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

Training Reformulated Product Units in Hybrid Neural Networks

Philip T. Elliott, Diven Topiwala, and Will N. Browne

Abstract— **Higher order networks allow modelling of correlates and geometrically invariant properties. Current techniques for their development either require domain knowledge, or are constrained by scaling properties or local minima. A novel reformulation of the product unit is introduced, motivated by a desire to improve scaling and training properties. The new unit allows developing high orders of positive and negative powers, and correlates in a single stage, but can be trained successfully using standard back propagation techniques. Tests on standard benchmarks in various hybrid topologies demonstrate the potential in a variety of problem domains.**

I. Introduction

Artificial Neural Networks (ANN) are designed to emulate the storage and learning mechanisms within biological brains. The standard ANN model is based upon summation, calculating the net input as the weighted sum of the inputs. Multi-layer Summation Unit Neural Networks (SUNN) have been found capable of representing any continuous function to an arbitrary degree of accuracy, provided there are sufficient number of hidden units [1-3]. However, biological evidence shows the capability of the animal nervous system to perform multiplication [4] as well as addition.

Multiplication in ANN allows for the development of correlation information and higher order terms providing increased capacity and the ability to learn geometrically invariant properties [5]. Appropriate domain knowledge enables these terms to be pre-calculated and included as additional inputs to a SUNN. Without this knowledge, all permutations of inputs and powers must be included, which results in a combinatorial explosion for large numbers of inputs. It has been shown that for logical problems at least, generally only a minority of variables require higher order terms [6]. Therefore, except for low dimensional problems, or where domain knowledge is available, it is considered most efficient to develop the necessary higher orders internally during training [7-12].

An alternative form is the pi-sigma NN [7], which takes into account the requirement for developing only a minority of higher order terms. A pi-sigma network consisting of K summing units feeding a single output product unit can produce a K^{th} order approximation of a continuous function. Some other architectures which allow development of higher orders and/or correlates include the functional link NN [13], ridge polynomial NN [14], and Product Units (PU) [11].

Out of which Product Units are the most efficient for developing higher orders, however the standard method of training NN, Gradient Descent (GD), which has proved successful in training SUNN tends to become stuck in local optima when training PUNN. The reason for this is the PU increase the complexity of the state space, introducing more local minima [8, 10, 15].

The focus of this paper is the development of a re-formulation of the product unit. The aim is to allow development of arbitrary orders of power, both positive and negative, within a single unit, facilitating description of system poles and zeros. A further aim is to harmonise this novel Reformulated Product Unit (RPU) for operation with the Summation Unit (SU) while minimising complexities introduced into the state space, and hence improve trainability. The RPU when instantiated in its minimal form will use the same parameter order, back propagation and weight update equations as a SU. These equivalences simplify the implementation of hybrid networks. Units are easily interchangeable allowing development of arbitrary topologies. The optimal topology to approximate a given function is postulated to utilise multiple forms of data fusion to produce a single output from the many input variables. Or simply stated, a function can be more accurately and compactly modelled if the relationships between the variables can be succinctly emulated, which requires a variety of neural tools to be available.

Using multiple neural blocks in order to further minimise the number of parameters in a model is

Manuscript received February 15, 2005. This work was supported in part by the UK Government's Engineering and Physical Sciences Research Council (EPSRC), and also by the Thales Group UK.

P. Elliott with Cybernetics, University of Reading, RG6 6AY, UK, (phone: +44 (0) 118 987 5123; e-mail: p.t.elliott@rdg.ac.uk).

D. Topiwala is with Thales Research & Technology (UK) ltd, RG2 0SB, (phone: +44 (0) 118 923 8276; e-mail: diven.topiwala@thalesgroup.com).

W. N. Browne with Cybernetics, University of Reading, RG6 6AY, UK, (phone: +44 (0) 118 378 6705; e-mail: w.n.browne@rdg.ac.uk).

0-7803-9490-9/06/$20.00/©2006 IEEE

commensurate the principle of Minimum Decription Length (MDL). Jorma Rissanen introduced the principle of MDL [16], which embodies Ocam's Razor. The fundamental concept is that any regularities within the data may be used to compress its form facilitating more compact descriptions, requiring fewer symbols, which tends to optimise average generalisation performance on unseen data.

The following sections of this paper will provide background on previous forms of multiplicative data fusion. Associated training difficulties are discussed along with potential means to minimise local minima for the standard PU. A new form of product unit is then introduced to address important issues and tested on benchmark problem data sets.

