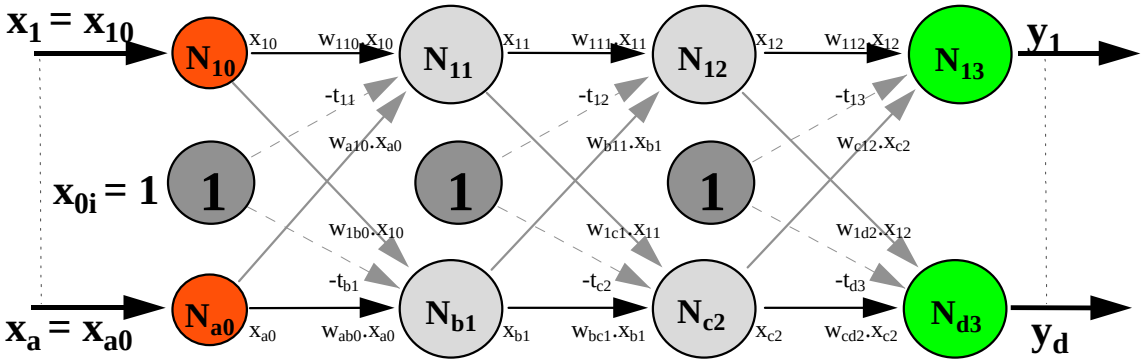

Figure 1: three-level MLP neural network a-b-c-d.

Representing the weights by $\underline{\mathbf{w}}$ and the inputs by $\underline{\mathbf{x}}$ with $x_0 = 1$ (and all other parameters variable), the activation function for the neurons was taken to be the normal 'sum-squash' sigmoid function:

$y = f(\underline{\mathbf{w}}.\underline{\mathbf{x}}) = f(z) = 1/(1 + e^{-bz})$ with $b > 0$.
Therefore, $f(z) \in (0, 1)$, $f(0) = 0.5$, $f(\infty) = 1$, $f(-\infty) = 0$ and $f(z) + f(-z) = 1$,
$f'(z) = b.e^{-bz}/(1+e^{-bz})^2 = b.f(z).f(-z)$
$= b.f(z).(1 - f(z)) > 0$, a relatively simple calculation.
As $b \rightarrow \infty$, the function f tends to the *Heaviside* step function. In this paper $b = 1$.

A *fully-connected* NN has every one of the $\sum_x N_x * N_{x+1}$ possible connections made: neuron N_{ix} in layer 'x' transmits its output with weight w_{ijx} to $N_{j,x+1}$ in the layer 'x+1' only. It may be that many of these connections are not significant and can be pruned out without loss of NN functionality. This speeds up training and improves the chances of inferring higher level rules of classification from the NN after the training phase.

A neural network requiring W weights each of B storage bits requires $W.B+\varepsilon$ bits to represent it and this storage requirement may be compared with other ways of storing the function it performs to give a *compression factor*. Such an NN can be in only one of $2^{W.B}$ states and can therefore only represent, with a given characterisation and coding of input and output, at most one of $2^{W.B}$ functions.

The *pure gradient descent* version of the back-propagation training procedure BPT (Rich and Knight, 1991) may be parallelised on multi-processor computers and reads:

set the weights initially at random to values in the [-0.1, +0.1] range
select a training set T of position characterisations $\{\underline{\mathbf{x}}_i\}$ with which to train the MLP
***start_loop*:** in one sequence or *epoch* of training ...
input each of the vectors $\underline{\mathbf{x}}_i$ in T and store the output vector $\underline{\mathbf{y}}_i$
after completing the epoch of training:
compare the outputs $\{\underline{\mathbf{y}}_i\}$ with the goal values $\{\underline{\mathbf{z}}_i\}$, calculating $\{\underline{\mathbf{y}}_i - \underline{\mathbf{z}}_i\}$
note how many of the instances in T are correctly classified
decide whether another epoch of training is required and if not, terminate
adjust the NN's weights $\underline{\mathbf{w}}$ to converge the outputs $\{\underline{\mathbf{y}}_i\}$ on goals $\{\underline{\mathbf{z}}_i\}$
loop to ***start_loop*** for the next epoch of training.

The original perceptron (Minsky and Papert, 1969) has just one layer and one neuron. It can rank objects, positions in this case, in the same order as a single linear weighted sum (SLWS) $\underline{\mathbf{w}}.\underline{\mathbf{x}}$. Such SLWSs were proposed and implemented by Samuel (1959) in checkers and are commonly used as the position evaluation functions in game programs. If a set of positions can be classified correctly into two subsets by the sign of the linear form $\underline{\mathbf{D}}.\underline{\mathbf{x}}$, a perceptron's weights $\underline{\mathbf{w}}$ can be iterated sufficiently close to $\underline{\mathbf{D}}$ to enable the perceptron to classify the positions correctly as well.

However, it is likely that the evaluation of positions using one linear function of their characteristics gives only an approximate guide to their correct ranking. Samuel (1967) concluded that an SLWS was too restricted to evaluate checkers positions effectively. Berliner (1980) improved the evaluation function of BKG 9.8, an early backgammon program, by upgrading from a SLWS to multiple LWS's (MLWS) and then, with more success, to a non-linear function. BKG 9.8 was the first program to defeat a World Champion at any board or card game. Tesauro, who later worked on DEEPER BLUE, also credits the shift from SLWS to non-linear NNs for the major improvements in his backgammon programs. The Othello program LOGISTELLO (Buro, 1995) was improved by changing from one SLWS to the next every four moves. Perhaps we should not be surprised that, for games requiring the effective combination of forces, position evaluation might involve non-linear evaluation of features on the board.

The study of perceptrons stalled when it was discovered (Minsky and Papert, 1969) that they had severe limitations in 'non-linear situations'. Their inability to emulate the XOR logic function is often illustrated in the textbooks. Fortunately, the generalisation of the single-layer perceptron to the MLP network and the discovery of the back-propagation training (BPT) method for MLPs revived the application of NNs (Rumelhart, Hinton and Williams, 1987; Beale and Jackson, 1990). Note that the neurons' activation function must be non-linear if the MLP is to react in a non-linear way to its inputs.

The increased adoption of non-linear functions coupled with the ability of MLPs to emulate such functions may offer a way forward for computer chess and other computer-played games. For example, any existing linear function $\underline{\mathbf{m}}.\underline{\mathbf{x}}$ used for evaluating chess positions on *n* characteristics can be used to set up the initial

position for training of not only an *n*-1 perceptron as above but of a potentially more subtle *n*-*n*-1 NN. In the notation above, the initialisation should be $w_{ii0} = 0.1$, $w_{ij0} \approx \varepsilon^2$ $(i \neq j)$ and $w_{i11} = \varepsilon m_i$ where $\varepsilon << m_i$ and suitably small.

