Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 71 (2008) 1477–1499
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Third-order generalization: A new approach to categorizing higher-order generalization

Richard Neville

School of Computer Science, University of Manchester, Kilburn Building, Oxford Road, Manchester M13 9PL, UK

Received 9 September 2005; received in revised form 22 March 2007; accepted 14 May 2007
Communicated by M. Magdon-Ismail
Available online 12 June 2007

Abstract

Generalization, in its most basic form, is an artificial neural network's (ANN's) ability to automatically classify data that were not seen during training. This paper presents a framework in which generalization in ANNs is quantified and different types of generalization are viewed as orders. The ordering of generalization is a means of categorizing different behaviours. These orders enable generalization to be evaluated in a detailed and systematic way. The approach used is based on existing definitions which are augmented in this paper. The generalization framework is a hierarchy of categories which directly aligns an ANN's ability to perform table look-up, interpolation, extrapolation, and hyper-extrapolation tasks.

The framework is empirically validated. Validation is undertaken with three different types of regression task: (1) a one-to-one (o–o) task, $f(x)$:$x_i \rightarrow y_j$; (2) the second, in its $f(x)$:$\{x_i, x_{i+1}, \ldots\} \rightarrow y_j$ formulation, maps a many-to-one (m–o) task; and (3) the third $f(x)$:$x_i \rightarrow \{y_j, y_{j+1}, \ldots\}$ a one-to-many (o–m) task. The first and second are assigned to feedforward nets, while the third, due to its complexity, is assigned to a recurrent neural net.

Throughout the empirical work, higher-order generalization is validated with reference to the ability of a net to perform symmetrically related or isomorphic functions generated using symmetric transformations (STs) of a net's weights. The transformed weights of a base net (BN) are inherited by a derived net (DN). The inheritance is viewed as the reuse of information. The overall framework is also considered in the light of alignment to neural models; for example, which order (or level) of generalization can be performed by which specific type of neuron model.

The complete framework may not be applicable to all neural models; in fact, some orders may be special cases which apply only to specific neuron models. This is, indeed, shown to be the case. Lower-order generalization is viewed as a general case and is applicable to all neuron models, whereas higher-order generalization is a particular or special case. This paper focuses on initial results; some of the aims have been demonstrated and amplified through the experimental work.

Keywords: Generalization; Information inheritance; Reuse of information; Higher-order; Sigma–pi; Recognition; Classifier; Interpolation; Extrapolation; Hyper-extrapolation; Symmetric transformations; Weight generation.

1. Introduction

The ability to generalize is one of the main reasons why artificial neural networks (ANNs) are utilized for so many different tasks. Generalization, in its most basic form, is an ANN's ability to automatically classify data that were not seen during training [31]. This differs from the view taken in statistical learning [73,83] which states that "in learning from a set of examples, the key property of a learning algorithm is *generalization*: the empirical[1] error must converge to the expected[2] error when the number of

[1]Empirical error is a mean sum squared error calculation that extends over a discrete set. Empirical error is the normal error utilized during supervised training, such as the backpropagation training regime [102].

[2]Expected error is an integration of the square error over the true function that extends over a set of continuous intervals.

E-mail address: r.neville@co.umist.ac.uk

0925-2312/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2007.05.003

Nomenclature

$\sigma(\cdot)$ termed the sigmoidal, logistic, or squashing function, $\sigma(\cdot)$ defined by $y=\sigma(a)=1/(1+e^{-a/\rho})$, where p is a positive valued parameter that determines the shape or 'softness' of the curve.

$W=[w_1,\ldots,w_n]$ termed a weight array or matrix, it is a row matrix representing an array of weights indexed from 1 to n.

$P_\mu=P_{\{\mu_1,\mu_2,\mu_3\}}$ where P is defined as a probability, then P_μ is the probability given input address μ.

μ termed "Mu," μ will be written μ_i, so that μ is the string $\mu_1,\mu_2,\ldots,\mu_n$. In this article, μ is an address.

w termed a weight, it applies a weighting, normally multiplicative, to another variable, i.e. xw multiplies x by w.

$P=\begin{bmatrix}P_1\\ \bullet\\ \bullet\\ \bullet\\ P_n\end{bmatrix}$ a probability array or matrix, it is a column matrix. An array of probabilities indexed from 1 to n.

$W\bullet P$ where "$\bullet$" denotes the inner product, $W\bullet P$, or the dot product. The "$\bullet$" is also known as the scalar product since its result is a number, rather than a vector.

Σ termed sum or summation of a series, e.g. $\sum_\mu$ implies summation over all values of μ and $\sum_{i=1}^{i=n}$ implies summation over index i from $i=1$ to $i=n$.

Π termed product of a series, e. g $\prod_{i=1}^{i=n}$ implies multplications of a number of terms over index i from $i=1$ to $i=n$.

$\in$ means "belongs to", i.e. $x\in A$ means "x belongs to the set A" or "x is a member of the set A."

$\aleph$ the set of natural numbers $1,2,3,\ldots$

: Used in set notation and set theory The notation $\{y_i\colon i=1,\ldots,n\}$ (or sometimes $\{y_i\,|i=1,\ldots,n\}$) is used to denote the set, $\{y_i\}$, containing all objects, given that $\{y_i\}$ is indexed over a set of natural numbers $\{1,2,3,\ldots,n\}$.

$\rightarrow$ means "maps to" between variables, functions, or sets, i.e. $f\colon x\rightarrow x^2$ or $\{x_1,x_2\}\rightarrow\{x_a,x_b\}$

$\approx$ means approximately equal to.

$\neq$ means not equal to.

$\forall$ means "for all."

$<$ means less than.

$||\,||$ means modulus, implies squared then square root, i.e. $\sqrt{()^2}$.

$\partial E/\partial W$ means the derivative of the function E at a point where the slope of the tangent is at $(W,E(W))$.

y^* superscript star, implies approximation, such as an approximate or estimate of a function, $y^*=f(x^t)$.

$f^{\leftrightarrow}(x)$ means reflection in a horizontal line or reflection in a line perpendicular to the y-axis.

$f^{\uparrow}(x)$ means reflection in a vertical line or reflection in a line perpendicular to the x-axis.

$f^{\leftrightarrow}(f^{\uparrow}(x))$ means combination or sequence of reflections. First reflection in a horizontal line or reflection in a line perpendicular to the y-axis, then reflection in a vertical line or reflection in a line perpendicular to the x-axis.

$f^{\updownarrow}$ means scale or multiply a function in the y direction

f^{θ° means rotate a f unction by θ degrees and can be augmented to f^{θ°, implies rotation of a set of functions.

$\Re_\theta$ means rotation matrix $\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}$ when applied to a point, $\{x,y\}$, the rotated coordinates are $x(\theta)=x\times\cos\theta+y\times\sin\theta$, and $y(\theta)=-x\times\sin\theta+y\times\cos\theta$.

samples n^3 increases[4]". This implies that an algorithm which guarantees good generalization for a given n will predict[5] well, if the empirical error on the training set is small. In the context of artificial intelligence, learning from observation may be cast as a process of searching for a good hypothesis: in this context, generalization is the extension of a hypothesis to correctly categorize and include new examples [88]. When learning is aligned to machine learning, generalization can be viewed as 'performance evaluation', achieving a higher degree of predictive accuracy when tested on unseen data [60]. In a frame-based[6] expert system interpretation, generalization is characterized by 'a-kind-of' or 'is-a' relationship between objects [74]. Clearly, there are numerous interpretations of generalization within different research fields. This paper develops a framework within which generalization specifically relating to ANNs is quantified.

[3] n is the number of examples in the training set.

[4] In their paper, Mukherjee et al. [73] state that: "the precise notion of generalization defined here roughly agrees with the informal use of the term in learning theory".

[5] Predict—"the basic goal of supervised learning is to use the training set to *learn* a function that evaluates [previously unseen] new inputs, x, and predicts the associated output, y [73,83].

[6] A frame is a data structure which captures knowledge that represents a particular object or concept.

Research into generalization in ANNs has focused on network generalization performance [20,48,94,99]. Other researchers have investigated the actual structures (or topology) of ANN and their effect on generalization [2,5,66]. This work has, as you would expect, evolved towards adaptive (or self-evolving) network topologies [29,51,86]. Naturally, methodologies for improving generalization are under investigation [59,76,104]. However, apart from [31] no real framework or descriptive methodology has been put forward to categorize different types of generalization.

This paper presents a framework for categorizing generalization that is both detailed and systematic. It proposes a framework within which different types of generalization can be evaluated. Previous research divided generalization into three categories [31]. However, these categories were fuzzy and imprecise. This paper further refines existing definitions by assigning to each category a logical predicate that it must fulfill in order to achieve a specific type (or order) of generalization. Then, it breaks the orders down into four different categories. The more sophisticated types of generalization are termed higher-order.

Higher-order generalization utilizes information inheritance. Information inheritance enables a trained base (or *parent*) net's weights to be symmetrically transformed and inherited by a DN that then maps a related but different isomorphic function. Information inheritance is achieved utilizing matrix transformations [75] (in this paper, they have been reformulated as STs [7]) which enable other DNs to have their weights prescribed (or generated). The framework generates the weights of an ANN using three steps.

Step I—a base net's (BN's) weights are generated by training the BN to perform a regression task, e.g. a base function.

Step II—a DN's weights are generated by ST of the BN's weights. The DN then maps a symmetrically related function.

Step III—finally, to attain the inverse of a symmetrically related function, a derived-trained net's (DTN's) weights are generated utilizing the DN. The DN generates an inverse set of previously unseen training vectors. The generated training vectors are then used to train the DTN. The DTN performs a regression task that then maps a symmetrically related inverse of the function mapped by the DN.

Note that the DN's and DTN's weights enable the so-called derived nets (DNs) to perform higher-order generalization by mapping a symmetrically related (or isomorphic) function. Here, inversion (or inverse) implies the transformation of points $\{x,y\}$ to a corresponding set of points $\{y,x\}$ known as their inverse points. It is also aligned to permutation inversion; a pair of elements $\{y,x\}$ is called a permutation inversion if the order of the elements was previously $\{x,y\}$. The DN could be cast as *intuitive* as this implies immediate perception or having knowledge without prior training.

A framework is then developed which enables higher-order generalization in ANNs to be categorized. The process was undertaken in six phases. These are listed below:

Phase I—development of a higher-order generalization framework.

Phase II—derivation of the distance function used to compare base and derived functions in order to validate that they are symmetrically related. Thus, we can validate that the DNs perform higher-order generalization.

Phase III—development of an ST theory based on matrix transformations. The third phase develops the necessary ST to generate the weights of DNs that perform higher-order generalization.

Phase IV—training of the BN.

Phase V—utilization of information inheritance (in the form of ST transformations of weights in phase III) to instantiate the DN's weights. If necessary, train the DTN with the inverse vectors generated by the DN.

Phase VI—validate that the BN, DN, and DTN perform 0°, 1°, 2° or 3°, generalization (utilizing the distance function derived in phase II), in an empirical (or experimental) phase.