II. Multiplicative Data Fusion in Neural Networks

The standard Summation Unit (SU) fuses multiple input signals to form a single net input signal using a weighted sum of the inputs plus a bias, (1), where o_j is the output of unit j, and w_{ij}, is an adjustable parameter between node i from a preceding layer and node j, and x_i represents the output value of node i. The final output of the node is then the net input passed through an activation function (2). Typically a sigmoid, such as the tan-sigmoid (3), is used but cosine, gaussian or any other smooth differentiable function may be used with back propagation. There are two primary functions performed by the sigmoid: firstly, to act as a squashing function to ensure that signals stay within controllable limits, since large-scale differences in signals tend to be detrimental to learning [12, 17]. Secondly, the activation function acts as an optional source of non-linearity that may be used where the linear net input is incapable of modelling the required functional relationships. The standard sigmoid activation function is linear around the unit's centre but increasingly non-linear towards the extremes.

$$(SU)\ net_j = \sum_{i=1}^{m}\left(w_{ij}.x_i\right) + w_0 \tag{1}$$

$$o_j = \sigma_j\left(net_j\right) \tag{2}$$

$$\sigma_j\left(net_j\right) = \frac{2}{1+\exp\left(-net_j\right)} - 1 \tag{3}$$

Driven by biological evidence of multiplicative processing in the nervous system, and by a pure engineering and mathematical desire to diversify the range of functions that can be represented and learned by Neural Networks, multiplicative data fusion has been utilised in NN in a variety of different forms. The most established form of multiplicative synapse is Higher Order Neuron (HON). The equation that defines a HON with output o, and inputs x_a, x_b, x_c … is defined in (4)

$$(\text{HON})\ o = \sigma\left(\begin{array}{l} w_0 + \sum_a w_a x_a + \sum_{a,b(a\le b)} w_{ab} x_a x_b \\ + \sum_{a,b,c(a\le b\le c)} w_{abc} x_a x_b x_c \end{array}\right) \tag{4}$$

The higher-order correlations contained allow the learning of geometrically invariant properties, and also allows the pre-calculation of the higher-order correlations as additional inputs so that a SUNN can be trained in the normal way for fast learning of low dimensional problems. However, due to the exponential scaling property of this formulation, it is limited to low orders of power and low dimensional problems.

Sigma-Pi architectures, so named due to their form that computes the sum of products, alleviate this scaling problem somewhat, however development of high orders is still very computationally expensive. The sigma unit uses the standard weighted sum as in (1), to compute the net input, which is then passed through an activation function σ, (2). The pi unit (PIU) calculates the net input as the product of weighted inputs (5).

$$(PIU)\ net_j = \prod_{i=1}^{m}\left(w_{ij}.x_i\right) \tag{5}$$

The PIU as defined in (5), [18, 19], is not a scaleable or controllable form with which to develop higher orders or correlates. The model can be factorised retaining identical in functionality to give (6), where a single common factor w_j represents the combined effect of the w_{ij}. Therefore, inputs cannot be individually controlled to increase or discount their effects.

$$(PIU)\ net_j = w_j.\prod_{i=1}^{m}\left(x_i\right) \tag{6}$$

Pi-sigma architectures [7] compute the product of sums rather than sum of products. The parameter order is reduced for this architecture by removing the adaptable weights connecting the sigma unit's outputs to the pi units. Pi-sigma networks reduce the scaling problems with the number of inputs present in sigma-pi networks. However to generate a cubic term, three summation units are required to act solely as through paths from the inputs to the outputs, with all other SU links being redundant. Pi-sigma networks are considered to be efficient for non-linear pattern classification and function approximation.

All of the above forms are limited in the maximum order that they can generate for a single variable. The majority, with exception of pi-sigma networks can only develop first order powers or correlates in a single layer, and the maximum order of pi-sigma is limited by the available number of sigma units, with increasing

parameter redundancy in the sigma units for development of higher orders.

Another form of multiplicative based data fusion, the Product Unit (PU) uses a power function to control the strength of propagation for each link rather than a linear scaling weight. The advantage of this formulation (7) is that it allows the development of arbitrary order of power for any input within a single unit. Correlates between the inputs raised to their separate powers can also be produced. Product Unit Neural Networks (PUNN) where developed by Durbin and Rumelhart [11], and have been further investigated by others including [8, 9, 15].

$$(PU)\ net_j = \prod_{i=1}^{m} \left(x_i^{w_{ij}} \right) \tag{7}$$

An additional advantage of product units is their increased information capacity of 3N relative to the SU capacity of 2N for N inputs, for binary logic problems at least [11] .It is worth noting that for Boolean inputs the PU has been shown to be approximated by a SU with a cosine activation function (8).