4. THE DESIGN OF THE NEURAL-NETWORK EXPERIMENT

A number of decisions needed to be made about:

- the goal ***G*** of the NN experiments on KPK,
- the characterisation ***C*** of KPK positions,
- the coding ***I*** of the characterisation as input to the NNs,
- the coding and interpretation ***O*** of the output,
- the number and size of the hidden layers ***H***,
- the choice of training sets ***T*** and testing sets ***S***, and
- the management ***M***, including termination decision, of the training phase.

Therefore, a NN experiment actually has parameters (G, C, I, O, H, T, S, M). The experimental space is indeed a large and complex one with a few navigational aids.

4.1 The Characterisation of KPK Positions

The experimental goal chosen here was to demonstrate NNs learning by training and to show to what extent the chosen NNs could classify wtm- or btm-KPK positions correctly into wins and draws. The classification of wtm and btm positions was seen as two tasks requiring distinct NNs.

The characterisation of the positions had to be *adequate* for the chosen goal and in terms of features which were *computable*. The characterisation should ideally lead to the *explanation* of concise classification rules if these exist.

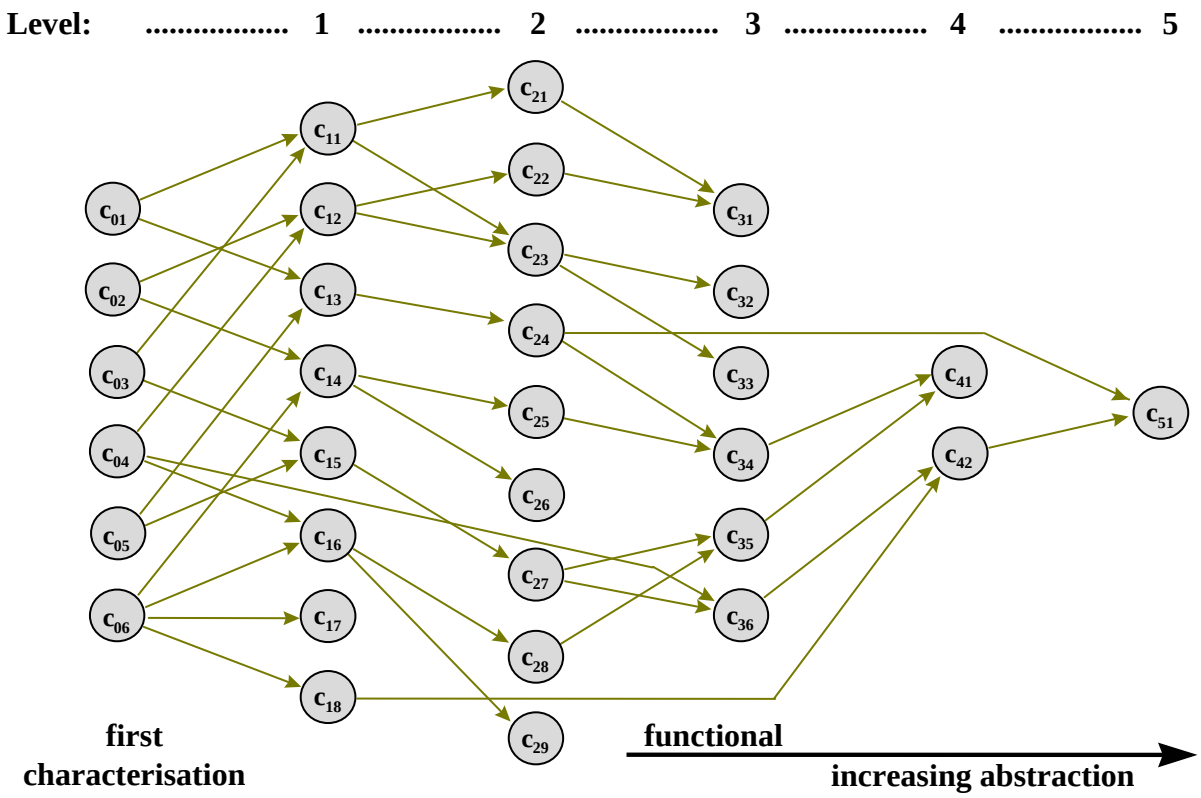

Figure 2: The lattice of characteristics and levels of abstraction.

An *inadequate* characterisation by definition makes correct classification impossible: apples and oranges cannot be distinguished for example on the basis of size and weight. The *physical characterisation* of KPK using only the physical positions of the pieces is certainly adequate. However, the theory of KPK is discussed in more abstract concepts such as *relative position*, *relative distance*, *key squares* and *opposition*. Such abstractions are useful because they capture in a compact way the essence of some feature while suppressing unnecessary details. Further, past KPK knowledge representations (Beal, 1980; Bramer, 1980, 1982; Shapiro and Niblett, 1982) incorporated these abstractions in a large number of arithmetical rules which no doubt frustrated their authors with their increasing complexity and decreasing applicability.

There is also plenty of evidence from other games (Buro, 1995; Tesauro, 1995) that concepts established in the theory of a game (1) enrich the basic physical characterisation, (2) expedite the NN training process and (3) facilitate the interpretation of the characterised data in an evaluation function.

The characterisation can be thought of as being built up in layers (Quinlan, 1979), later characteristics being functionally derived from earlier ones as illustrated in Figure 2. The rule of assigning layer numbers L(c) to the characteristics 'c' is here:

$L(c) \geq 0$; L(c) is minimised subject to constraint '$c = f(c_1, \ldots, c_n) \Rightarrow L(c) > L(c_i)$'

If for example, $c_2 = |c_1|$ it is clear that $L(c_2) > L(c_1)$. But if $c_3 = c_1 - c_2$, then $c_1 = c_2 + c_3$ and $c_2 = c_1 - c_3$. Therefore, one of c_1, c_2 or c_3 must be chosen to be a function of the other two c_i.