During this experimental phase, a set of basic and complex polynomial functions were used to evaluate the regression or function mapping ability of ANNs which are claimed to have higher-order generalization capabilities. These were utilized in order to provide a more general case through which to evaluate this research. A hypothesis could then be formulated to test whether the general case is true for one-to-one (o–o), many-to-one (m–o) (single-valued functions), and one-to-many (o–m) (multi-valued functions). A detailed description of the different types of function is given in Section 4.1.2. The m–o and o–m regression tasks are undertaken in order to show that the higher-order framework and the associated transformations are not only applicable to simple regression tasks, e.g. o–o. By evaluating these types of regression tasks, we are able to quantify the claims and state explicitly which type of function (o–o, m–o and o–m) validate the framework. In order to validate the framework and to make the associated procedures more generally applicable to connectionists, a set of four geometric transformation procedures was devised in Section 3.2.14.

The paper associates an ST and a mathematical (symbolic) notation relating to a transformation of a function, $T()$, in a particular manner. For example, an inverse transformation, $\ldots S^{-1}$, is symbolically associated with a transformation of a function $f(x)$ to $f^{\mapsto}(x)$ and is denoted "CCS^{-1} (i.e. $T() \Rightarrow f^{\mapsto}(x)$)", where "$\Rightarrow$ "implies 'equivalent to' a function, $f^{\mapsto}(x)$, this notation is utilized in 3.2.14. Note that $f^{\mapsto}(x)$ implies reflection in a line perpendicular to the y-axis, while f^{1} implies reflection in a line perpendicular to the x-axis.

In previous research, [31] investigated issues relating to three levels of generalization. One issue that Gurney did not develop further, and hence was the motivation for this study, was a more precise definition of the levels of generalization. These issues are addressed in this paper by the presentation of a set of logical predicates that have to be met in order to achieve each level of generalization. In

Table 1
Tabulation of description of higher-order generalization

Order	Level	Symbol	Equivalence	Description	Generic type
0	–	0°	–	The network's weights are generated utilizing a set of stimuli and its prerequisite is that it must correctly map its stimuli	Look-up table
1	1	1°_1	–	The network's weights are generated utilizing a set of stimuli and its prerequisite is that it must correctly map unseen data through interpolation	Interpolation
1	2	1°_2	–	The network's weights are generated utilizing a set of stimuli and its prerequisite is that it must correctly map unseen data through extrapolation	Extrapolation
2	–	2°	0°	The network's weights are generated and its prerequisite is that it must correctly map unseen symmetrically related (or isomorphic) data (e.g. reflected (or rotated) and/or scaled functions)	Hyper (or transformed) look-up table
3	1	3°_1	1°_1	The network's weights are generated and its prerequisite is that it must correctly map unseen data that map symmetrically related data utilizing interpolation	Hyper-interpolation
3	2	3°_2	1°_2	The network's weights are generated and its prerequisite is that it must correctly map unseen data that map symmetrically related data utilizing extrapolation	Hyper-extrapolation

addition, the orders of generalization are more strictly differentiated.

This paper begins by relating this research to other connectionist studies. It then introduces a systematic framework for categorizing higher-order generalization. Table 1 provides textual definitions to describe higher-order generalization. Table 2 then gives a set of visual depictions of the different orders. After this, the neural model utilized in the regression mapping networks in this study is introduced. The following paragraphs outline the underlying theory for isomorphic functions, the derivation of the distance function, symmetric transformations (STs), and geometric transformations. The experimental work then validates our claims, using a number of regression or function estimation tasks. Finally, conclusions are drawn. A nomenclature (or taxonomy) of the entire mathematical notation included in this paper is provided to help the reader comprehend the notation used.

2. Related research

2.1. Research into generalization

Research into generalization in artificial networks covers a number of subdomains, from investigation into a network's generalization performance to studies into generalization performance aligned with different network structures or topologies. Investigations have also been carried out into methods of improving or optimizing generalization. Several researchers have presented methods for estimating or predicting generalization error. A few have even put forward theories of generalization themselves. An overview of all these subdomains is briefly given below.

2.1.1. Generalization performance of networks

The most basic research into generalization initially focussed on the ability of neural networks to interpolate and extrapolate [20]. This was followed by research, which concluded that large training sets do not guarantee better generalization and that training sets with subsets of border patterns may be no better than any others [99]. Some connectionist researchers have investigated the generalization capabilities of backpropagation and recirculation networks [48], while others have investigated the generalization of feedforward networks [94]. Bayesian criteria have been applied to neural networks [8,68,69] and have been shown to generalize better than early stopping when learning nonlinear functions [90]. Holden and Rayner studied the generalization of single-layer radial basis function networks [44]. Lawrence et al. [63] presented an investigation into local minima and generalization. The effects of initial conditions on generalization performance in large networks has been studied by Atiya and Chuanyi [3]. Two papers by Chen and Mills [13,14] looked at methodologies for analysing neural network generalization in control systems. Fault tolerance and generalization have been studied by Hyeoncheol and LiMin [49]. Other researchers have also investigated the generalization ability of fault-tolerant feedforward neural networks [22]. Phanak [82] studied inter-related fault tolerance, generalization and the Vapnik–Chervonenkis (VC) dimension.

Ji [53] speculated why generalization occurs with respect to sample size and capacity of the neural network. His analysis presented an explanation as to why the number of samples needed for generalization may be fewer than the bounds given by the VC dimension. Other researchers have inferred that it is possible to determine the level of generalization by monitoring learning behaviour [106].

2.1.2. Neural net structure in relation to generalization performance

It is important to note that the Kolmogorov theorem [58] underpins this area as it implies that any continuous

function can be implemented by a three-layer feedforward neural network (3LFNN) [41,43], utilizing different non-linear activation–output (a–o) transfer functions [19]. Other researchers in the field [42,43] have also stated that an ANN of at least three layers or more is required to approximate a function f. Note, another body of research has stated that only one internal hidden layer (or a two-layer) ANN is necessary if sigmoidal a–o functions are employed [19,45,46].

Research has also been carried out into the size of a network and its effect on generalization [5]; further work in this area investigated the number of hidden units in a network [2]. Studies [66] have been carried out into particular types of cascade network architectures (CNN). Adaptive network topologies have also been investigated [29,51,86]. An interesting study into one- and two-hidden layer feedforward networks was carried out by Redondo and Espinosa [85]. They concluded that one hidden layer was better than two and that two hidden layers were more prone to fall into bad local minima. In the same year, Zhong and Cherkassky [110] investigated the factors which control the generalization ability of multi-layer perceptron (MLP) networks. Sparse architectures have also been studied by Chakraborty [11]. Several researchers have conducted research using regularization. One paper [92] studied adaptive regularization and compared two different types, whilst another [79] compared a three-layer recurrent neural network (3LRNN) with a 3LFNN. Regularization with respect to additive noise was investigated and a regularizing term was derived based on a general noise model [12]. Also, weight elimination [101] has been used to help prevent overfitting, which should promote generalization.

2.1.3. Methodologies for improving generalization

The next step in this area of research was to investigate methods of enhancing the generalization performance of neural networks [76]. Kruschke [59] looked at methods for improving generalization in backpropagation networks with distributed bottlenecks. Whitley and Karunanithi [104] presented two methods for improving generalization. One method selected the training ensemble while the other partitioned the learning strategy. Williams [105] developed a regularization or penalty term to improve generalization. In 1994, Watanabe and Shimizu [100] introduced a new learning algorithm for improving generalization. Sarkar [89] developed a measure of randomness in generalization ability. Many researchers have also utilized genetic algorithms (GAs) or evolutionary computing methodologies to improve generalization. Bebis and Georgiopoulos [6] were among them; their paper utilized a GA to build nets that had good generalization performance and which were relatively small, and hence decreased the number of free parameters. Another method of building hierarchical networks was developed using what are termed 'stacked generalizers' [27]. Szabo and Horvath [47,93] investigated methods of improving the generalization capabilities of cerebellar model articulation controller (CMAC) neural networks. Researchers have also utilized modular ANNs [25] to develop a method of selecting training examples, as opposed to the random selection of examples, thus showing that their algorithm improves generalization. Another approach [54,55] utilized teacher-directed learning to improve generalization. In this work, they surmised that information which relates to training or target patterns should be maximized. A survey of methods of improving ANN generalization [107] segmented them into five categories that work through: modifying the training set (i.e. modifying the objective function with a regularization component); adjusting the fitting ability of an ANN (e.g. adjusting the size of the net, growing, and pruning); ending network training time (i.e. early stopping); integrating ANNs (e.g. ensemble of ANNs); and giving a view on different types of training samples (e.g. consequence of added noise at different levels).

The above research investigated methods for improving generalization performance; whereas Ogawa and Oja [80] researched optimally generalizing neural networks (OGNNs). Lin and Meador's [64] study presented a metric (i.e. a measurement) which enabled trained ANNs to optimize classification accuracy of unseen patterns.

2.1.4. Estimating/predicting generalization error

Research in this domain has been undertaken by Wada and Kawato [95]. They presented an information criterion for predicting the generalization capability of MLPs. Another metric (or measurement), termed generalized prediction error (GPE), was based on the number of effective parameters [71]. Larsen [61] went on to develop another estimate of generalization error (GEN). Other researchers [81] have minimized an estimated generalization error in order to prevent overfitting. Larsen and Hansen [62] developed a new average generalization error (FPER), and compared it with previous work (i.e. FPE and GPE).

Before concluding this short summary, a notable paper that evaluated neural networks and the bias/variance dilemma [26] must be mentioned. It contrasted ANN to nearest neighbour regression, and Parzen-window regression, and presented the limitations of neural modelling.

Finally, directly associated research by Christiansen [15] is complementary to the research presented in this paper. Christiansen put forward the conjecture that improved learning and generalization can be facilitated through the acquisition [learning] of multiple related functions. In fact, he stated that, "forcing neural networks to learn several related functions together results in both improved learning and better generalization." Our dual conjecture is that, "weights derived from a net which learns one function can be utilized to prescribe the weights for other nets which perform related isomorphic functions." In this paper, the newly prescribed networks are termed DNs.

3. Methods

3.1. Systematic framework for categorizing higher-order generalization

In this paper, different types of generalization are viewed as *orders*. This naming convention is elaborated in the three tables below.

Table 1 first tabulates zero-order (0°), first-order (1°), second-order (2°) and third-order (3°) generalization in column one. It then splits a number of these into two different levels (subtypes). This is followed by the symbolic notation—where the number denotes the cardinality of the order, the superscript '°' specifies that it is an order (e.g. zero-order), the subscripts '1' or '2' specify its level (i.e. level 1 or 2), e.g. 1°_{1} implies first-order (1°) level 1 (i.e. level 1_1 denoted by the subscript). After this, the equivalence is given, e.g. 2° is equivalent [analogue] to 0° in that they both correctly map the training data in some way. Then, a textual description is aligned to each order (level). Finally, the *generic type* of generalization is specified in the final column, and *type* is aligned to work in the field of data abstraction and hierarchy [65], in particular types and type hierarchy. In recent years, types and type hierarchy concepts have been researched under the guise of software patterns [17] and symmetry [108,109]. In the context of conventional generalization, machine learning extrapolation may be viewed as out-of-sample performance, whereas interpolation is cast as in-sample performance.