$$iff\ \left(x_i \subset \{-1,1\} \text{ are remapped onto } \{1,0\} \right)$$
$$(PU)\ net_j = \cos \pi \sum_{i=1}^{m} \left(w_{ij} . x_i \right) \tag{8}$$

This equivalence simplifies the implementation of the PU for binary inputs. Stemming from this form, approximating the PU as a SU with a cosine activation function, is the suggestion that the Vapnik-Chervonenkis (VC) dimension is practically infinite, for binary inputs at least [15, 20]. The VC dimension directly relates partly to the versatility but also to the generalisation properties. A large VC dimension implies that generalisation may not be guaranteed; although it is not clear to what extent this comparison extends to a real non-binary implementation.

PU are powerful in their representative abilities, but also tend to be more difficult to train than SU. The reason for this is that the PU introduces increased amounts of local minima into the state space along with deep ravines and valleys that tend to trap the solution [8-10, 15]. An important cause of local minima introduction is due to the power function being used to control the strength with which signals are propagated. In (7) if the base argument x_i of the power is zero then the contribution for the input must be either zero or infinity for negative and non-negative powers respectively, not the desired unit identity. More simply stated, even with a controlling power of zero an input cannot be ignored if x_i touches or crosses zero. An integral aspect of scaleable learning is the ability to completely discount any inputs unrelated to the target output. Use of either standard form of input (x_i) representation, uni-polar [0..1] or bi-polar [-1..1] will result in the introduction of artefacts.

The use of bi-polar inputs with the standard PU equation (7) will result in further local minima due to the inclusion of sign affects within the controlling power operator. The power function preserves input sign information for negative powers, but for positive powers sign information is removed. This can make a purely local search for global solutions almost impossible; consider modelling z^3 as in [8]. Training from small random initial conditions, as is normal with SUNN, will increase the controlling weight towards unit power. However, due to the incorporation of sign in the power, training will not surmount the large error introduced where the sign information is lost at the power of two in order to reach the cubed power where sign information is restored. The same problem would arise in attempting to learn the problem with power initial condition greater than four. An alternative suggestion to reduce local minima is to use a signed power, where sign information is maintained regardless of the controlling power.

In summary, the requirements for an adaptable learning element to generate higher order correlates are scalability, trainability and inclusion of the combining operator's identity for each input. Scaling to accommodate large numbers of inputs requires the ability to ignore or only partially consider signals, many of which may be unrelated to the output. This feature is notably lacking from (5). The PU (7) allows partially discounting inputs, but with the production of artefacts being detrimental to trainability and accuracy in representing continuous functions. Trainability is increasingly important with network size to make problems analytically tractable. Evolutionary algorithms tend to produce reasonable solutions quickly, but often take much longer to tune; making gradient descent a desirable feature for a practical solution. If zeros and zero crossing are to be utilised it is imperative that the location of the zero point on the input space is adaptable to allow discounting signals.

III. Reformulated Product Unit (RPU)

What is desired is a means of adaptively learning to develop the required higher order terms to model a given function for minimal cost. Since only a few higher order terms are generally required to solve a problem it is desirable to allow generation of arbitrary powers in a single stage to minimise the parameter cost. The product unit introduced by Durbin and Rumelhart, [11], allows this. However, in order to minimise the introduction of minima into the state space by the inclusion of higher order variables, and improve trainability, the product unit will be reformulated. A second priority in this

reformulation is to harmonise the product unit with the standard summation unit to allow back propagating hybrid networks. Another factor to consider is that if multiple different types of neurons are to operate simultaneously in a network, then there is an additional performance cost in determining and utilising the corresponding specialised error and weight update formulae. Ideally the same equations would be effective for each node type.

The standard SU conventions assumed here are:

net_j is the net input for node j which has a total of n units in its layer. Unit i is in the preceding layer to unit j and has a total of m units in its layer. The input for unit i is denoted by x_i and has range [-1 …1], w_{ij} represents an adjustable signal propagation strength parameter between unit i and unit j. The error for unit j is e_j, where for output nodes equation (9) is used and for hidden nodes, where target information is not present, the errors are back propagated according to the strengths of the connecting link parameters w_{ij} using (10). The equation controlling weight adaptation is given by (11), where the learning rate is a network parameter controlling step size.