Level	Char.	Description	x_i	Formula	Range	|Range|
0	c_{01}	f(wK): file of wK	x_1	x_1	[1, 8]	8
0	c_{02}	r(wK): rank of wK	x_2	x_2	[1, 8]	8
0	c_{03}	f(bK): file of bK	x_3	x_3	[1, 8]	8
0	c_{04}	r(bK): rank of bK	x_4	x_4	[1, 8]	8
0	c_{05}	f(wP): file of wP	x_5	x_5	[1, 4]	4
0	c_{06}	r(wP): rank of wP	x_6	x_6	[2, 7]	6
1	c_{11}	fd(wK, bK): wK-bK file difference	x_7	$x_1 - x_3$	[-7, 7]	15
1	c_{12}	rd(wK, bK): wK-bK rank difference	x_8	$x_2 - x_4$	[-7, 7]	15
1	c_{13}	fd(wK, wP): wK-wP file difference	x_9	$x_1 - x_5$	[-3, 7]	11
1	c_{14}	rd(wK, wP): wK-wP rank difference	x_{10}	$x_2 - x_6$	[-6, 6]	13
1	c_{15}	fd(bK, wP): bK-wP file difference	x_{11}	$x_3 - x_5$	[-3, 7]	11
1	c_{16}	rd(bK, wP): bK-wP rank difference	x_{12}	$x_4 - x_6$	[-6, 6]	13
1	c_{17}	r(wP) = 8?	x_{13}	$0;\ x_6 = 8 \Rightarrow 1$	[-6, 6]	13
1	c_{18}	no. of P-moves needed to convert P	x_{14}	$8 - x_6;\ x_6 = 2 \Rightarrow 5$	[1, 5]	5
2	c_{21}	abs fd(wK, bK)	x_{15}	$\|x_7\|$	[0, 7]	8
2	c_{22}	abs rd(wK, bK)	x_{16}	$\|x_8\|$	[0, 7]	8
2	c_{23}	parity re 'opposition'	x_{17}	$0;\ x_7$ even & x_8 even $\Rightarrow 1$	[0, 1]	2
2	c_{24}	abs fd(wK, wP)	x_{18}	$\|x_9\|$	[0, 7]	8
2	c_{25}	abs rd(wK, wP)	x_{19}	$\|x_{10}\|$	[0, 6]	7
2	c_{26}	wK behind wP?	x_{20}	$0;\ x_{10} < 0 \Rightarrow 1$	[0, 1]	2
2	c_{27}	abs fd(bK, wP)	x_{21}	$\|x_{11}\|$	[0, 7]	8
2	c_{28}	abs rd(bK, wP)	x_{22}	$\|x_{12}\|$	[0, 6]	7
2	c_{29}	bK behind wP?	x_{23}	$0;\ x_{12} < 0 \Rightarrow 1$	[0, 1]	2
3	c_{31}	d(wK, bK): wK-bK 'distance'	x_{24}	$\max(x_{15}, x_{16})$	[2, 7]	6
3	c_{32}	Wh. 'may have opposition'	x_{25}	$0;\ wtm/x_{17} = 0$ or $btm/x_{17} = 1 \Rightarrow 1$	[0, 1]	2
3	c_{33}	Bl. 'may have opposition'	x_{26}	$0;\ wtm/x_{17} = 1$ or $btm/x_{17} = 0 \Rightarrow 1$	[0, 1]	2
3	c_{34}	d(wK, wP): wK-wP 'distance'	x_{27}	$\max(x_{18}, x_{19})$	[1, 7]	7
3	c_{35}	d(bK, wP): bK-wP 'distance'	x_{28}	$\max(x_{21}, x_{22})$	[1, 7]	7
3	c_{36}	'distance', bK to wP's queening sq.	x_{29}	$\max(x_{21}, \|8 - x_4\|)$	[0, 7]	8
4	c_{41}	d(wK, wP) – d(bK, wP)	x_{30}	$x_{27} - x_{28}$	[-6, 6]	13
4	c_{42}	d(bK, Qs) – m(P, Qs)	x_{31}	$x_{29} - x_{14}$	[-5, 6]	12
5	c_{51}	P can run and convert?	x_{32}	$x_{18} > 0$ & $x_{31} > 1$ (btm, 2) $\Rightarrow 1$	[0, 1]	2

Table 4: Input format and characterisation of KPK.

The chosen characterisation, shown in Figure 2 and Table 4, was compiled as a practical subset of the available theory and included the following features:

- relative distances of the black King and Pawn from the Pawn's 'queening' square,
- rank and file positions of the Kings relative to the Pawn, and
- '*possible opposition*' indicators based on the parity of the Kings' relative rank and file positions.

All characteristics were included in the chosen characterisation if they were en route to any characteristic that was also included. Note that this is not always necessary; had the board had an infinite number of files, x_1, x_3 and x_5 would have been irrelevant. It was not necessary to indicate which side had the move as an individual NN was being presented with either wtm- or btm-positions.

Positions which were clearly illegal or known to be unreachable (see Table 1) were not included in the training and testing sets. In this spirit, a belated decision was taken not to follow Clarke (1977) who included positions with the Pawn on the 8th rank. Input x_{13} therefore became redundant and could have been removed. Similarly, one of x_{25} and x_{26} is redundant but both would have been required if a closer approximation to the concept of *opposition* had been used which occasionally credited neither side with having the opposition.

The concept 'Kings on opposite sides of the Pawn' was not included; example 3 of Table 1 suggests that it might also have been useful. Nor did the characterisation emphasise 'Pawn on 2nd rank' or 'King distances from c8' in the context of the Pawn being on the a-file.

4.2 Coding the Characterisation of Positions

The 'natural' coding was chosen for the KPK NNs, each input taking the values indicated under 'range' in Table 4. There was more than one alternative, as illustrated below, but these seemed to have no compelling advantages.

A binary coding of the input could have been used with for example c_{11} (x_7 in Table 4) being represented by 4 binary inputs covering the range 0-14. The win/draw classification problem using for example just the six basic characteristics c_{01}–c_{06} then becomes one of emulating a Boolean function on 17 Boolean variables.

The reduction of the classification problem to one of emulating a Boolean function appears to be attractive, given a concise but little known result by Messom (1992) cited by Hinde *et al.* (1997):

> any Boolean function on *n* variables may be emulated by an *n*-*n*-1 NN with only $(n+1)^2$ weights

Although this result nominates an adequate NN and potentially removes the need to choose any other NN, a quick calculation shows that the price to be paid is in the precision of the weights. There are 2^n values for a vector of *n* Boolean variables and $2\hat{}2^n$ Boolean functions mapping these to {false, true}. Therefore, as $(n+1)^2$ weights of B bits can only be in one of $2\hat{}(n+1)^2B$ states, the number of storage bits required for at least some of the NNs' weights must be greater than $2^n/(n+1)^2$. This implies for example at least 42, 73 and 405 bits when *n* takes the values 13, 14 and 17. The chosen characterisation would have required 92 binary inputs.

Given a standard representation of weight numbers, the least possible value of the maximum number of bits required in any weight to emulate a particular Boolean function could act as the Boolean function's *class number* and possibly as its *complexity index*.

The Boolean coding of the characterisation was not chosen for two further reasons. First, it seems to lose the apparently essential arithmetical properties of 'order' and 'difference'. Second, it is not clear how quickly the *n*-*n*-1 network will train or whether it will reveal more readily any higher-level concepts of value when trained.