Each of the above orders of generalization is validated by the fact that the regression tasks are "statistically similar." The statistical similarity is accessed via a Euclidian distance metric 3.2.6. Note the use of the term 'generated' in the textual description of 0° and 1° generalization. In this paper, they are generated by training. But they could also be calculated using any other methodology that developed a valid set of weights—e.g. a method such as evolutionary computing could be employed to develop the weights. It is important to note that Table 1 implicitly implies that the mappings that each order performs are all related to a base function. The BNs that perform 0° generalization are trained with a set of input–output stimuli; this is the only set of training vectors utilized throughout the whole process. Note that *map* is viewed as synonymous with approximate, or estimate, $y^* = f^*(x)$. The above framework is supported in Table 2 with a set of visual depictions of the different orders.

Table 2 visually depicts generalization with respect to a function mapping task, where $y^t = f(x^t)$. The diagrammatic notation used is:

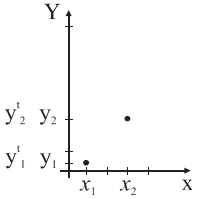
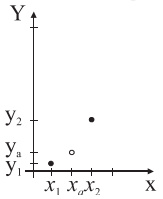
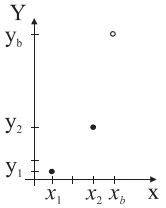
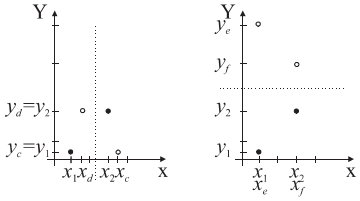
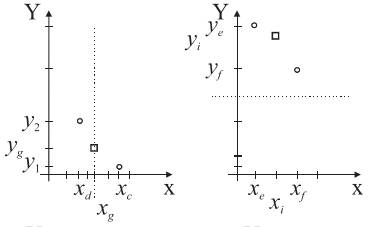
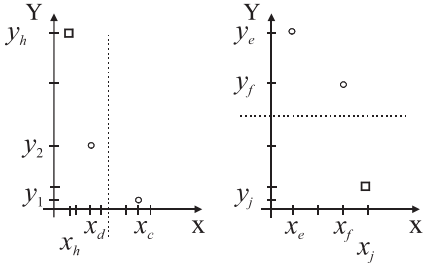
'●' (filled-in circles) training data,

'○' (empty circles) test stimuli or unseen validation data,

'□' (squares) used in third-order depiction to denote interpolation (3°_{1}) or extrapolation (3°_{2}).

Next, it is important to note that the dotted lines in second- and third-order cases denote the line or plane about which the points (function) are reflected. They also denote a symmetry axis, or a line of symmetry for a graph.

Table 2
Visualization of *n*th-order generalization

Order	Level	Symbol	Visualizations
0	–	0°	
1	1	1°_{1}	
1	2	1°_{2}	
2	–	2°	
3	1	3°_{1}	
3	2	3°_{2}	

3.1.1. 0° visualization

To attain 0° generalization the neural network's task is to approximate or estimate the function, $y^* = f(x^t)$. Training data is depicted as filled-in circles '●'; these map input stimuli x_i to output target response y^t_i. An ANN is trained with an input vector $X^t \in \{x^t_1, \ldots, x^t_n\}$ given $n \in \aleph$. It is trained to map this input vector to an associated target output vector $Y^t \in \{y^t_1, \ldots, y^t_n\}$ given $n \in \aleph$, where $\aleph$ is the set

of natural numbers 1,2,3, … The graph depicted in Table 2, order 0, level-(0°) shows the input vectors $\{x_1,x_2\}$ and the actual output vectors $\{y_1,y_2\}$ derived from a trained neural net that is said to be performing zero-order generalization. In fact the output vectors $\{y_1,y_2\}$ of this net are equivalent to the target training vectors $\{y_1^t, y_2^t\}$ that were used when a supervised training regime was utilized to train the net. Hence, all the network has done is to correctly classify or map a set of input vectors to a set of output vectors, i.e. $\{x_1, x_2\} \rightarrow \{y_1, y_2\} \approx \{y_1^t, y_2^t\}$. Note that $\rightarrow$ means 'maps to' and reads $\{x_1,x_2\}$ maps to $\{y_1^t, y_2^t\}$.

3.1.2. 1° visualization

The graph at order 1, level 1 (1_1°) depicts interpolated first-order generalization. The unseen test stimuli, x_a, are presented to the trained net and they elicit the response y_a. x_a are test stimuli '○' that are positioned in between two input stimuli utilized for training, $\{x_1,x_2\}$. By using the training data '●' the net is able to utilize the knowledge encapsulated in it during training to output a response that is in line with the regression task it was trained to map. Hence, it has performed interpolated first-order generalization. There is a caveat to this statement, in that if the data points surrounding the test stimuli are not sufficiently close—for example, in a Euclidian distance [40] sense—the net may not be able to model the function sufficiently and it may not generalize well.

The graph at order 1, level 2 (1_2°) visualizes extrapolated first-order generalization. The unseen test stimuli, x_b, are presented to the trained net and they elicit the response y_b. x_b is outside the cluster of two input stimuli utilized for training $\{x_1,x_2\}$. If x_b is close enough to the cluster of training points (i.e. the Euclidian distance is small), the elicited output y_b will be in line with the regression task that the net was trained to map and will be performing extrapolated first-order generalization. Note that this is again predicated on the Euclidian distance between the unseen test stimuli, x_b, and the nearest training input stimuli x_2. If the distance is too large, the net may not generalize well.

3.1.3. 2° visualization

The graph at order 2, level-(2°) shows two different reflections of the original base function. The left-hand side (LHS) diagram depicts a net performing second-order generalization after the learnt data points '●' that represent a base function have been reflected in a line perpendicular to the x-axis (dotted line halfway along the x-axis). The right-hand side (RHS) depicts a net performing second-order generalization after the learnt data points have been reflected in a line perpendicular to the y-axis (dotted line halfway up the y-axis). The transformed data points '○' are represented as input vectors $\{x_c,x_d\}$, $\{x_e,x_f\}$ and the actual output vectors $\{y_c,y_d\}$, $\{y_c,y_d\}$ are derived by transforming the trained base neural net's weights. These transformed weights are prescribed to the DNs; it is the DNs that perform second-order generalization. Hence, all that the DNs have done is to correctly classify or map a set of transformed input vectors to a set of transformed output vectors, i.e. $\{x_1,x_2\} \rightarrow \{x_c,x_d\}$, $\{x_e,x_f\}$ and $\{y_1, y_2\} \approx \{y_1^t, y_2^t\} \rightarrow \{y_c, y_d\}$, $\{y_e, y_f\}$. This could be cast as hyper-extrapolation (or to be more precise a hyper, or transformed, look-up table), as none of the base function's data points 'covers' the space that the reflected points cover. Note that $\rightarrow$ means 'maps to' and reads x$\{y_1^t, y_2^t\}$ maps to $\{y_c, y_d\}$.

3.1.4. 3° visualization

The graph at order 3, level 1 (3_1°) depicts interpolated third-order generalization. The unseen test stimuli, x_g and x_i, are presented to the transformed prescribed DNs and they elicit the responses y_g and y_i. x_g and x_i are in between two transformed input stimuli, $\{x_d,x_c\}$ and $\{x_e,x_f\}$. These unseen test stimuli are depicted as boxes '□'. The transformed points are depicted as circles '○'. The transformed points can only be elicited once the BN's weights have been transformed and allocated to the prescribed DN. By using the transformed weights, the nets are able to utilize the knowledge encapsulated in them during the prescription phase to output responses that are in line with the transformed functions they were assigned to map. Hence, they have performed interpolated third-order generalization. There is a caveat to this statement in that if the data points '○' surrounding the test stimuli '□' are not sufficiently close—i.e. in a Euclidian distance [40] sense—the nets may not be able to model the transformed functions sufficiently and they may not generalize well.

The second diagram at order 3, level 2 (3°), visualizes extrapolated third-order generalization. The unseen test stimuli, x_h and x_j, are presented to the DNs and they elicit the responses y_h and y_j. x_h and x_j are outside the cluster of two transformed input stimuli, $\{x_d,x_c\}$ and $\{x_e,x_f\}$. If x_h and x_j are close enough to the cluster of transformed points (i.e. the Euclidian distance is small), the elicited outputs y_h and y_j will be in line with the functions that the nets were derived to map and will be performing extrapolated third-order generalization. Note that this is again predicated on the Euclidian distance between the unseen test stimuli, x_h and x_j, and the nearest training input stimuli x_d and x_f. If the distance is too large, the net will not generalize well.

3.1.5. Logical predicates for higher-order generalization

Table 3 depicts the logical predicates that have to be met for each order to be attained. The table depicts the order, then the required logical predicate, and lastly the meta-data that helps to explain the predicate. Note that when target vectors are references, a superscript 't' is assigned to the input x_i^t or output y_i^t, $\{x_i^t, y_i^t : i = 1, \ldots, n\}$. Where they are assigned a superscript 'v', this refers to the validation [unseen] vectors, $\{x_j^v, y_j^v : j = 1, \ldots, n\}$.

Constraint C_1: $\{x_j^v \neq x_i^t \forall x_j^v\}$;

Constraint C_2: $d_{ij} < d_{\max}$ given that: $d_{ij} = ||x_i - x_j||$.

Note 1: The approximate equal sign, $\approx$, is used since a supervised training regime cannot guarantee that the

Table 3
Logical predicates required to attain *n*th-order generalization

Order	Logical predicates	Note
0	If and only if $y_i^* \approx y_i^t \forall x_i^t$	1
1	If and only if $y_j^* \approx f(x_j^v) \forall x_j^v$ given C_1 and C_2	2
2	If and only if $y_i^* \approx T(y_i^t) \forall (x_i^t)$	3
3	If and only if $y_j^* \approx T\left(f(x_j^v)\right) \forall T\left(f(x_j^v)\right)$ given C_2 and C_2	4

gradient ∇ or the derivative with respect to the weights $\partial E/\partial W$ at the end of training is $\partial E/\partial W = 0$.

Note 2: d_{ij} is the distance between the unseen stimuli, $x_j \in \{x_a, x_b\}$, and the nearest input training stimuli $x_i \in \{x_1, x_2\}$, x_1 in the case of x_a and x_2 in the case of x_b. d_{ij} is the Euclidian distance between the test stimuli x_a (or x_b) and the input training stimuli x_1 (or x_2).

This implies that in order to perform 1° generalization the unseen stimuli must elicit output response y^* to estimate (approximate) the function $f(x^t)$ that the net was trained to map. Here, the approximation is bounded in that the Euclidian distance between the unseen test stimuli, x_b, and the nearest training stimuli, x_2 (or x_a and x_1), must be less than a maximum value $d_{\max}$; this could be viewed as a constraint. However, the maximum Euclidian distance, $d_{\max}$, is not a value that can be easily calculated. Suffice to say that, as this Euclidean distance increases, the generalization ability of the net drops off. Note that the approximate equal sign, $\approx$, is used in this case to imply that one function, y^*, is approximately equal to another function, $f(.)$.

Note 3: T defines a transformation of the output regression task (function mapping task $y^* = T(f(y))$) but in actuality transforms the BN's weights, W, to a set of transformed weights, $T(W)$, that are then utilized by the DN.

Note 4: This implies that in order to perform 3° generalization, the unseen stimuli must conform to the following requirement. In order for the estimated output response y^* to be in line with the function $T(f(x^t))$, the net derived to map it must meet the constraints defined for 1° generalization.

Section 3.2 explains the theory behind the paper's validation procedures for higher-order (2° and 3°) generalization.