$$e_j = \sigma_j'\left(net_j\right).\left(t_j - o_j\right) \tag{9}$$

$$e_i = \sigma_i'\left(net_i\right).\left(\sum_{j=1}^{n}\left(\frac{w_{ij}.e_j}{n}\right)\right) \tag{10}$$

$$\Delta w_{ij} = x_i.e_j.\eta \tag{11}$$

To allow the use of the standard SU back propagation equations it is essential to align the identities and orientation of all of the operators. The binary combining function used is a product so requires unity identity for discounting of signals. The base argument in the power function also has unity identity and the exponent argument acts as the controllable strength of signal propagation. The standard power function is undefined for negative inputs so the base of the power must remain positive, otherwise a complex power or signed power must be used. Where a signed power would have the advantage of reduced numbers of local minima.

$$(RPU)\ net_j = \prod_{i=1}^{m}\left(\left(1 + c_j\beta(x_i)\right)^{w_{ij}}\right) - 1 + w_0 \tag{12}$$

The summation-based identity at zero on the input space can then be mapped to unity identity for the base argument in the power function as in (12). The simplest form of (12) would be to use a global constant scaling parameter c, where (c<1) to ensure that the base of the power remains positive with $\beta(x_i)$ as the raw input (13). This helps avoid exceptions by "softening" the zero effect, preserving accountability by avoiding saturation, and also allows smooth discounting of signals which is important for scalability with large numbers of inputs,

$$\beta_i\left(x_i\right) = x_i \tag{13}$$

This format maintains the integrity of all the standard SU formulae with only the net input equation specialised. Finally, after the links input factors have been multiplied together, to reconcile the differences in the operators' identities and convert back to standard summation based form, a unit bias is removed. For minimal extra cost the scaling parameter c can be individual to each node, to allow increased flexibility.

A descriptive aspect that has been removed from the original PU during its reformation is the ability to develop 'V' or 'U' shapes from bi-polar inputs; bi-polar inputs with even power will form a 'U' shape centred at zero. An extension of the RPU form is to allow development of such local features in a manner similar to radial basis functions, [20]. Equation (12), used with (14), allows description of local features internal to the input space, for example the absolute distances from a feature centre. The minimum additional cost to describe such features within this formulation is a single additional parameter per node to describe a single common feature centre in input space (14). However, where multiple internal features are anticipated significantly increased expressive capabilities can be achieved through using a feature centre parameter per link (15).

$$\beta_i\left(x_i\right) = \left(abs\left(x_i - b_j\right) - abs\left(b_j\right)\right) \tag{14}$$

TABLE I
DISPARITY IN SIGNAL SHAPES AND RANGES BETWEEN POSITIVE AND NEGATIVE CONTROLLING POWERS

Scalar const (c)	Disparity in range between ±1 power	Unit Positive Power			Unit Negative Power		
		$x_i = -1$	$x_i = +1$	Range	$x_i = -1$	$x_i = +1$	Range
0.100	0.002	0.900	1.111	0.100	0.002	0.900	1.111
0.300	0.059	0.700	1.300	0.300	0.059	0.700	1.300
0.500	0.333	0.500	1.500	0.500	0.333	0.500	1.500
0.700	1.345	0.300	1.700	0.700	1.345	0.300	1.700
0.900	7.674	0.100	1.900	0.900	7.674	0.100	1.900
0.990	97.499	0.010	1.990	0.990	97.499	0.010	1.990
1.000	∞	0.000	2.000	1.000	∞	0.000	2.000

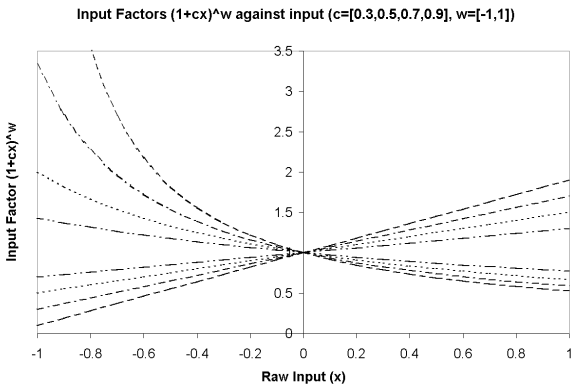

Fig. 1. Input Factors against Inputs for (c=[0.3,0.5,0.7,0.9], w=[-1,1])