Tesauro (1992a, Section 2.3) regarded this and the output coding as key decisions; his choice of characterisation coding for TD-GAMMON used neither of the 'natural' or 'binary' extremes. The number of white or black pieces on a point of the board was represented by eight inputs, four for each colour and 192 inputs in all. As pieces of only one colour are allowed on each point, at least one of these quartets would represent 'zero'. Each of the first three inputs (per colour) represented the absence/presence of the first, second and third pieces by 0/1; the fourth input was 0 or (*n*-3)/2 if there were more than 3 pieces on the point. Tesauro's aim seems to have been to keep the coding straightforward and the inputs mainly in the [0, 1] range.

4.3 Coding Target Outputs; Interpreting Actual Outputs

Two techniques were adopted here. Five of the six NNs used had just one output neuron while the last one had two outputs. The outputs y_i are in the (0, 1) range (see Figure 1).

For single-output NNs, the target values for y_1 were '1' for a win and '0' for a draw. Actual outputs were always interpreted by "$y_1 \geq 0.5$ indicates 'win'; $y_1 < 0.5$ indicates 'draw' ".

For the last NN, we followed the recommendation of providing an NN with T outputs if it is intended to classify objects into T types: the NN was given two outputs. Target values for (y_1, y_2) were (1, 0) for a win and (0, 1) for a draw. Actual outputs were interpreted by the rule:

"$y_1 \geq 0.5 + m$, $y_2 < 0.5 - m$ indicates 'win'; $y_1 < 0.5 - m$, $y_2 \geq 0.5 + m$ indicates 'draw' ", m = 0/0.4

with the 0.4 value being used in the stopping condition for trials 38 to 41 in Table 5. If m > 0, this allows an NN to 'pass' on a hybrid panel of advisors (Walker, 1996) on some occasions. Table 6 shows that the NN succeeded in producing symmetric outputs on y_1 and y_2 as requested.

Had the NN been trained not only on position value but on depth to win for won positions, the coding of target *depth d* would have had to be decided. Perhaps $1 - [\log_e(d + 4)/4]$ would have sufficed as depth in ply is less than 40 for KPK.

For training positions, the classification can be noted either as soon as the training position is encountered or at the end of an epoch of training. The experiments here did the latter.

4.4 Choice of Hidden Topology

Tesauro's TD-GAMMON has an NN with just two levels but some 40 to 160 hidden neurons (Sutton and Barto, 1998). However, there are few guidelines about the efficacy of 2-layer NNs and they are not sufficient to guarantee that a 2-layer NN will be effective for the chosen experimental goal. Messom's (1992) result is an existence proof only and says nothing about the required precision of the NN's weights.

Beale and Jackson (1990, pp. 84-85) note that, provided the characterisation is adequate:

- a 2-layer NN can adequately classify objects in a single *convex hull* in characterisation space, the convex hull being bounded by *hyperplanes* $\underline{\mathbf{w}_i}.\underline{\mathbf{x}} = 0$ **(**i = 1, ... , k);
- a set of N objects can be segregated from those objects not in the set by ≤ N convex hulls;
- a 3-layer NN can classify N objects by OR-ing together outputs from ≤ N 2-layer NNs.

A decision was therefore made to adopt 3-layer NNs throughout rather than experiment with possibly limited 2-layer NNs. The decision cost is that 3-layer NNs take longer to train than 2-layer NNs.

The first and second layers of hidden neurons had N_1 and N_2 neurons with $N_1 \geq N_2$ because the 'win subspace' was expected to involve more hyperplanes than convex hulls. The numbers chosen fairly arbitrarily probably varied from the parsimonious to the generous. Table 5 indicates in column 3 which NN was being used in any particular trial.

4.5 The Management of NN Training

The classification of wtm- and btm-positions was regarded as two separate problems. While a program interface to Thompson's KPK database was being developed, two small sets of respectively wtm- and btm-positions were hand prepared. These positions were either interesting 'borderline' ones from the texts or were generated at random. They were checked for *legality* and *reachability* and evaluated as wins or draws. These sets were then randomly divided into two sets, creating the training sets T_1 and T_2 and the testing sets S_1 and S_2, to ensure as far as possible that T_i was characteristic of S_i. No position in S_i is in T_i to ensure that the positions used in the testing phase are all previously unseen.

Later, the Thompson KPK database yielded the three large sets T_3 , T_4 and T_5. These sets were used in their entirety in the training mode. The complete collection of position sets was therefore:

T_1 and T_2	125 wtm- and 136 btm-positions respectively,
S_1 and S_2	(the testing sets) 125 wtm- and 136 btm-positions: $T_1 \cap S_1 = T_2 \cap S_2 = \varnothing$,
T_3	all 81,664 wtm positions,
T_4	F_a (see Section 2): the 20,643 wtm positions with the wP on the a-file
T_5	F_d: the 20,346 wtm-positions with the wP on the d-file.

The positions were presented in an arbitrary order which did not change from one training epoch to the next. Note that no attempt was made to present positions in any defined order of complexity.

With y_{ij} as the actual output value and d_{ij} as the target value of output neuron j to object i, error-functions E_1 and E_2 were defined as E_1 = average $|y_{ij} - d_{ij}|$ and E_2 = max $|y_{ij} - d_{ij}|$. Clearly, $E_2 > E_1$.

Targets were set for the reduction of E_1 and E_2; these were variously 0.05, 0.1 and 0.15. Training was terminated if, for the last 20 epochs of training, the error target had been exceeded or the NN had failed to improve the number of positions classified correctly.

Training was carried out using only the training set of positions and the trained network was then tested if necessary on the testing set of unseen positions. The training algorithm (Rich and Knight, 1991) was a variant of the 'pure gradient descent method' described previously and was as follows:

set the weights initially at random to **init** with values in the [-0.1, +0.1] range,
***start_loop*:** perform an *epoch* of training, presenting each input *training set* position in turn
after presenting each position characterisation vector $\underline{\mathbf{x}}_i$:
    compare the outputs $\underline{\mathbf{y}}_i$ with the goal values $\underline{\mathbf{d}}_i$
    adjust the NN's weights $\underline{\mathbf{w}}$ such that the output $\underline{\mathbf{y}}_i$ would have been closer to $\underline{\mathbf{d}}_i$
    (no use was made of the NN-training concepts of *momentum* and *annealing*)
at the end of an epoch of training, decide whether another epoch of training is needed:
    classify the positions of the *training set* using the end-epoch state of the NN ...
    noting the individual output errors $|y_{ij} - d_{ij}|$ for each position,
    (record the state of the NN in terms of its weights' values, $E_{1/2}$ and %-accuracy)
    consider whether the error measure $E_{1/2}$ has been consistently below the error target E, and
    consider whether the NN's success rate on the *training set* positions has apparently peaked
if another epoch of training is required, return to ***start_loop***
if another epoch is not indicated, choose the end-of-epoch NN-state with the minimal $E_{1/2}$

The training process is effectively 'programming by example'. It replaces the intellectual process of deciding how to apply a set of identified rules either manually (Beal and Clarke, 1980; Bramer, 1980) or with a degree of automation (Quinlan, 1979; Bramer, 1982; Shapiro and Niblett, 1982).