3.1.6. 2° and 3° generalization validation

In order to validate that an ANN performed 2° and 3° generalization, each function (trained or derived) is validated utilizing the mean ranked distance metric, $\overline{D}_{rd}$, in Section 3.2.6. To be specific, the three other symmetrically related functions are validated by comparing the original base function, $f^b = f(x)$ with a set of related functions $f^r \in \{f^{r1}, f^{r2}, f^{r3}\}$. The related functions are transformed functions rotated by 90° $f^{r1} = f^{90^\circ}(x)$, by 180° $f^{r2} = f^{180^\circ}(x)$, and by 270° $f^{r3} = f^{270^\circ}(x)$.

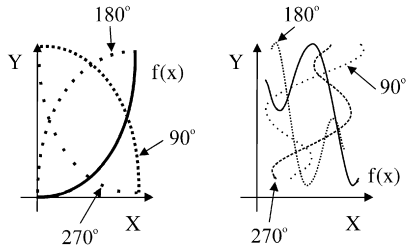


Fig. 1. Depiction of the three (symmetrically) related isomorphic functions.

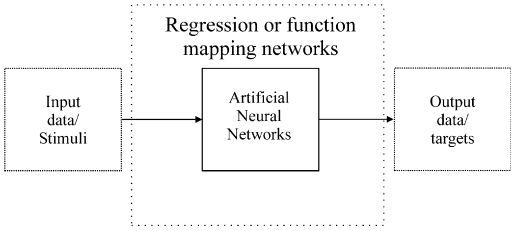


Fig. 2. Abstract depiction of the ANN model used to perform regression (or function estimation) tasks.

In order to map the set of isomorphic functions, the interval of the input spans was constrained to $x \in [0,1]$ (domain) and the output to $y \in [0,1]$ (range); hence the function state space was constrained to $f(x) \in [0,1]$ (range), see Fig. 1. They then met the requirements of the minimum and maximum bounds of the net's input and output.

3.2. The ANN model used to perform regression or function estimation tasks

The classical approach to performing regression (or function estimation [43]) tasks with ANNs is presented below. For an abstract depiction of the ANN model used to perform regression (or function estimation) tasks, see Fig. 2.

The topology of the ANN used to perform regression or function estimation tasks is explained in detail in Section 3.2.1.

3.2.1. Regression or function mapping networks

To perform higher-order generalization (2° and above) a procedure for generating *DNs* and *a DTN* which map symmetrically related functions is required. The method used to generate the weights is the inheritance of weights from the BN, as shown in Fig. 3.

The DNs inherit a set of transformed [75] weights from a BN. The DTN then utilizes the DN to generate an inverse set of training stimuli which are then used to train the DTNs. The *base's* network is trained, in the normal way, with a set of input stimuli, $\{x_1, \ldots, x_n\}$, and target output vectors, $\{y_1^t, \ldots, y_n^t\}$, pairs $\{x_1, y_1^t; \ldots; x_n, y_n^t\}$. The BNs are essential during the prescription phase of the other DNs and DTNs, because only the BN has an associated set of input–output vectors and, as such, can be trained.

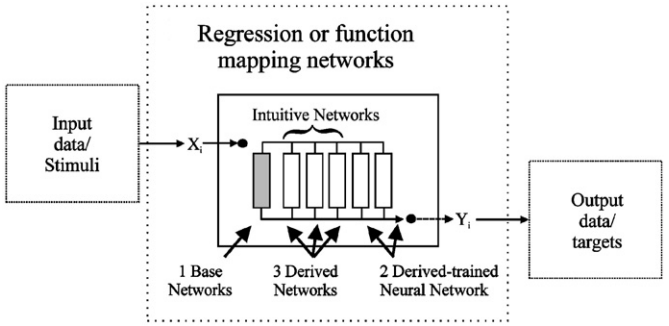


Fig. 3. Low-level depiction of the regression or function mapping networks.

The prescribed DNs have no target output vectors. Hence, the only way to generate the weights is to utilize the BNs by inheriting their weights then utilizing ST to transform the weights and instantiate the DN's weights. The algorithmics utilized to train the nets are specified below. First the neural model used for the experimental work is presented.

3.2.2. Introduction to the sigma–pi neuron model

The neural model utilized for this work is termed a sigma–pi neuron model [77,78]. The sigma–pi neuron's functionality viewed as a matrix equation [78] is

$$Y = \sigma(W_\mu \cdot P_\mu), \tag{1}$$

where, for a single unit, W is a row matrix of the weights $W_\mu = [w_1, \ldots, w_n]$, and P is a column matrix of the probabilities of the weights being addressed $P_\mu = [P_1, \ldots, P_n]^{\mathrm{T}}$ where $\mu \in \{1, \ldots, n\}$ is the index and σ a sigmoid function $Y = \sigma(U) = 1/1 + e^{U/\rho}$, given that ρ defines the shape of the sigmoid. They are termed sigma–pi units as their functionality has previously been defined by Gurney [31]. The real-valued activation can be defined as

$$a_{(t)} = \frac{1}{w_m 2^n} \sum_{\mu} w_\mu \prod_{i=1}^{i=n} (1 + \overline{\mu}_j z_i), \tag{2}$$

where z_i, the input probability distribution, defines the probability of the input x_i. Recently a subset of sigma–pi neurons, a multi-cube unit (MCU), has been used to study information processing in dendrites [35,36]. Previous papers showed how the MCUs stopped combinatorial explosion which could affect RAM-based neural models [31,32,34,37]. Sigma–pi units can be configured as multi-cube sigma–pi networks. A review of the sigma–pi neuron model and associated models is given in Appendix A. Ferguson [23,24] describes in great detail the backpropagation training algorithms utilized in this research.

3.2.3. Algorithmics for weight generation of the ANNs used to perform regression (or function estimation) tasks

The algorithmics for generating the weights of the ANNs used to perform regression or function estimation tasks are discussed below. First the BN is trained. Then the BN is utilized to generate the weights for the other DNs and DTNs.

The family of networks are composed of a set of ANNs, each with the same (internal) topology, see Fig. 3. However, the weights of the nets are generated using four different procedures, see Section 3.2.14.

3.2.4. Theory

The theory behind isomorphic functions is outlined in Section 3.2.5. This is followed by the theory of derivation of the distance function in Section 3.2.6, STs in Section 3.2.8, and, finally, geometric transformation procedures in Section 3.2.14.

3.2.5. Isomorphic functions: symmetric functions

The STs (utilized for higher-order generalization in Section 3.2.8) are isomorphic, i.e. "A is isomorphic to B" is denoted $A \cong B$ [38]. Isomorphic transforms denote geometric congruence, and are related to isometry. Isometry is a bijective map between two metric spaces that preserves distances, i.e. $d(f(x),f(y)) = d(x,y)$ where f is the map and $d(x,y)$ is the distance function. Isometries are sometimes also called congruence transformations. Two figures that can be transformed into each other by an isometry are said to be congruent [18]. The most significant point about these isomorphic transforms is that the intermediate distance, i.e. relationship between the data points (x_i,y_i), is retained after the transformation. This implies that if T defines a transformation then $f(d(x_i,y_i)) \cong T(f(d(x_i,y_i)))$ denotes an isomorphic relationship, *iff* (implies if and only if) the distances between the data points $d(x_i,y_i) \forall x_i,y_i$ are equivalent, before and after the transformation. The distance metric used to validate that the base and DNs map isomorphic (symmetric) functions is presented in Section 3.2.6.

3.2.6. Derivation of the distance function

A Euclidian distance function (see Fig. 4), is used to compare the different functions (which are the resultant of the symmetry transformations).

The specific steps taken to derive the final mean ranked distance function are:

1. Given two graphs, b, which represents the base function f^b, and d, which represents the derived function f^d,[7] the two graphs form a set $G \in \{b,d\}$ that represents two sets of point pairs $b \in \{(x_i^b, y_i^b), (x_{i+1}^b, y_{i+1}^b)\}$ and $d \in \{(x_i^d, y_i^d), (x_{i+1}^d, y_{i+1}^d)\}$, given $i \in (1,\ldots,n-1)$,[8] Fig. 4.
2. Calculate the Euclidian distance, ed_i, between all point pairs in each graph, (x_1,y_1) and (x_2,y_2):

$$ed_i = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \quad \text{for } i = 1, \ldots, n-1. \tag{3}$$

[7] Note that the derived function is obtained utilizing one of the symmetry transformations ST_x.

[8] A successor function is defined for each vector (n-tuple). Note: this is allowed because the tuples are either finite or countably infinite.

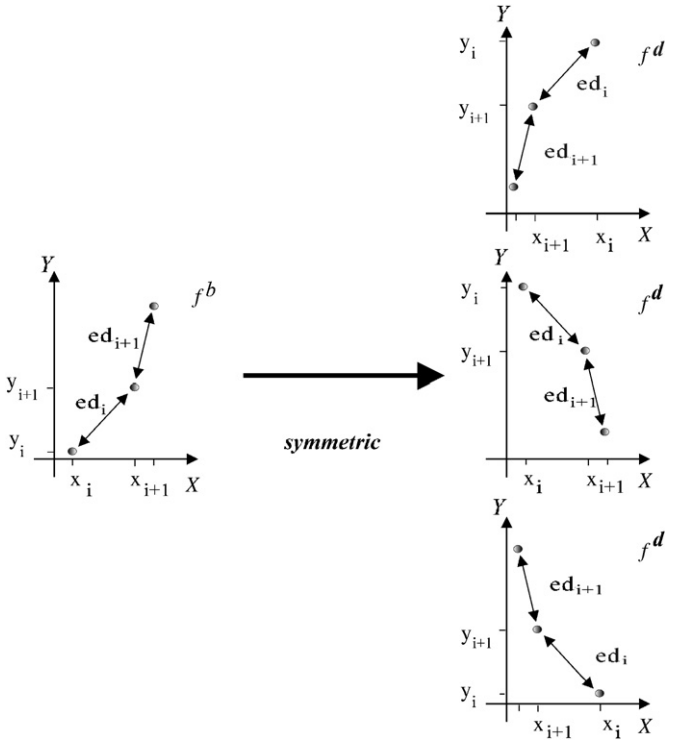


Fig. 4. Euclidian distances ed_i, between all pairs of points, (x_i,y_i) and (x_{i+1},y_{i+1}).

where n–1 is the number of ed_i calculations, also n–1 is the number of point pairs in each graph (given n points). The vector of Euclidian distances, $ED \in [ed_i,\ldots,ed_{n-1}]$ given $i \in \{1,\ldots,n-1\}$ is then formulated.

3. Calculate a vector, R, of ranked Euclidian distances for all subsequent point pairs in the graphs:

$$R = [\mathrm{Rank}_m(ED)] \quad \forall \quad m = \{1,\ldots,n-1\}, \tag{4}$$

m is defined as the rank index, and R is a vector of ranked Euclidian distances of all subsequent point pairs in the graphs. R is a vector of $n-1$ components. The rank of a value is simply the order in which it would appear if the pairs of points were sorted.

4. Calculate the difference between R^b and R^d for all the associated point pairs in the two graphs. The difference metric for each associated point pair is

$$\mathrm{Diff}_m = \left|R_m^b - R_m^d\right| \quad \forall \quad m = \{1,\ldots,n-1\}, \tag{5}$$

where $||$ defines the *abs*() absolute value. The absolute value is required as the difference $R_m^b - R_m^d$ may be negative, and the final mean ranked Euclidian distance is a non-negative number.