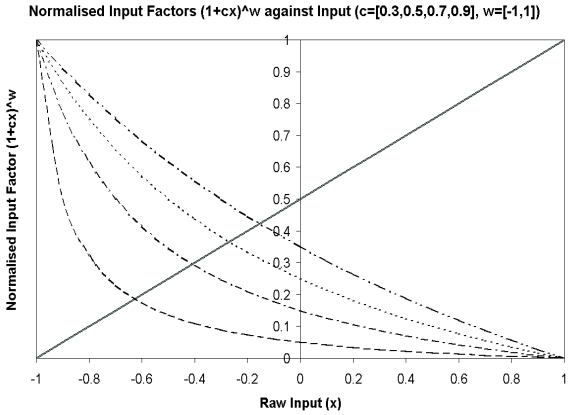

Fig. 2. Normalised Input Factors against Inputs to illustrate increasing shape disparity as c approaches 1. for (c=[0.3,0.5,0.7,0.9], w=[-1,1])

$$\beta_i(x_i) = \left(abs\left(x_i - b_{ij}\right) - abs\left(b_{ij}\right)\right) \quad (15)$$

Inclusion of features internal to the input space, using equation (14) or (15) instead of (13), requires specialising the update equation (11) for the (RPU). Where internal features are used the weight update for links should be amended to (16).

$$\Delta w_{ij} = \beta(x_i).e_j.\eta \quad (16)$$

An important point regarding the use of a power as a strength controller for signal propagation is the operator bias. Inverting the sign of a standard weight is essentially the same as inverting the sign of the input. However, for powers this is not the case as there is a disparity between produced signal shapes and ranges for positive and negative values of power. This disparity is illustrated numerically in Table I, and the same information graphically in Figure 1. The increasing non-linearity as c approaches 1 while using negative powers is most clearly evident in Figure 2; note that the output of the power based signal control is linear for all tested values of c with unit positive power, resulting in the normalised plots being co-incident, however for negative unit power as c approaches 1 the dominance of system poles over zeros causes increasing non-linearity. This disparity increases with the magnitude of the controlling power.

To allow back propagation of the errors from the output units to hidden units as described in (10) an estimated error is formed based on the average error passed back through the network. Similarly to forward propagation of signals; the back propagation of errors uses the connecting link's signal propagation strength to scale the magnitude and sign of the error passed to previous nodes. If a power function is used to control strength of propagation then any disparity in the shape and ranges between positive and negative powers will adversely affect the calculated error for preceding units. Biased or erroneous errors will tend to have adverse affects upon training, potentially even resulting in training being detrimental to performance. It is worth noting that the adverse effect upon trainability of previous units, which increases as c approaches 1, appeared stronger for a preceding SU than RPU.

It is postulated that the majority of optimal ANN models of problems will contain multiple different forms of data fusion; large fixed values of c should generally be avoided for the sake of compatibility. The optimal value of c will be problem dependant and ideally should be a node or link level variable rather than network level.

Comparison between the PU and the RPU shows many similarities. Both units are capable of generating multiple positive powers of an arbitrary order, allowing production of higher order correlates in a single stage, for the same parameter order as a SU. The forms diverge on their zero placement, the PU includes a zero in the input space by default, which is detrimental to scalability and trainability as signals cannot be entirely ignored. The RPU does not place a zero internal to the input space by default and so has better scaling properties for larger numbers of inputs. The RPU can develop reciprocals and higher order negative powers to encompass a commutative form of division. The PU can model combinatorial affects of interacting signs for bi-polar inputs, although in a very limited way since the zero location is fixed. The PU has associated training difficulties, which are minimised by the RPU form. An implementation benefit of the basic RPU form is its harmony with SU conventions providing the ability to use standard SU based training techniques.

IV. Hybrid Neural Network Topologies

The traditional approach to network topology development is the use of a single type of element in layer-based structures where every element connects only to elements in its preceding and subsequent layer. A natural

extension to this approach, when using multiple different types of processing element, is to have layers being composed of different types of elements. This leads to the development of 'sum of product' and 'product of sums' network architectures. Since two types of processing element are to be used, test will be performed for the four possible permutations while maintaining homogeneity in types for the layers. The four forms to be used are the product of sums (SU:PU), sum of products (PU:SU), product of products (PU:PU), and the standard sum of sums (SU:SU). All network architectures are utilising a single hidden layer. Additionally a flexible architecture will be tested where the homogeneity of neural types per layer constraint is removed.