Testing the network after the training phase merely involves presenting the positions of the *testing set* to the NN and interpreting the outputs as described in Section 4.3.

5. THE EXPERIMENTAL RESULTS

Table 5 provides an overview of our experimental results. The column *comp* compares the memory requirements of a KPK database (taken to be 2^{17} bits) with the memory requirements to store the NNs' weights. An NN which effectively stores some data but with a smaller memory requirement has effectively *compressed* that data. Each weight was a 64-bit floating point number and no attempt was made to reduce the size or number of weights by pruning out small weights or rounding to 32-bit form.

Trial #	wtm/ btm	Neural network #	Topology	# wts.	bits	comp.	Tr. set T	Target Error e	Epochs taken	Training Accuracy, % correct		Testing Accuracy, % correct	
For Error Function E1													
1	wtm	1	32-10-10-1	451	28,864	4.541	1	0.15	20	94	6	77	23
2							1		29	92	8	74	26
3							1	0.10	39	98	2	82	18
4							1		33	96	4	79	21
5							1	0.05	123	98	2	76	24
6							1		99	98	2	77	23
7							1		320	97	3	75	25
8		2	32-20-10-1	881	56,384	2.325	1	0.15	17	97	3	78	22
9							1	0.10	28	98	2	79	21
10							1	0.05	50	98	2	80	20
11		3	32-32-5-1	1,227	78,528	1.669	1	0.15	17	98	2	76	24
12							1	0.10	22	97	3	81	19
13							1	0.05	48	99	1	79	21
14	btm	1	32-20-10-1	881	56,384	2.325	2	0.15	37	92	8	82	18
15							2	0.10	48	97	3	83	17
16							2	0.05	130	98	2	87	13
17							2		235	98	2	84	16
18		3	32-32-5-1	1,227	78,528	1.669	2	0.15	41	93	7	87	13
19							2	0.10	70	98	2	85	15
20							2	0.05	99	98	2	88	12
For Error Function E2													
21	wtm	3	32-32-5-1	1,227	78,528	1.669	1	0.10	90	98	2	77	23
22							1	0.10	53	100	0	79	21
23							1	0.10	113	100	0	81	19
24		4	32-64-10-1	2,773	177,472	0.739	1	0.10	60	100	0	77	23
25							1	0.10	67	99	1	79	21
26							1	0.10	60	99	1	77	23
27		5	32-64-20-1	3,433	219,712	0.597	1	0.10	49	100	0	74	26
28							1	0.10	48	100	0	76	24
29	btm	3	32-32-5-1	1,227	78,528	1.669	2	0.10	101	99	1	87	13
30							2	0.10	116	100	0	90	10
31							2	0.10	194	100	0	86	14
32							2	0.10	300	100	0	86	14
33		4	32-64-10-1	2,773	177,472	0.739	2	0.10	105	100	0	88	12
34							2	0.10	110	100	0	89	11
35							2	0.10	79	99	1	87	13
36		5	32-64-20-1	3,433	219,712	0.597	2	0.10	90	99	1	87	13
37							2	0.10	74	100	0	88	12
... and later, for NN_6 and Error Function E2													
38	wtm	6	32-32-5-2	1,233	78,912	1.661	3	0.10	1,000	95	5	---	---
39							4	0.10	1,610	99.6	0.4	---	---
40									6,285	99.8	0.2	---	---
41							5	0.10	461	97.3	2.7	---	---

Table 5: NN training and testing results.

5.1 Comments on the Results

Note first that all the trials achieved the immediate objective which was to demonstrate NNs learning to classify KPK positions. The choice of 3-layer networks throughout made this more likely. Because the initial

state of the NN was randomly chosen, it was expected that only half the positions would initially be classified correctly and this was the case. However, the first few epochs of training typically improved accuracy rapidly.

All trials produced NNs with at least 92% accuracy on the training set. For comparison, note that an NN configured to indicate that all wtm-positions were wins would be 76.5% accurate and an NN configured to indicate all btm-positions as wins for White would be 58.4% accurate (see Section 2). The difference between %-accuracy achieved and % of 'wins' was greatest for NNs trained on btm-positions.

The lowest success rates in trials 2 and 14 were due to the less rigorous error function E1 and the least challenging target of 0.15. The use of the more rigorous error function E2 significantly improved the NNs' accuracy in classifying the positions of the training set and was used exclusively for the larger networks.

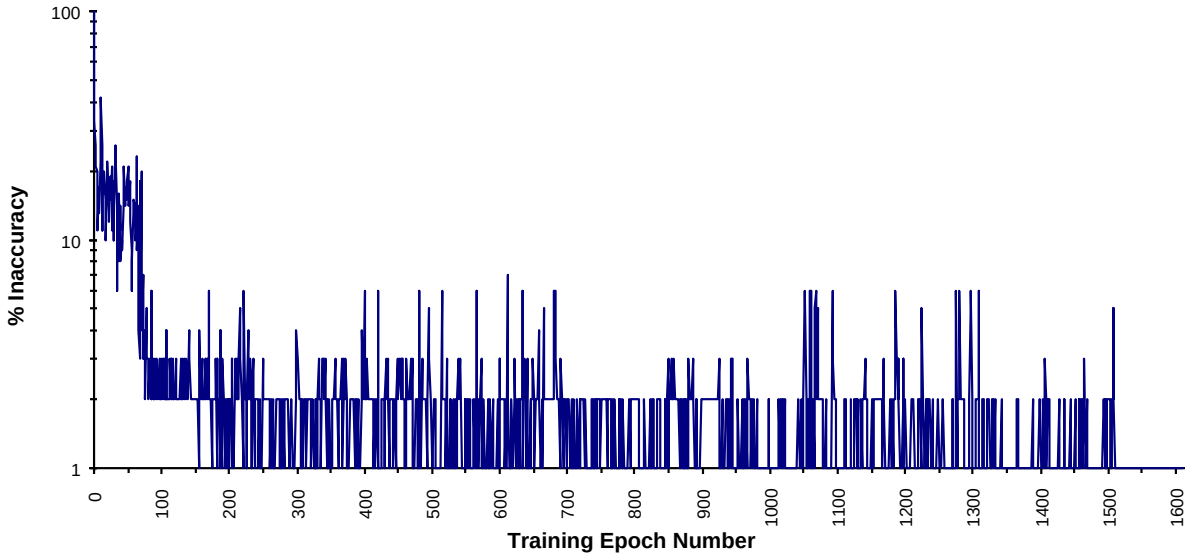

Figure 3: Residual inaccuracy of NN on a-file KPK wtm-positions.

The reduction in the percentage of misclassified positions for trial 39 (see Figure 3), shows that progress as measured by this criterion is far from monotonic. The NNs cited in Table 5 were not necessarily the final state of the NNs at the end of the training period.