5. Finally, the mean ranked distance is

$$\overline{D}_{rd} = \tfrac{1}{n-1} \sum_{i=1}^{i=n-1} \mathrm{Diff}_i, \tag{6}$$

where $n-1$ represents the cardinality of ranked list/s.

6. If the mean ranked distance is less than or equal to 0.01:

$$\overline{D}_{rd} \leqslant 0.01, \tag{7}$$

then the two function graphs are treated as symmetric and the two functions are called symmetry functions or related functions f^r.

In previous research [77] 1% error (i.e. 99% precision) was utilized as an error metric, e.g. 1% is equal to 0.01. If this is cast as a root mean square error, e_{rms}, it implies a value of 0.01, which is equivalent to a mean squared error (MSE) of 0.0001 or 1×10^{-4}.

3.2.7. Geometric inheritance of information

We present a theory for performing a selection of geometric (symmetric) transformations in order to enable ANNs to perform higher-order generalization. The following sections present a set of STs used to build a family of ANNs, depicted visually in Fig. 5.

3.2.8. Symmetric transformations (STs)

The theoretical hierarchy of networks is shown in Fig. 5. In this hierarchy we have three major components:

1. The BN is trained to map a base function, f^b.
2. The weights of the DNs are generated by the transformation of the BN's weights.
3. The DTNs utilize the first two components in order to:
   - 3(i). generate the weights of a DN, which is hierarchically below the DTN.
   - 3(ii). generate the inverse training vectors for the DTN using a DN.
   - 3(iii). train the DTN with the inverse or exchanged DN's x and y vectors, i.e. $x \leftrightarrow y$.

Again, note that due to the complexity of the o–m function, a 3LRNN DTN is utilized, while a 3LFNN DTN is used for the o–o and m–o functions.

The DNs (in Fig. 5) are not trained to map the related functions, $\{f^{r_1},\ldots,f^{r_n}\}$; their weights are generated by multiplying the weight matrices of the BN by one of two STs. The size of the symmetric matrices below is dependent on the size of the weight vector. All the symmetric matrices below have been sized to match a two-weight vector node. Hence, the size of the weight vector is 2, i.e. $W = [w_1,w_2]$. If the weight vector (or matrix) is a two-element row matrix then the symmetric matrices below K, and M, will be a 2×2 square matrix. If the weight matrix is made up of q elements then the square matrix K, and M, will be a $q \times q$ square matrix.

3.2.9. Inverse transformation: inversion of polarity of internal weights

The symmetry transformation, S^{-1}, that performs reflection of the weights' polarity can be obtained by performing a weight transformation on the trained network's internal weight matrix, W. The S^{-1} is

$$S^{-1} = W \times K. \tag{8}$$

K is a symmetric matrix where $k = -1$ transforms the polarity of the weights in the weight matrix so

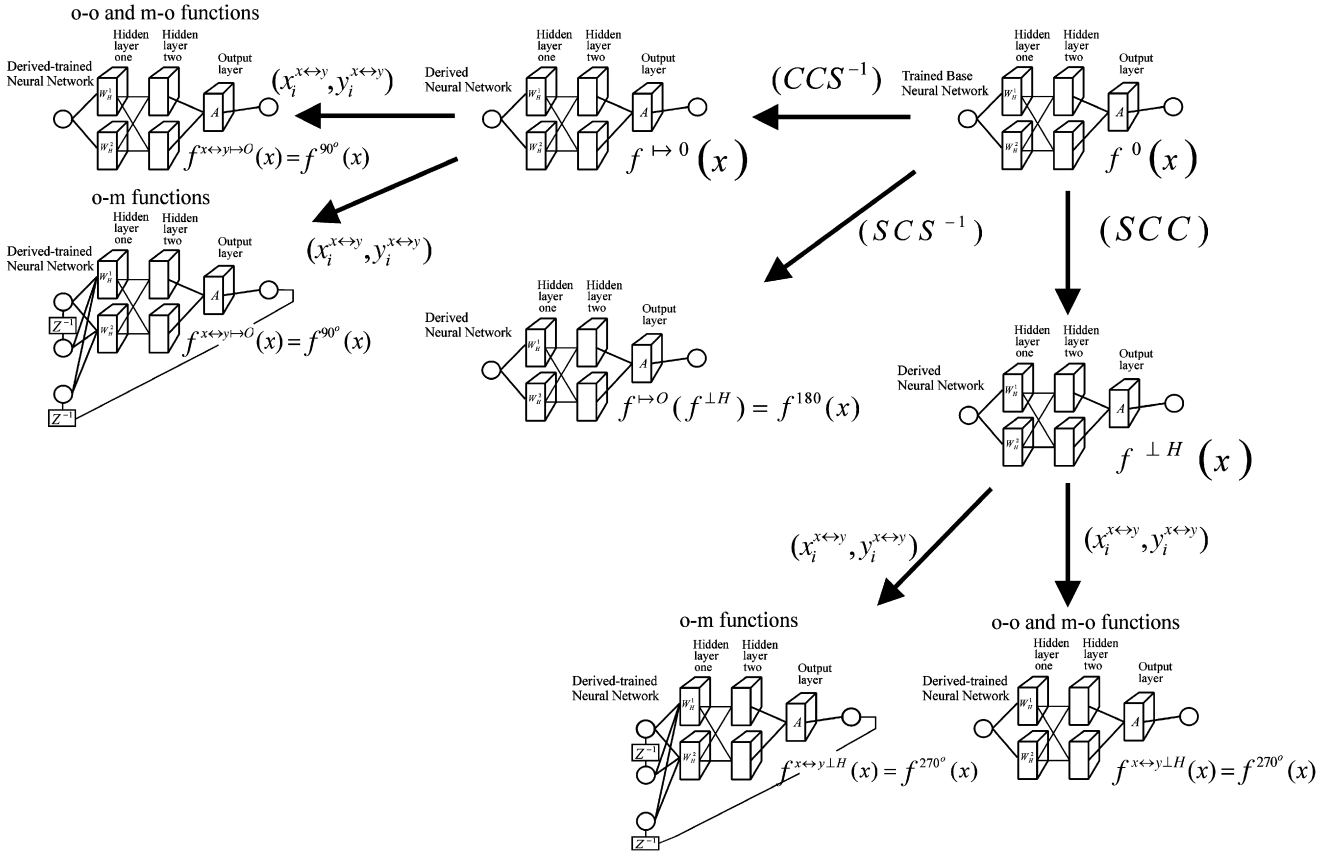


Fig. 5. Depiction of the set of STs used to build a family of ANNs.

that they now utilize the inverse or opposite weights and K is defined as

$$K = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{9}$$

For example, if $W = [w_1, w_2]$, then $S^{-1} = [w_1, w_2] \times \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = [-w_1, -w_2]$.

The transformed weights are assigned to a particular layer of the DN, while all other (non-transformed) layer weights are copied, C, to the DN and reused.

3.2.10. Symmetry transformation–permutation of the internal weights order

The symmetry transform, s, required to perform a reordering or permutation of the internal weights is

$$S = W \times M, \tag{10}$$

where M is

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{11}$$

For example, if $W = [w_1, w_2]$, then $S = [w_1, w_2] \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = [w_2, w_1]$.

Note that the s transformation is applied to a particular layer in order to calculate the weights for the DN. All other layer weights in the DN are copied, C, directly from the trained net.

3.2.11. Dilation transformation-scaling of the internal weights

The dilation transform, D, required to perform a scaling or dilation of the internal weights order is

$$D = W \times V, \tag{12}$$

where M is

$$V = \begin{bmatrix} 0 & v \\ v & 0 \end{bmatrix}. \tag{13}$$

For example, if $W = [w_1, w_2]$, then $D = [w_1, w_2] \times \begin{bmatrix} v & 0 \\ 0 & v \end{bmatrix} = [v \times w_2, v \times w_1]$,

where V is a symmetry transformation matrix. v scales or dilates the weights in the weight matrix, W, so that they are scaled in the Y-direction or *vertical* direction.

Note that the D transformation is applied to a particular layer in order to calculate the weights for the DN. All other layer weights in the DN are copied, C, directly from the trained net.

3.2.12. Sequences of transformations

In order to perform a sequence of transformations (or composite transformations), the above transforhmations

are performed one after another. For example, an S transformation can be followed by a second transformation, S^{-1}.

3.2.13. Mapping the ST to a 3-layer sigma–pi ANN

The above STs are applied to a 3LFNN utilizing the sigma–pi neuron model. The coding for this mapping is detailed below.

Firstly, in this paper the transforms are only applied to the first hidden layer, H_1, and the output layer, O. The second hidden layer's weights, H_2, are just copied, C, from the BN. The coding is portrayed as a tuple (or triple) relating to the three layers, H_1 H_2O. Hence, the coding SCC implies the transformation of the first layer's weights H_1 by S, while the second and output layers' weights, H_2 and O, are generated by copying the BN's weights. When two transforms are applied sequentially, they are coded SCC, CCS. When associating specific STs to particular layers in an ANN, the notation utilized for the resultant transformed functions, $f^{\mapsto 0}$ and $f^{\perp H}$, implies that the STs were applied to the output layer, super case ^{o} or the first hidden layer, super case ^{H}.

3.2.14. Geometric transformation procedures

In order to derive all the necessary symmetrically related functions referred to in Section 3.1.6, four procedures were developed. Procedures I–IV enable different symmetrically related functions to be derived. These are explained below.

Procedure I

Procedure I trains the BN with a training set with the backpropagation training regime. The procedure builds on the theoretical foundations described in Section 3.2.8 and uses the following step.

I.1 Define and train a BN to perform a base function mapping task. This BN is called a generator because it is used to generate DNs.

Procedure II

Procedure II uses a symmetric transform (CCS^{-1}) to transform the BN's weights to derive an auxiliary DN. The auxiliary DN is then utilized in its vector generating mode to generate the inverse training vectors for a DTN. It does this by exchanging the input–output vector (generated by the DN) and then uses backpropagation to train the DTN. This enables the DTN to perform the $f^{90^\circ}(x)$-related function. The procedure builds on the theoretical foundations described in Sections 3.2.6 and 3.2.8 and uses the following steps.

II.1. Define one untrained DN and then transform the weights of the BN into this net using the CCS^{-1} weight transformations. The result is an auxiliary DN.

II.2. Validate that the DN's transformed weights $T(CCS^{-1}) \Rightarrow f^{\mapsto 0}$ map the correct derived function, and determine if it is symmetrically related to the base function using the validation method described in Section 3.2.6 to compare the $f(x)$ and the $f^{\mapsto 0}$ regression tasks. If the mean ranked distance is less than or equal to 0.01 the DN is symmetrically related to the BN.

II.3. Generate a set of inverse input–output vectors $\{x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y}\}$ utilizing the $T(CCS)^{-1}$ transformed DN. The vectors are termed the derived-generated (DG) vector set $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$; see Procedure Note 1 below for more information.

II.4. Train a DTN with the DG vector set, $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$; see Procedure Note 2 below for type of DTN trained.