V. Evolutionary Back Propagation Algorithm

The algorithm used for training is a simple Evolutionary Algorithm (EA), combined with back propagation. The purpose of the EA is twofold; firstly to allow optimisation of the RPU c_j variable, which is not trained through back-propagation, and secondly as a more efficient means of exploring multiple initial conditions. The EA is (9+9)-ES +1elite with networks failing selection being re-initialised, which ensures genetic diversity in order to prevent stagnation. Each generation the population is filled to the desired size ($p=20$) by generating networks with random initial condition for their weights. The weight range used for initial conditions was –1 to +1 for all weights except the RPU c_j variable, which was given a fixed range between 0.05 and 0.8. The RPU units were implemented using (12) and (14). Networks were then trained using back propagation for a maximum of 50 epochs. If the network performance degraded from its local optimum for two epochs training was ceased for computational efficiency and then the network was returned to its prior optimum. To promote rapid convergence into the correct region prior to fine-tuning a short annealing schedule was used (17) where $\eta(0)=0.2$, t is the networks current epoch and $T = 4$.

$$\eta(t) = \frac{\eta(0)}{1 + t/T} \tag{17}$$

Selection is performed using a binary tournament; the winner of each tournament replicates itself with weight mutations. The mutation rate used was 0.15. The best network from each generation is immortalised passing through to the next generation unmodified to continue its annealing schedule. A maximum of 50 generations are allowed for each problem.

VI. Experimental Results

In this section the RPU defined by (12) and (14) will be utilised in a variety of topologies to investigate its representational and training properties. Four standard classification problems taken from, [22], will be used to allow comparison of representational performance for binary classification against standard techniques. Additionally a new modelling problem, involving modelling the net torque acting on a physical system, is introduced to help analyse performance in modelling continuous outputs for a problem with a known solution. All results are for ten repetitions of stratified ten fold cross validation on the whole pattern set. Average classification accuracy and the standard deviation are presented for both the training folds and unseen testing folds allowing a measurement of generalisation performance. The classification benchmark C4.5 Tree-Construction Algorithm [23] performance is included, where available, along with the best performances found from other classification literature.

A. Iris

TABLE 2
Iris Ten Fold Cross Validation Results

IRI (I4 O3)	Train		Test	
Topology	Class	Std	Class	Std
3-SU:RPU	98.75	0.51	97.15	4.45
3-RPU:SU	98.74	0.51	97.09	4.45
3-RPU:RPU	98.85	0.52	96.36	4.81
3-SU:SU	98.72	0.51	97.33	4.33
C4.5			95.3	3.2
Best Lit			99.3	1.9

The iris data set is a simple well-known non-linearly separable data set; the problem is to correctly classify the type of iris based upon petal and sepal dimensions. The set has 4 continuous numerical inputs and 3 potential classes. The results for each of the 4 homogenous layer topologies are very similar, all of which outperformed C4.5 on cross validation. The largest deviation from the norm is for the PU:PU topology, which relies on multiplication to fuse all inputs.

B. Diabetes

TABLE 3
Diabetes Ten Fold Cross Validation Results

DIA (I8 O1)	Train		Test	
Topology	Class	Std	Class	Std
3SU:RPU	79.44	0.95	75.68	5.34
3-RPU:SU	78.18	1.02	75.69	4.45
3-RPU:RPU	78.80	1.11	76.06	4.64
3-SU:SU	79.81	0.98	75.10	4.89
2-SU:RPU2	78.77	0.95	76.50	4.83
1MX:MX*	78.46	0.71	76.90	4.48
C4.5			73/71.6	
Best Lit			76.2	4.8

Similar performance is achieved by each of the 4 fixed topology variants. Of which the best performance is

through the RPU:RPU architecture, which is not significantly different to the best reported results, while the worst performance is from the standard SU:SU form. All topology variants outperform C4.5. Optimal free topology performance (*) was achieved by using a single hidden node in a dense network, where the output was given direct connections to the inputs and the hidden node.

C. Balance Scale

TABLE 4
BALANCE SCALE TEN FOLD CROSS VALIDATION RESULTS

BSC(I4:O3)	Train		Test	
Topology	Class	Std	Class	Std
2-SU:RPU	91.67	0.36	91.77	3.20
2-RPU:SU	99.38	0.49	99.16	1.36
2-RPU:RPU	99.49	0.50	98.84	1.48
2-SU:SU	91.67	0.35	91.74	3.17
2MIX:MIX	99.45	0.44	98.91	1.40
Best Lit			91.4	1.3

The balance scale problem shows much more significant deviations in the performance of the different topologies.

The best performance is achieved by RPU:SU form, achieving more than 99% generalisation accuracy, with RPU:RPU having very similar performance. Both of these forms outperform all other know performances on this problem. The standard SU:SU form and the SU:PU form have significantly worse performance and also have larger standard deviation.