The success rate on the 'new' testing-set positions was predictably lower than on the training-set positions. This is usually because the training set is not intrinsically characteristic of the testing set, or because the NN *identifies* too closely with the training set through overtraining. As positions were allocated at random to the small training and testing sets, the lower effectiveness on the testing set suggests overtraining. The slightly better performance on btm-positions is interesting and unexplained; it is not thought to have been caused by the slightly larger training set T_2.

No clear pattern emerged as to what shape or size of NN was the best choice. The NNs with a compression factor of less than one did not prove conspicuously more effective. The last NN exercised on F_a and F_d certainly demonstrated that compression can be achieved as misclassified positions could be filed in a residual database and checked before the NN was consulted.

Trial 39 on the most difficult file subset F_a left only 51 positions out of 20,643 misclassified. An examination of these positions in Table 6 is a salutary reminder that the NN has not been told about most aspects of chess, for instance, the rules of chess or the movement of pieces. Its essentially arithmetical/logical view of the positions has its limitations – but this is true of any static position evaluation function. Table 6 also indicates that some win-defining rules input as characteristics were not 100% effective (see # 8, 17, 21, 22, 24). Therefore we see an argument for predefining subsets of the NN to force the effect of known rules.

Of the 51 misclassified positions, 42 are *critical* in that the result is affected by whose move it is. Eight are drawn with either wtm or btm (# 1, 17, 21, 22, 24, 41, 49, 51) and one (# 33) is a win with either side to move.

Some positions translated one file to the right (e.g., # 1) would have different values but most (e.g., 'wins' and # 17, 21, 22, 24) would not. Positions # 29 to 38 remind us that the wP is sometimes better on a2 than a3. The bK cannot move into check on b3, is forced to move to b2, and allows the wP to escape to a4. Neither the 'a2→a4' or the 'cannot move into check' rules are known to the NN. Nor is the fact that the board has only eight ranks and files. On reflection, it is surprising that the NN manages to classify the other 20,592 positions correctly.

				result ...		NN Outputs								result ...		NN Outputs			
#	wK	bK	wP	w	(b)	Target		Actual		#	wK	bK	wP	w	(b)	Target		Actual	
				to move		o1	o2	o1	o2					to move		o1	o2	o1	o2
1	**a8**	**c7**	**a7**	**d**	**d**	0	1	0.999	0.001	**27**	a4	f8	a3	w	d	1	0	0.090	0.910
2	c6	c8	a6	w	d	1	0	0.001	0.999	**28**	a4	f7	a3	w	d	1	0	0.090	0.910
3	b8	b6	a6	w	d	1	0	0.000	1.000	**29**	g6	f3	a2	w	d	1	0	0.007	0.993
4	b6	c8	a6	w	d	1	0	0.149	0.851	**30**	g5	f3	a2	w	d	1	0	0.007	0.993
5	a8	c6	a6	w	d	1	0	0.089	0.911	**31**	f6	f3	a2	w	d	1	0	0.100	0.900
6	a7	c6	a6	w	d	1	0	0.089	0.911	**32**	f6	e3	a2	w	d	1	0	0.007	0.993
7	a7	b5	a5	w	d	1	0	0.001	0.999	**33**	**f5**	**f3**	**a2**	**w**	**w**	1	0	0.100	0.900
8	h8	f8	a4	w	d	1	0	0.007	0.993	**34**	f5	e3	a2	w	d	1	0	0.007	0.993
9	d5	e8	a4	w	d	1	0	0.001	0.999	**35**	e6	d3	a2	w	d	1	0	0.000	1.000
10	a6	b4	a4	w	d	1	0	0.128	0.872	**36**	e5	d3	a2	w	d	1	0	0.007	0.993
11	a5	e8	a4	w	d	1	0	0.090	0.910	**37**	d6	c3	a2	w	d	1	0	0.000	1.000
12	e4	f8	a3	w	d	1	0	0.001	0.999	**38**	d5	c3	a2	w	d	1	0	0.000	1.000
13	e4	f7	a3	w	d	1	0	0.090	0.910	**39**	c7	d5	a2	w	d	1	0	0.091	0.909
14	d5	e8	a3	w	d	1	0	0.090	0.910	**40**	b7	d5	a2	w	d	1	0	0.091	0.909
15	d4	f8	a3	w	d	1	0	0.090	0.910	**41**	**b4**	**e6**	**a2**	**d**	**d**	0	1	0.999	0.001
16	d4	f7	a3	w	d	1	0	0.090	0.910	**42**	a8	d2	a2	w	d	1	0	0.094	0.906
17	**c8**	**d3**	**a3**	**d**	**d**	0	1	1.000	0.000	**43**	a8	c2	a2	w	d	1	0	0.091	0.909
18	c5	e8	a3	w	d	1	0	0.090	0.910	**44**	a8	b2	a2	w	d	1	0	0.091	0.909
19	c4	f8	a3	w	d	1	0	0.090	0.910	**45**	a7	d5	a2	w	d	1	0	0.091	0.909
20	c4	f7	a3	w	d	1	0	0.090	0.910	**46**	a7	c2	a2	w	d	1	0	0.091	0.909
21	**b8**	**d3**	**a3**	**d**	**d**	0	1	1.000	0.000	**47**	a7	b2	a2	w	d	1	0	0.091	0.909
22	**b7**	**c3**	**a3**	**d**	**d**	0	1	1.000	0.000	**48**	a6	b2	a2	w	d	1	0	0.091	0.909
23	b4	f8	a3	w	d	1	0	0.090	0.910	**49**	**a5**	**c4**	**a2**	**d**	**d**	0	1	1.000	0.000
24	**a8**	**d3**	**a3**	**d**	**d**	0	1	0.999	0.001	**50**	a3	g7	a2	w	d	1	0	0.090	0.910
25	a8	c2	a3	w	d	1	0	0.092	0.908	**51**	**a3**	**c3**	**a2**	**d**	**d**	0	1	0.985	0.015
26	a5	e8	a3	w	d	1	0	0.090	0.910										

Table 6: The 51 misclassified 'P on the a-file' positions.

6. THE EXPERIMENTAL TECHNIQUE AND FUTURE TASKS

In the project as defined there was no time available to broaden the experimental goals or to finesse the experimental technique by following some tempting avenues of investigation which revealed themselves along the way. Other workers in this field may wish to take up the enquiry on some of the topics below. As discussed in Section 4, questions arose about characterising the positions, the NN-topology and training, and the chess implications of the results. Some examples are:

- What would 2-layer MLPs have achieved in comparison with 3-layer MLPs?
- What would an *n*-*n*-1 MLP have achieved on a Boolean coding of a KPK characterisation?
- Would the evolution of training/testing accuracy on $T_{1/2}/S_{1/2}$ have demonstrated over-training?
- How would training/testing accuracy for an NN vary as the size of training/testing sets increased?
- Which weights were most significant and which might have been pruned out to speed training?
- If NNs with initial weight values **init** and **-init** had been trained in parallel, would a comparison of their final values for weight w_{ijk} have indicated whether that $connection_{ijk}$ was significant?
- Would the NN-training concepts of *momentum* and *annealing* have contributed significantly?