II.5. Validate that the DTN maps the correct derived function by determining if it is symmetrically related to the base function using the validation method described in Section 3.2.6 to compare the $f(x)$ and the $f^{90^\circ}(x)$ regression tasks.

II.6. If the mean ranked distance is less than or equal to 0.01 the DTN is symmetrically related to the BN and may be termed a related net.

Procedure III

Procedure III uses a symmetric transform (SCS^{-1}) to transform the BN's weights to generate the DN's weights in order to enable it to perform the $f^{180^\circ}(x)$ related function. The procedure builds on the theoretical foundations described in Sections 3.2.6 and 3.2.8 and uses the following steps.

III.1. Define one untrained DN and then transform the weights of the BN into this net using the SCS^{-1} weight transformations. The result is a DN.

III.2. Validate that the DN's transformed weights $T(SCS^{-1}) \Rightarrow f^{\mapsto 0}(f^{\perp H})$ map the correct derived function by determining if it is symmetrically related to the base function using the validation method described in Section 3.2.6 in order to compare the $f(x)$ and the $f^{180^\circ}(x)$ regression tasks.

III.3. If the mean ranked distance is less than or equal to 0.01 the DN is symmetrically related to the BN and is termed a related net.

Procedure IV:

Procedure IV uses a symmetric transform (SCC) to transform the BN's weights to derive an auxiliary DN. The auxiliary DN is utilized in its vector generating mode to generate the inverse training vectors for a DTN by exchanging the input–output vector. It then uses backpropagation to train the auxiliary DTN. This enables the DTN to perform the $f^{270^\circ}(x)$ related function. The procedure builds on the theoretical foundations described in Sections 3.2.6 and 3.2.8 and uses the following steps:

IV.1. Define one untrained DN and then transform the weights of the BN into this net using the SCC weight transformations. The result is an auxiliary DN.

IV.2. Validate that the transformed weights $T(SCC) \Rightarrow f^{\perp H}$ map the correct derived function. Determine if it is symmetrically related to the base function using the validation method described in Section 3.2.6 which compares the $f(x)$ and the $f^{\perp H}$ regression tasks. If the mean ranked distance is less than or equal to 0.01 the DN is symmetrically related to the BN.

IV.3. Generate a set of inverse input–output vectors $\{x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y}\}$ utilizing the transformed net $T(CCS^{-1})$. These are termed the derived-generated (DG) vector set $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$; see Procedure Note 1 below for more information.

IV.4. Train a DTN with the derived-generated vector set, $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$; see Procedure Note 2 for the type of DTN trained.

IV.5. Validate that the derived-generated net maps the correct derived function by determining if it is symmetrically related to the base function using the validation method described in Section 3.2.6. This compares the $f(x)$ and the $f^{270^\circ}(x)$ regression tasks.

IV.6. If the mean ranked distance is less than or equal to 0.01 the derived-generated net is symmetrically related to the BN and may be termed a related net.

The BN, DNs and DTN form a family.

Procedure Note 1: The two sets of vectors were obtained by exchanging the input–output vector set to provide an augmented training vector set. The inverse vector set exchanged the x and y vectors, i.e. $x \leftrightarrow y$. Now, the input vector set is $x^{x\leftrightarrow y} = \{y_1, \ldots, y_{10}\}$ and the output target vector set is $y^{x\leftrightarrow y} = \{x_1, \ldots, x_{10}\}$, e.g. $\{x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y}\}$.

Procedure Note 2: Two types of DTN are required. In the case of the inverse o–o or o–m a FNN is utilized, while for the o–m function an RNN is required.

4. Empirical results

4.1. Experimental work: validating 0°, 1°, 2°, and 3°, generalization conjecture

4.1.1. Aims of experimental work

The major goal of the experimental work was to train and validate the nets. Validation involved testing that the nets performed 0°, 1°, 2°, and 3° generalization. Before delineating the experimental work, the types of regression task undertaken are discussed.

4.1.2. Types of regression tasks performed by ANNS

There are many different types of regression task that ANNs can be trained to estimate. The normal regression tasks undertaken by ANNs are the approximation of continuous functions $f{:}A \subset \Re^n \rightarrow f{:}B \subset \Re^m$, where $\{A,B\}$ are compact sets. The function f is normally defined as o–o and onto. These are typified by functions of the form $f = f(x) = x^2{:}\{x \in \Re^+$ and $x \in [0,1]\}$. However, other typical functions are m–o (i.e. $f(x) = \mathrm{Sin}(x)$). Both of these are single-valued functions. Typically, both o–o and m–o functions can be learnt by multi-layer feedforward ANNs. More complex functions are termed multi-valued, and could be viewed as o–m functions. These functions are typified by the inverse sine which is the multi-valued function [57] $\mathrm{Sin}^{-1}(x)$ [111], also denoted $\arcsin(x)$ [1,39,52], that is the inverse function of the sine. An example of a multi-valued function is depicted in Fig. 6C and D. In the experimental work below, a number of regression tasks were undertaken. One task was an o–o mapping task. The other was a complex polynomial m–o, mapped by the DN. A set of auxiliary DNs that initially mapped o–m functions was used to generate the inverse vectors for the $f^{-1}(x)$ o–m regression task. The o–m $f^{-1}(x)$ task was mapped by the DTN's net in Section 3.2.14.

The mathematical examples given in this research utilize:

(1) a simple polynomial function:

$$f(x) = x^2 : \{x \in \Re^+ \quad \text{and} \quad x \in [0,1]\} \tag{14}$$

as the o–o regression task (see Fig. 6A and B); and

(2) a complex polynomial function

$$f(x) = -91x^6 + 329x^5 - 409x^4 + 209x^3 - 37x^2 - 0.8x + 1 : \{x \in \Re^+ \quad \text{and} \quad x \in [0,1]\} \tag{15}$$

as the m–o (see Fig. 6C and D). In its inverse formulation, an o–m (e.g. $f^{-1}(x)$) task after transformation is presented in Figs. 7, 9C and D.

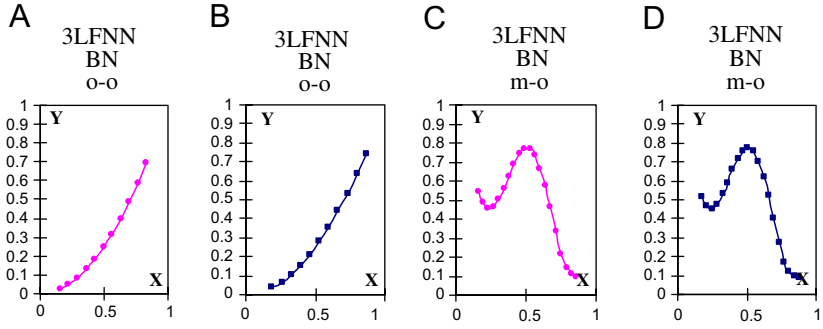


Fig. 6. 0° (A, C) and 1° (B, D) generalization test of the BN, $f(x)$.

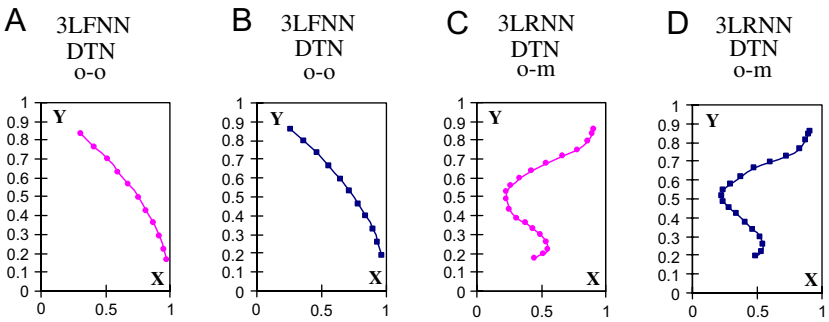


Fig. 7. 2° (A, C) and 3° (B, D) generalization test of the $f^{90^\circ}(x)$ generated by a DTN utilizing $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$ vectors, derived using transformation $T(CCS^{-1}) \Rightarrow f^{\leftrightarrow 0}$.

Two 3LFNN BNs mapped the o–o and the m–o base regression task, a set of 3LFNN DNs mapped the related o–o and o–m tasks, and a set of 3LFNNs and 3LRNNs mapped the inverse m–o tasks. The assignments were made in order to attain the three symmetrically related functions of each base function, see Fig. 1.

4.1.3. Rationale: experimental delimitations

The Kolmogorov theorem [58] states that any continuous function can be implemented by a 3LFNN [41,43], utilizing different nonlinear a–o transfer functions [19]. Other researchers in the field [42,43] have also stated that an ANN of at least three-layers is required to approximate a function f. Due to this constraint, 3LFNNs are utilized. Note, another body of research has stated that only one internal hidden layer (or a two-layer) ANN is necessary if sigmoidal a–o functions are employed [19,45,46]. However, our work also enables us to validate the methodology on more than one internal hidden layer. The following paragraphs discuss the ANN topologies, training set, and methods and parameters used to train the different networks. The error metric used and the results visualization methodology are then presented.

4.1.4. Topology of base and DNs

The two distinct regression tasks (single- and multi-valued) required different types of network topology. The o–o and m–o regression tasks were mapped to 3LFNNs. The inverse multi-valued regression task was mapped to a 3LRNN, similar to Elman's [21] except the output units were used as the context units (i.e. to attain the delayed previous state). The 3LRNN was termed an *output–input* 3LRNN by Mori and Ogasawara [72] when they examined 'four recurrent neural networks.' The reason for utilizing the output–input 3LRNN was related to the ease of reconfiguring a 3LFNN into a 3LRNN. The output–input RNN only requires a context layer between the output and inputs. Hence, for a 1NN1 (3LRNN) net it would require only one context unit, whereas an Elman's 3LRNN would require n^H context units if the number of units in the hidden layer was n^H. The *output–input* 3LRNN was utilized for the multi-valued task, as feedforward and time delayed nets could not accurately map the inverse multi-valued regression tasks, $f^{-1}(x)$.

The topology of a 3LFNN had a single input which was connected to eight 1-input sigmoidal hidden units, which were in turn connected to eight 8-input sigmoidal hidden units, which were then connected to one 8-input linear visible unit, i.e. a 1–8–8–1 network. The 3LRNN was assigned the same basic topology m–8–8–1, with m inputs. In order to try to attain a 1% error (i.e. 99% accuracy) the 3LRNN augmented m inputs were composed of two sequential inputs (one delayed input) and a delayed output vector $x_i \in \{x_n, x_{n-1}, y_{n-1}\}$. However, these nets achieved only a 1.5% accuracy (precision). To achieve the required 1% accuracy, a methodology termed an "expansion of internal state-space" [78] was utilized. Refer to Appendix B for more details. This methodology was used because even nets with nine, ten, or more hidden units (with 1-cubes in the first hidden layer) could not achieve the required accuracy.

4.1.5. Size of the BN training set

The BN 3LFNN 1–8–8–1 net was trained with a set of eleven ($n = 11$ e.g. *the size of the training set*) input–output, $\{x;y\}$, pattern pairs, $\{x_i; y_i^t\}$, distributed evenly across the input space, whereas the DTN 3LRNN 3–8–8–1 net was trained with a set of twenty ($n = 20$ e.g. *the size of the training set*) input–output pattern tuples, $\{x_n, x_{n-1}, y_{n-1}; y_n^t\}$, distributed evenly across the input space.