D. Torque

TABLE 5
TORQUE TEN FOLD CROSS VALIDATION RESULTS

TOR(I4:O1)	Train		Test	
Topology	Class	Std	Class	Std
2SU:RPU	5.67E-3	8.62E-5	5.85E-3	7.45E-4
2RPU:SU	8.48E-3	5.38E-3	8.74E-3	6.00E-3
2RPU:RPU	3.39E-3	1.24E-3	3.52E-3	1.42E-3
2SU:SU	5.62E-3	8.41E-5	5.79E-3	7.65E-4

The problem is to calculate the resultant torque about a fixed point with 2 opposing forces acting at different lever lengths from the pivot. This is an interesting problem since it is firstly a real physically embedded problem and also because it has a known solution. The resultant torque for an applied force (f) is the force multiplied by the lever distance (d) measured perpendicular to the force. The problem has 1000 patterns each with 4 uniformly random real values. The single output is the net torque (τ).

$$\tau = f_1.d_1 - f_2.d_2 \tag{18}$$

The best network performance, perhaps surprisingly given the inherent sum of product structure of the problem, is the RPU:RPU architecture with just over half the error of the next best architecture.

E. Glass

TABLE 6
GLASS TEN FOLD CROSS VALIDATION RESULTS

GLA(I9:O6)	Train		Test	
Topology	Class	Std	Class	Std
9SU:RPU	88.83	2.01	68.86	9.88
9RPU:SU	79.71	3.67	64.83	10.51
9RPU:RPU	81.13	3.52	66.03	10.72
9SU:SU	89.61	1.94	68.75	9.74
C4.5			68.5	10.4
Best Lit			71.5	1.9

The forensic analysis of glass problem was tackled almost equally well by both the SU:RPU and SU:SU forms, which both slightly outperform C4.5 on cross validation.

VII. DISCUSSION

Two different types of neural blocks have been utilised to develop hybrid neural networks. Investigations in topology have focussed on the four permutations with homogenous node types in each layer and standard adjacent layer based connectivity using a single hidden layer. Equal network size was used for each topology providing roughly equivalent description length. Each of the four constrained topology permutations demonstrated capacity to model the tested problems, despite network sizes not being optimised specifically for each topology variant. Some problems showed much greater sensitivity to the data fusion strategies used with significant differences in attained performance for equal network size. It seems clear that no single topological permutation is universally optimal, suggesting the need to automate the discovery of the optimal data fusion strategy topology through inheritance within the EA.

Initial investigations allowing a mixture of neuron types within any layer showed increased expressivities, requiring smaller network sizes to avoid over-fitting the training data.

The significance of the data fusion strategy topology is problem dependant and is related to the completeness with which the example patterns describe the system. The fewer the number of example patterns that are available for training the greater the significance of the topology in avoiding over-fitting the training data in order to improve generalisation accuracy.

VIII. CONCLUSION

Firstly, It should also be noted that based on small sample runs the single common internal absolute feature did not seem to offer performance advantages for the tested problems. The somewhat simpler form (13) could therefore be substituted for (14) to leave the weight update equations the same as for standard SU (11).

For the majority of the problems the network performances are quite close together regardless of the topology that was used. However for some of the problems more pronounced differences in performance were found. The optimal set of neural elements and their topology are problem dependant. Many problems can be approximated to similar performance levels by a variety of different architectures. However, some problems show significant improvements in compactness of representation and in generalisation performance through using a selection of different types of neural elements. Problems most suited to modelling with RPU are problems with large degrees of direct and/or inverse proportionality of arbitrary order. Systems that are amenable to description in terms of interactions of poles and zeros are recommended for modelling by RPUs.

IX. Further Work

This paper has focussed on modelling a number of real world classification problems. However the most obviously appropriate domains for the RPU are the accurate modelling and generalisation of non-linear continuous functions such as modelling of physical system transfer functions or financial markets.

This work has focussed on the scaleable adaptable modelling of systems towards the affects of poles and zeros. Due to this and the inclusion of inverse proportionality zero crossing has been omitted to avoid generating infinities. It is thought desirable to accommodate the description and crossing of both zeros and poles in order to develop trainable transfer function networks.

$$(RPU)\ net_j = q\prod_{i=1}^{m}\left(\left(-z + c\beta(x_i)\right)^p\right) + w_0 \qquad (19)$$

Further work will proceed to include zero crossing, whilst ensuring that no input has a zero forced upon its input space to avoid degrading scalability and trainability. The expanded parameter form (19) will be explored further. Note the power function used is the signed power of the absolute of the variable to avoid creating local minima. Investigation will proceed by varying the level of the *z*, *c*, and *p* parameters from network constants, through node level and down to individual link level. The resultant modelling flexibility and trainability will then be evaluated.