- How would MLPs have performed if tested on position *value and depth* instead of just *value*?
- How successfully would the MLPs have played as White (evaluating btm-positions) or Black (evaluating wtm-positions) against perfect play, success being measured in terms of %-wins won and %-of draws drawn?
- Would non-MLP NN topologies be applicable to the chess-endgame problem domain?

7. THE NN CAPABILITY IN CONTEXT

Rather than considering NN technology in isolation, let us ask how the 'NN approach' compares with the use of other technologies and whether NNs can be included in a hybrid approach to chess endgames.

Computer agents provide a mix of definitive, statistical and unqualified advice. Second-sourced endgame databases and *optimal* or *correct* programs are in the first category; Buro's (1995) Othello position evaluations are in the second. The NNs here are in the last category as they do not define the likelihood that their output is correct.

Reference has already been made to previous work (Beal and Clarke, 1980; Bramer, 1980) where the authors first had to recognise key features of positions by observation or received wisdom and then determined the sequence of use of these characteristics in a program code. The use of induction automation (Hunt and Stone, 1966; Quinlan, 1979; Shapiro and Niblett, 1982) promised to assist the latter activity but in fact they produced only a few results on chess endgames. The cause may be: (1) suitable position features were not used, (2) they were not used in the right combination, or (3) they do not exist at all.

Neural networks are dependent on the input characterisation of the positions but are free – almost too free – to combine these characteristics as encouraged by the target outputs. This suggests an experiment with a chosen position characterisation to compare the ease of construction, the efficiency and the effectiveness of:

- programs using the rules included in the characterisation;
- agents produced by rule-induction processes;
- agents using evaluation functions derived from Bayesian classification as in Buro (1995);
- a variety of NNs with a range of parameters including their compression factor.

Perhaps categorical data analysis (Buro, 1995) can further be used to initialise a 1-level NN which after training can in turn seed a 2-level NN as mentioned at the end of Section 3. NNs also offer the promise that, when trained, their most significant internal weights will reveal higher-level characteristics which are combinations of the input features. These characteristics may then enrich the original characterisation and after a further round of training, the first cycle of an evolutionary inductive process is complete. Then, any position features highlighted by NNs may also be input to programs, rule-induction processes or domain experts.

However, inferring new characteristics from weights is not easy and the best strategy may be to isolate them one at a time in subdomains where they are most conspicuous. Given that, in the last stages of an endgame, the winning side has accumulated some advantages by achieving some preliminary goals, it seems logical to investigate these feature-rich subdomains of the endgame first and – in a second example of a gradual approach – 'reach back' into the endgame, perhaps using *move features* as well as *positional features*. For example with KPK, to win a position, White must achieve a subset of the following five objectives:

(1) defend the white Pawn from attack,
(2) pre-empt/loosen the black King's control of the Pawn's conversion square,
(3) clear a path for the white Pawn to run,
(4) run the Pawn to conversion to Queen or Rook, and
(5) mate the black King after the endgame conversion to KQK or KRK.

This suggests that an NN trained on the closed sets C_p or $C_p \cap F_x$ (see Section 2) for increasing p and varying x might pick up position features more easily one by one just as a human would. Other closed sets not used in this paper can be used to give a finer-grain approach to the full KPK domain.

There is however the danger that NNs will ascribe features to chess positions which are meaningful to NNs but not usable over the board by human players. Danger probably becomes likelihood if the input characterisation does not include the characteristics already known to the chess community.

Although this paper focuses on chess endgames, another natural question concerns the applicability of NNs in the opening phase of a game. Again there is a database of opening positions, compiled this time from play over the board. In the absence of definitive knowledge, each opening position may be assigned a 'probability' or 'likelihood' of winning based on past results. Domains with more smoothly varying probabilities are thought to be more amenable to NN modelling than endgames where the position value is +1, 0 or –1 (Tesauro, 1992b, 1995). Existing linear position evaluation functions can serve as the start point for at least 2-level MLPs performing non-linear evaluations.

Given all the above, this section suggests a program of data mining on chess endgames and perhaps other areas of chess which will combine the best features of the various approaches today.

8. SUMMARY

The study reported here was triggered by a student project to demonstrate neural-network learning from a suitable dataset. The KPK endgame was chosen and a number of NNs were trained and tested: the immediate goal was achieved. The project raised a number of issues, particularly about the characterisation of the positions and the approach to the design of the best NNs to represent the original endgame database. It also raised questions about the relative merits of NNs and other data-mining techniques and their possible use in combination.

We propose a programme analysing chess data, particularly endgame databases, by using the available range of data-mining techniques and technologies. The purpose of the programme would be to create higher-level information and knowledge from the base data in the correct endgame databases. New knowledge will include both heuristics of good play and those perverse positions which are the 'exceptions to the rule' and by their singularity deserve to be enshrined as problems and studies.

We hope that Nunn's expectation of "more human friendly endgame explanations" will be fulfilled but expect that chess will not be reduced to a 32-man endgame with a slim 'how to win' pamphlet.

9. ACKNOWLEDGEMENTS

First of course, our thanks goes to Ken Thompson for his database work and particularly for providing us with one of his endgame CDs which included KPK. Our thanks go also to Don Beal, Jaap van den Herik and Dennis Breuker who all encouraged us in this initiative.

The web made available Steven Edwards' databases and John Tamplin's access service to Ken Thompson's databases. Finally, our thanks go to the referees for their constructive comments on the original content and presentation.

10. REFERENCES

Beal, D.F. and Clarke, M.R.B. (1980). The Construction of Economical and Correct Algorithms for King and Pawn against King. *Advances in Computer Chess 2* (ed. M.R.B. Clarke), pp. 1-30. Edinburgh University Press, Edinburgh. ISBN 0-85224-377-4.

Beale, R. and Jackson, T. (1990). *Neural Computing: An Introduction*. Institute of Physics Publishing, Bristol. ISBN 0-8527-4262-2.

Berliner, H. (1980). Computer Backgammon. *Scientific American*, Vol. 242, No. 6, pp. 54-62. ISSN 0036-8733.

Bramer, M.A. (1980). An Optimal Algorithm for King and Pawn against King using Pattern Knowledge. *Advances in Computer Chess 2* (ed. M.R.B. Clarke), pp. 82-96. Edinburgh University Press, Edinburgh. ISBN 0-85224-377-4.