4.1.6. Training method and parameters of nets

Both the BN and DTN 3LFNN and the DTN 3LRNN were trained for 50,000 epochs with the following back-propagation rule learning rate and momentum settings: $\alpha = 0.6$ for the BN and DTN 3LFNN (and $\alpha = 1.0$ for the DTN 3LRNN), $\lambda = 0.99$; their outputs were linear a–o functions. Their hidden units used sigmoidal a–o functions with a $\rho = 0.1$ (i.e. defines the slope of the sigmoid).

The training patterns for the BN and DTN 3LFNN networks were presented sequentially to the network, and each pattern was selected at random from the set of training vectors, where x_i is a set of inputs which map to a set of outputs, y_i^t. The DTN 3LRNN was trained using

truncated backpropagation [21]. Truncated implied that the y_{n-1} was regarded as an extra input [9]. The 3LRNN stored the previous input/output states $\{x_{n-1},y_{n-1}\}$, and added storage delays (Z^{-1}), which affected the feedforward phase. The backpropagation phase (inside the net) was performed without delays. The training patterns for the DTN 3LRNN networks were presented sequentially to the network, and each pattern was selected sequentially from the set of training vectors, where $x_i \in \{x_n, x_{n-1}, y_{n-1}\}$ was a set of inputs which map to a set of outputs, y_n^t. The three inputs connected to eight hidden units were initially 3-cube units, $m=3$. In order to achieve a 1% error the number of input lines to the first layer of hidden units,H_1, was expanded to three inputs per x_i, which effectively expanded the 3-cube to 9-cube units. This meant the input probability distribution, z_i, oversampled the input probability, x_i, to attain higher mapping accuracy. Refer to Appendix B for a more detailed description.

4.1.7. A methodology for distance measurement and results visualization

To compare the symmetrically related functions, f^r, that the BN, DNs and DTNs map, a mean ranked Euclidian distance metric, $\overline{D}_{rd}$, is used, see Section 3.2.6. This elevates the problem of comparing error metrics with respect to a training set and a validation set, which is a norm when evaluating the generalization ability of an ANN [34]. It also aligns with our original premise that only one set of training vectors is required to train the BN; this was a prerequisite for our methodology and framework. The following experimental research considers a set of possible symmetrical transformations. The transformations are performed singly (CCS^{-1}) or as composites (SCS^{-1}). The aims are to:

(i) show that the functions the transformed DN and DTN map are isomorphic (symmetric) to the base function $f^b = f(x)$.
(ii) analyse the nets and the regression tasks they map and state which conform to specific generalization theory axioms (i.e. orders).
(iii) categorize the types of generalization order each ANN performs.

Note that only those with a $\overline{D}_{rd} \leqslant 0.01$ are considered isomorphic, or statistically similar, and hence can even be considered for certain orders of generalization.

To validate that the trained BN mapped the base functions, a root MSE e_{rms} was calculated:

$$e_{\text{rms}} = 1/n \sum_{p=1}^{p=n} \sqrt{(y_i^t - y_i)^2} = 1/N_p \sum_{p=1}^{p=N_p} |y_i^t - y_i|, \tag{16}$$

where n is the number of training or N_p test (or validation) vectors, y_i^t the target output and y_i the actual output, and $(\cdot)^2$ is the squared difference term. The error was summed over $n=11$ 3LFNN and $n=20$ 3LRNN examples or target vectors utilized for the BN. The test results depict the 0° and 2° points as circles, and the 1° and 3° points as squares, see Figs. 6–9. The graph plots the node's output, y, on the vertical axis against the node's input, x, on the horizontal axis, see Fig. 10.

4.1.8. Experimental work: validating that BN performs 0° and 1° generalization

Validation of the BN took place after training. This validated that the BN performed 0° generalization. The networks were trained on a set of stimuli and a prerequisite was that they must correctly map their training data. Fig. 6A and C depicts the results of the 0° generalization test of the BN; the resultant accuracies are depicted in Table 4.

After it had been trained, the BN was then tested with a set of validation vectors [34], to check that it performed 1° generalization, Fig. 6B and D. The validation set was a set of unseen vectors. To validate that the BN performed 1° generalization, each function (o–o or m–o) utilized a set of validation input vectors x_i^v and a set of output validation vectors y_i^v. The validation set was derived from the training set. The validation data points were deduced by dividing the input interval between the training set points in half. Hence, if the input training vector's interval was

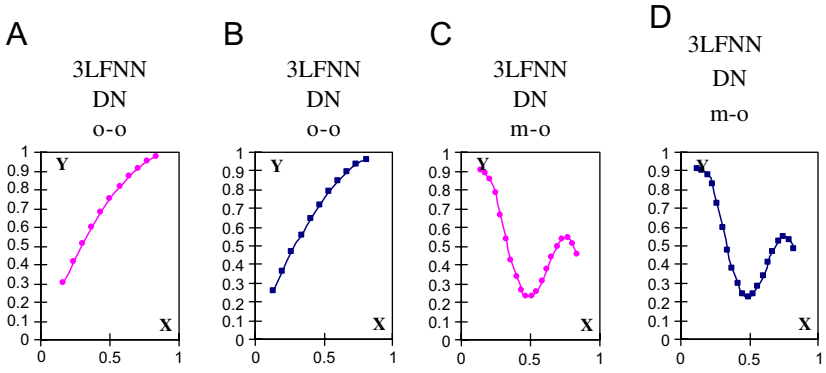


Fig. 8. 2° (A, C) and 3° (B, D) generalization test of the $f^{180^\circ}(x)$ generated by a DN utilizing transformation $T(CCS^{-1}) \Rightarrow f^{\mapsto 0}(f^{\perp H})$.

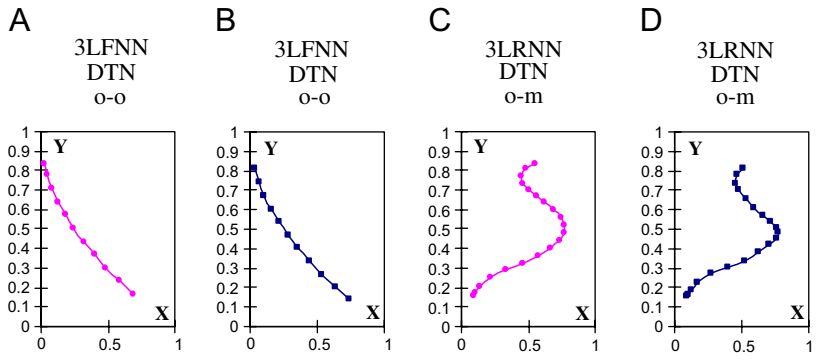


Fig. 9. 2° (A, C) and 3° (B, D) generalization test of the $f^{270^\circ}(x)$ generated by a DTN utilizing $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$ vectors, derived using transformation $T(SCC)\Rightarrow$.

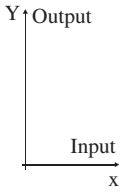


Fig. 10. Axis information for graphs displayed in the experimental work.

Table 4
Accuracy of the BN's function mapping regression task

Regression task	3LFNN e_{rms} 0°G	3LFNN e_{rms} 1°G
$f(x)=x^2:\{x\in\Re^+ \text{ and } x\in[0,1]\}$	0.000519	0.001395
$f(x)=-91x^6+329x^5-409x^4+209x^3-37x^2-0.8x+1:$ $\{x\in\Re^+ \text{ and } x\in[0,1]\}$	0.004096	0.006158

$\{x_1^t, x_2^t\}=\{0.14, 0.22\}$ then the validation test point would have been $\{x_1^v\}=\{0.18\}$. These points tested that the net was performing interpolation. The final validation point was outside the input vector interval, and as such tested the net's extrapolation capability, i.e. $\{x_9^t, x_{10}^t\}=\{0.78, 0.86\}$ then $\{x_{10}^v\}=\{0.9\}$. These tests validated that the BN performed 0° and 1° generalization as they are both less than our required 1% accuracy ($e_{\text{rms}}<0.01$, re. Section 4.1.4).

4.1.9. Experimental work: validating that DNs perform 2° and 3° generalization

This took the form of three separate validation tests.

Validation Test I

The initial test validated that the first DTNs mapped the $f^{90^\circ}(x)$ function generated by a DTN utilizing $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$ vectors derived using transformation $T(CCS^{-1})\Rightarrow f^{\mapsto 0}$ and thus performed 2° and 3° generalization. Fig. 7A and C depicts the 2° generalization results and Fig. 7B and D the 3° generalization results. Both conform to our requirement of $\overline{D}_{rd}\leqslant 0.01$ which implies that they are statistically similar and hence are related via symmetry.

Validation Test II

The next test validates that the second DNs mapped the $f^{180^\circ}(H)$ function generated by a DN utilizing transformation $T(SCS^{-1})\Rightarrow f^{\mapsto 0}(f^{\perp H})$ and thus performed 2° and 3° generalization. See Fig. 8A and C and Fig. 8B and D.

Validation Test III:

The final test validated that the third DN mapped the $f^{270^\circ}(x)$ function generated by a DTN utilizing $DG(x_i^{x\leftrightarrow y}, y_i^{x\leftrightarrow y})$ vectors, derived using transformation $T(SCC)\Rightarrow(f^{\perp H})$ and performed 2° and 3° generalization., see Fig. 9A and C and Fig. 9B and D.

5. Discussion and conclusions

This section will focus on the two different (single- and multi-valued) regression tasks undertaken by the 3LFNN and the 3LRNN. In particular, it will look at the accuracy attained by these two types of nets, that is to say how accurately they performed the function estimation regression tasks. In order to evaluate the methodology utilized to attain higher-order generalization, the accuracy of the networks (trained and derived) must be of the same order. Accuracy is aligned with a 1% root MSE accuracy ($e_{\text{rms}}<0.01$, re. Section 4.1.4) in the case of the BN and a mean ranked distance $\overline{D}_{rd}\leqslant 0.01$ (re. Section 3.2.6) which implies that the base functions f^b and the related functions f^r are statistically similar and hence are related via symmetry. The discussion first addresses the nets' mapping accuracy in Section 5.1.

5.1. Nets' mapping accuracy

The accuracy of the nets is discussed in relation to the experimental work which validated BN, DN and DTN performance. The accuracy of the DNs is depicted in Table 5. All the DNs and DTNs achieve the required accuracy and hence perform 2° and 3° generalization.

Table 5 tabulates the following columns:

Column I: particular nets and the regression tasks they perform.

Table 5
Accuracy of the derived nets

Compare nets and regression tasks	ST	o–o function $\overline{D}_{rd}$ 2°G	o–o function $\overline{D}_{rd}$ 3°G	m–o and o–m functions $\overline{D}_{rd}$ 2°G	m–o and o–m functions $\overline{D}_{rd}$ 3°G
Base net $f(x)$ and derived-trained net $f^{90^\circ}(x)$	CCS^{-1}	0.002571	0.001279	0.002698	0.009130
Base net $f(x)$ and derived net $f^{180^\circ}(x)$	SCS^{-1}	2.081668E−17	1.804112E−17	6.318045E−17	3.725090E−17
Base net $f(x)$ and derived-trained net $f^{270^\circ}(x)$	SCC	0.002565	0.001304	0.002959	0.004345
Mean $\overline{D}_{rd}$		0.001712	0.000861	0.001885	0.004491

Column II: the symmetry transforms performed on the BN in order to attain the DNs (and then the DN is utilized to train the DTN's) weights.