This work has concentrated solely on use of a single hidden layer with standard layer based connectivity. Further work will include topological evolution in order to allow development of deeper sparse structures.

Further research should include Ensemble development from networks composed of different neural components.

Acknowledgment

P. Elliott offers many thanks to D. Topiwala for his time patience and understanding, and TRT UK for the resources made available. Eternal thanks also to W. Browne for his virtuous guidance throughout times of adversity.

References

[1] Funahashi, A.K., *On the approximate realization of continuous mappings by neural networks.* Neural Netw., 1989. **2**: p. 183-192.

[2] K. Hornik, M.S., *Multilayer feedforward networks are universal approximators*, in *Artificial Neural Networks*, H. White, Editor. 1989, Blackwell: Oxford. p. 13-28.

[3] K. Hornik, M.S., H. White, *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks.* Neural Netw., 1990. **3**: p. 551-560.

[4] C. Koch, a.T.P., *Multiplying with synapses and neurons*, in *Single neuron computation*. 1992, Academic Press Professional, Inc. p. 315-345.

[5] C.L Giles, a.T.M., *Learning, invariance, and generalization in high-order neural networks.* Applied Optics, 1987. **26**: p. 4972-4978.

[6] N. J. Redding, A.K., and T Downs, *Constructive Higher-Order Network Algorithm that is Polynomial in Time.* Neural Networks, 1993. **6**: p. 997-1010.

[7] Y. Shin, a.J.G. *The Pi-sigma Network : An Efficient Higher-Order Neural Network for Pattern Classification and Function Approximation*. in *Proceedings of IJCNN*. 1991. Seattle.

[8] A Ismail, a.A.E. *Global Optimization Algorithms for Training Product Unit Neural Networks*. in *International joint conference on neural networks ijcnn'2000*. 2000. Como, Italy: IEEE.

[9] D.J. Janson, a.J.F.F., *Training Product unit Neural Networks with Genetic Algorithms*, in *IEEE Expert Magazine*. 1993. p. 26-33.

[10] L.R. Leerink, C.L.G., B.G. Horne, and M.A. Jabri, *Learning with Product Units.* Advances in Neural Information Processing Systems, 1995. **7**: p. 537.

[11] R. Durbin, a.D.R., *Product Units: A Computationally Powerfull and Biologically Plausible Extension to Backpropagation Networks*. Neural Computation, 1989. **1**: p. 133-142.

[12] C. Lin, K.W., J Wang. *Scale Equalized Higher-order Neural Networks*. in *Conference on Systems, Man and Cybernetics*. 2005. Waikoloa, Hawaii: IEEE.

[13] A. Hussain, J.J.S., T.S. Durbani, *A New Neural Network for Nonlinear Time-Series Modelling*. Neurovest Journal, 1997: p. 16-26.

[14] Y. Shin, a.J.G. *Approximation of Multivariate Functions Using Ridge Polynomial Networks*. in *JCNN*. 1992. Baltimore.

[15] Schmitt, M., *On the complexity of computing and learning with multiplicative neural networks.* Neural Computation, 2002. **14**(2): p. 241-301.

[16] J. Rissanen. (1978). *Modeling by the shortest data description.* Automatica 14, 465-471

[17] J. Wang, K.W., F. Chang. *Scale Equalisation Higher-order Neural Networks*. 2004: IEEE.

[18] B. Zhang, P.O., H. Muhlenbein, *Evolutionary Induction of Sparse Neural Trees.* Evolutionary Computation, 1997. **5**(2): p. 213-236.

[19] D. Li, K.H., J. Hu, and J. Murata, *Training a kind of hybrid universal learning networks withclassification problems.* Neural Networks, 2002. **1**: p. 703-708.

[20] A. Blumer, A.E., D. Haussler, and M. Warmuth, *Learnability and the vapnik-chervonenkis dimension.* Journal of the association for computer machinery, 1989. **36**(4): p. 929-965.

[21] Reyneri, L.M., *Weighted Radial Basis Functions for Improved Pattern Recognition and Signal Processing*. 1994, University of Pisa: Pisa. p.6.

[22] Newman, D.J.H., S. & Blake, C.L. & Merz, C.J., *UCI Repository of machine learning databases*. 1998, University of California, Irvine, Dept. of Information and Computer Sciences.

[23] J.R Quinlan, C4.5: Programs for Machine Learning (Morgan Kaufmnan, San mateo,, CA 1992