Bramer, M.A. (1982). Machine-Aided Refinement of Correct Strategies for the Endgame in Chess. *Advances in Computer Chess 3* (ed. M.R.B. Clarke), pp. 93-112, esp. Appendix I. Pergamon Press, Oxford. ISBN 0-08-026898-6.

Buro, M. (1995). Statistical Feature Combination for the Evaluation of Game Positions. *Journal of Artificial Intelligence Research*, Vol. 3, pp. 373-382. ISSN 1076-9757.

Clarke, M.R.B. (1977). A Quantitative Study of King and Pawn Against King. *Advances in Computer Chess 1* (ed. M.R.B. Clarke), pp. 108-118. Edinburgh University Press, Edinburgh. ISBN 0-85224-292-1.

Edwards, S.J. (1994). Endgame Databases of Optimal Play to Mate. At publication time, this was to be found on the World Wide Web with ftp://chess.onenet.net/pub/chess/TB/ ... KPK.tbs/tbb/tbw

Fürnkranz, J. (1996). Machine Learning in Computer Chess: the Next Generation. *ICCA Journal*, Vol. 19, No. 3, pp. 147-161.

Hinde, C.J., Fletcher G.P., West A.A. and Williams, D.J. (1997). Neural Networks. *ICL Technical Journal*, Vol. 11 No. 2, pp. 244-278, esp. p. 259. ISSN 0142-1557.

Hunt, E.B., Marin, J. and Stone, P. (1966). *Experiments in Induction*. Academic Press. LCC 65-26400.

Messom, C.H. (1992). *Engineering Reliable Neural Networks*. Ph.D. thesis, Loughborough University, UK.

Michie, D. (1980). Chess with Computers. *Interdisciplinary Science Reviews*. Vol. 5, No. 3, pp. 215-227. ISSN 0308-0188.

Minsky, M.L. and Papert, S.A. (1969). *Perceptrons, An Introduction to Computational Geometry*. MIT Press, Cambridge, MA. Expanded edition (1988), MIT Press, Cambridge, Mass. ISBN 0-262-63111-3.

Nunn, J. (1992). *Secrets of Rook Endings*. B.T. Batsford, London. ISBN 0-7134-7164-6.

Nunn, J. (1994). *Secrets of Pawnless Endings*. B.T. Batsford, London. ISBN 0-7134-7508-0.

Nunn, J. (1995). *Secrets of Minor-Piece Endings*. B.T. Batsford, London. ISBN 0-7134-7727-X.

Quinlan, J.R. (1979). Discovering Rules by Induction from Large Collections of Examples. *Expert Systems in the Micro-electronic Age.* (ed. D. Michie), pp. 168-201. Edinburgh University Press, Edinburgh. ISBN 0-85224-381-2.

Rich, E. and Knight, K. (1991). *Artificial Intelligence*. Second Edition. McGraw-Hill Book Company, New York, N.Y. ISBN 0-07-100894-2.

Roycroft, A.J. and Thompson, K.L. (1986). Queen and Pawn on a6 against Queen. *Roycroft's 5-Men Chess Endgame Series*, No. 2. Chess Endgame Consultants and Publishers, London.

Roycroft, A.J. and Beal, D.F. (1991). To Make Dumb Endgame Databases Speak. *Advances in Computer Chess 6* (ed. D.F. Beal), pp. 149-159. Ellis Horwood Ltd., Chichester, England. ISBN 0-13-006537-4.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1987). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1. MIT Press, Cambridge, Mass. ISBN 0-262-18120-7.
Samuel, A.L. (1959). Some Studies in Machine Learning using the Game of Checkers. *IBM Journal of Research and Development*, Vol. 3, No. 3, pp. 210-229. ISSN 0018-8646. Reprinted (1963) in *Computers and Thought* (eds. E.A. Feigenbaum and J. Feldman), pp. 71-105. McGraw-Hill Book Company, New York, N.Y. ISBN 07-020370-9.

Samuel, A.L. (1967). Some Studies in Machine Learning Using the Game of Checkers II Recent Progress. *IBM Journal of Research and Development*, Vol. 11, No. 6, pp. 601-617. ISSN 0018-8646. Reprinted (1970) in *Human and Artificial Intelligence* (ed. F.J. Crosson), pp. 81-116. Appleton-Century-Crofts, Educational Division, Meredith Corporation, New York, N.Y. ISBN 0-8919-7220-X.

Shapiro, A.D. and Niblett, T. (1982). Automatic Induction of Classification Rules for a Chess Endgame. *Advances in Computer Chess 3* (ed. M.R.B. Clarke), pp. 73-92. Pergamon Press, Oxford. ISBN 0-08-026898-6.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. (esp. Section 11.1). MIT Press, Cambridge, Mass. ISBN 0-2621-9398-1.

Tesauro, G. (1992a). Practical Issues in Temporal Difference Learning. *Machine Learning*, Vol. 8, pp. 257-277. ISSN 0885-6125.

Tesauro, G. (1992b). Temporal Difference Learning of Backgammon Strategy. *Machine Learning: Proceedings of the 9th International Conference on Machine Learning 92* (eds. D. Sleeman and P. Edwards), pp. 451-457. Morgan Kaufmann, San Mateo, CA. ISBN 1-55860-247-X.

Tesauro, G. (1994). TD-Gammon, a Self-Teaching Backgammon Program, achieves Master-Level Play. *Neural Computation*, Vol. 6, No. 2, pp. 215-219. ISSN 0899-7667.

Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, Vol. 38, No. 3, pp. 55-68. ISSN 0001-0782.

Thompson, K. (1986). Retrograde Analysis of Certain Endgames. *ICCA Journal*, Vol. 9, No. 3, pp. 131-139.

Walker, A.N. (1996). Hybrid Heuristic Search. *ICCA Journal*, Vol. 19, No. 1, pp. 17-23.

POST-PUBLICATION NOTE

This version corrects a few slips in or clarifies the content of the published version as follows:

- p212, Table 1, Set 5: 'Kb4/Kc5 are sub-optimal' for 'Kc5 (sub-opt.) or Kb4?' – see correction 4;
- p213, para 1, last line: 'P_q' for 'P_7';
- p213, para 2, line 3: '$W(C_9)$' for 'W(9)';
- p213, line –3: 'Kb4/Kc5 are sub-optimal' for 'Kb4 only draws' – Kb4 still wins;
- p214, Figure 1: '$w_{a10}.x_{a0}$' for '$w_{ab0}.x_{a0}$' on the arrow from N_{a0} to N_{11};
- p215, para 3, line 3: 'represent at most one of' for 'represent one of';
- p215, para 5: 'linear form' for 'linear equation';
- p216, Figure 2: the connector from c_{24} to c_{51} replaces that from c_{22} to c_{51} – see next correction;
- p217, Table 4, last row (c_{51}): 'x_{18}' for 'x_{16}' – the check is that the wK and the wP are on different files.