Column III: the measured mean ranked distance $\overline{D}_{rd}$ in relation to 2°G of the BN and DN (or DTN) o–o regression function mapping, given that the BN is stimulated by a set of training vectors.

Column IV: the measured mean ranked distance $\overline{D}_{rd}$ in relation to 3°G of the BN and DN (or DTN) o–o regression function mapping, given that the BN is stimulated by a set of validation vectors.

Columns V and IV: measure the $\overline{D}_{rd}$ for the m–o and o–m regression tasks; the description is the same as for columns III and IV.

5.1.1. Discussion of distance function measurements

In Section 3.2.6, a methodology was developed to measure statistical similarity of regression tasks performed by ANNs. The claim that ANNs can indeed perform higher-order generalization is validated by the fact that the regression tasks derived from transformation are "statistically similar." One of the ways of showing this alignment is to compare the original graph, b (mapped by the BN), with the transformed graph, d (mapped by the DN or DTN). Then, if the interpoint distance is preserved, the graphs are "statistically similar."

The empirical work validated that the BN's regression tasks are statistically similar to those of the DN and DTN. The empirical work investigated a selection of single and composite transformations, see Fig. 6 to 9A–D. They all achieved a mean ranked distance of equal to or less than 1% and hence this confirms that they are all "statistically similar"; see Table 5.

5.2. Relation to other research

The work presented relates to a network's ability to interpolate and extrapolate [20]. It supplements this research by quantifying the terminology used to describe these two effects (e.g. interpolate and extrapolate). Refer to Sections 3.1, 3.1.1/4, and 3.1.5. It could also be regarded as a methodology for improving generalization [59,76,104]. However, it does not improve lower-order generalization ability. It enables the information encapsulated in the nets performing the lower-order generalization task to be reused to instantiate weights of other nets that perform related tasks, and which perform higher-order generalization. The higher-order technical description, visualization, and logical predicates are a natural detailed extension to Gurney's [31] initial imprecise classification.

5.2.1. Comparison with existing approaches to the reuse of information in ANNs

The research presented in this paper relates to several existing connectionist research areas aligned with the reuse of information in ANNs and to three in particular. The closest field is that of *reuse of information*, typified by the work of Ghosh and others [10,28,84]. The second field uses *neural networks to perform mathematics* [50,67,96–98]. Finally, the third field is that of *hybrid systems* [103]. It is also directly linked to a fourth research field for solving functional equations using an approach which shares and inverts weights in multi-network systems [56].

5.3. Conclusion

The major contribution of this work is the development of a detailed systematic framework for higher-order generalization. In this paper, the theoretical methodology required to perform a type of 0°, 1°, 2°, and 3° generalization has been outlined. This type of generalization is of a more abstract nature than 0° or 1° generalization. A set of symmetry transformations (matrix transformations) was presented; these transformations were then applied to the weight array of the BN. In this way, the BN and the DNs were able to provide a process that enabled a family of ANNs to perform multiple regression or function estimation tasks. The empirical results support the claim that all the orders of generalization are achievable for ANNs that perform o–o, m–o and o–m regression tasks.

Geman et. al [26] identified a important issue which should always be considered when investigating issues related to extrapolation. They stated that, "… extrapolate, that is, generalize in a very nontrivial sense," which they then go onto qualify, "… since the training data will never 'cover' the space of all the points." In fact, by utilizing weight transformations, higher-order generalization does indeed 'cover' [some] of the space that the initial training data (points) did not cover.

This paper puts forward the conjecture that, "related isomorphic regression tasks can be performed by ANNs without extra training". The system presented has a set of DNs that uses a type of one-shot learning. It is termed one-shot because only one of the nets (i.e. the base) in the system was trained with a set of input–output vectors. The rest of the DN's prescribed knowledge is inherited from the *BN* and its associated weight set. The regression system is able to map related isomorphic functions; in this case, isomorphic implies the same type of isomorphic regression task but the function is reflected and/or scaled in different axes. Hence, one of the main contributions of this paper is the concept of information inheritance derived from the so-called BN. This leads to the premise that a type of higher-order generalization can be performed utilizing information inheritance by transformation of the weights obtained from the BN: a new neural net, called the DN, is then able to approximate an isometric function without any training at all.

The reason why the sigma–pi model is employed in this research is due to the procedure it uses to address its internal weights. Appendix A details how the inputs set up a probability distribution across the weights. This means that the activation is a weighted product, as each weight is multiplied by its probability of being addressed. This distribution samples a linear array of weights. It is the linear (or sequential) manner in which the internal weights are addressed that enables these transforms to be performed and, in fact, why they are implemented in hardware in RAMs. These transforms cannot be performed with standard semi-linear or classical neural units [87] as the weights are not addressed in this manner.

Generally speaking, a specific set of STs was investigated (S and S^{-1}), though these may not be the only transformations that can be performed on ANN weights. In fact, extra investigations into complex transformations were carried out, which investigated rotating the weights utilizing the standard rotation matrix $\Re_\theta$. However, after a few degrees rotation, the weights distorted and did not accurately estimate the rotated function. Hence, the transforms that enable inheritance of information are able to perform only a subset of all affine transformations. They can perform reflection (S and S^{-1}), dilation (D), and their combination (CCS^{-1}, SCS^{-1}, and SCC). However, these transformations appear to be restricted to isometric (or congruent) functions.

Finally, the results have been mathematically validated in this paper using a distance metric, $\overline{D}_{rd}$ (re. Section 3.2.6), to authenticate that the regression tasks undertaken by the BN are indeed symmetrically related to the DNs and DTNs.

5.4. Future research issues

In the future, the author intends to investigate different types of 3LRNN and their capabilities with respect to higher-order generalization. This would be advantageous, as this paper utilized only a particular type of Elman's 3LRNN [21] (or an *output–input* Elman 3LRNN [72]). Higher-order generalization utilizes symmetrical transformations which could be viewed as symmetry operations. The transformations are also related to a branch of mathematics concerned with equivalence relations. A relation has been defined as, "… two objects [are] equal in relation to some particular feature if both possess this feature" by Shubnikov and Koptsik [91]. Both symmetry operations and equivalence relations are thought to be an associated research domain worthy of investigation. In future papers we will investigate transformations on all layers and transformations on hierarchies of related nets.

Acknowledgements

I am grateful to Dr. Laurence Dixon for his comments while developing the mathematics and algorithms in this article. Dr. Laurence Dixon is an emeritus Professor at the University of Hertfordshire. I would also like to thank Mrs. Susan Watts who assisted us during the preparation of the paper and without whose dedication the final paper would not achieve its goals. I would also like to acknowledge all the help and advice given by the editor of the journal and all reviewers whom have helped in the evolution of this research and paper. Finally, I would thank Liping Zhao, a Lecturer (and valued colleague) at the University of Manchester, for her continuing help in evolving symmetry as a domain.

Appendix A. The sigma–pi neuron model

To understand how to perform transformations, it is necessary to explain the theory of sigma–pi units. The initial theory of sigma–pi units was first introduced in 1989 [31]. A reappraisal of sigma–pi networks was presented in 1999, in which the RAM's site-values [31] were viewed as weights [75].

The a–o function of a classical neural unit [87] is defined as $y = \sigma(\Sigma w_i x_i)$. The a–o function of a RAM [4,16,30,70] or sigma–pi neural unit [4,23,24,31–37] is defined as $y = \sigma(\Sigma w_\mu P_\mu)$. Sigma–pi units may be viewed as probabilistic neural networks. The two neural models are very similar. The similarities are that: (i) both utilize a sigmoidal output function σ; (ii) both use a–o functions $\sigma\Sigma w \times f(x)$; and (iii) both have a summation activation function $\Sigma w \times f(x)$.

The difference lies in the way the RAM neural unit addresses and stores its weights. In the case of the classical neural unit, each input, x, has associated with it a subscript, e.g. x_1, which connects that input to a given input line, i.e. input line 1. The weights, w, also have a subscript associating them with a given input line, i.e. w_1 connects the weight to input line 1.

RAM or sigma–pi neural units however do not associate a weight with a given input line. They use the set of input variables $\{x_1, x_2, x_3\}$ as an index to a sigma–pi memory

array. For ease of understanding, the inputs are interpreted as binary $x \in \{0,1\}$. If, for example, the input address is $\{x_1, x_2, x_3\} = \{1,0,1\}$ then the weight addressed in the sigma–pi neuron unit is $w_\mu = w_{101}$, where the input address, μ, is the weight index to the sigma–pi memory array. This does not mean that these units cannot be used with real-valued inputs. We can utilize the analogue or A-model [31], which enables these units to input and output real-values or analogue values. Alternatively, the inputs can define the probability of addressing the weights as in the case of the stochastic or S-model of [31]. The S-model uses analogue inputs $0 \leqslant x_i \leqslant 1$ to define the probabilities of accessing the weights. Then the probability that each weight is addressed is defined by calculating the probability $\{P_{000}, \ldots, P_{111}\}$. Thus, in the case of a three-input sigma–pi unit the probability of addressing weight index $\mu = \{\mu_1,\mu_2,\mu_3\} = \{0,0,0\}$ is $P_\mu = P_{\{\mu_1,\mu_2,\mu_3\}}$ and $P_{000} = (1-x_1) \times (1-x_2) \times (1-x_3)$. When the inputs are set to $\mu = \{\mu_1,\mu_2,\mu_3\} = \{0,0,1\}$ the probability is $P_{001} = (1-x_1) \times (1-x_2) \times (x_3)$, and when the inputs are set to $\mu = \{\mu_1,\mu_2,\mu_3\} = \{1,1,1\}$ the probability is $P_{111} = (x_1) \times (x_2) \times (x_3)$. In the case of a one-input sigma–pi unit, P_0, the probability is that the input line takes on the value zero and P_1 is the probability that the input line takes on the value one. Subscript μ in w_μ is a pointer (index) to a specific weight in the weight matrix. The input stimuli or input variables to a network define the probability of addressing each weight, P_μ.

To illustrate how a real-valued input can define a binary weight address consider a one-input unit with an input variable, $x = 0.25$. The single input addresses 2^n weights in the sigma–pi's memory array. Given $n = 1$ the weight array contains $2^1 = 2$ weights, e.g. $w_{\mu=0}$ and $w_{\mu=1}$. Then, the probabilities of addressing the two weights are $P_0 = (1-x) = (1-0.25) = 0.75$ and $P_1 = (x) = (0.25) = 0.25$. The probability matrix uses the real-valued inputs to calculate the probability of addressing each weight. The probability matrix may be viewed as a means of sampling the weight matrix. Each weight then makes a $W_\mu \times P_\mu$ contribution to the activation $\Sigma W_\mu \times P_\mu$. The output equation implies that the probabilities, P_μ, may be perceived as a means of setting up a probability sampling profile across the weights W_μ. The probability sampling profile defines the amount each weight, W_μ, adds to the activation. Only those weights which were sampled, dependent on the input values, were then used to define the unit's activation. When training these units, the sample profile defines which weights were adapted by the training regime.

In order to apply matrix transformations to sigma–pi networks, their functionality must be viewed as a matrix equation:

$$Y = \sigma(W \cdot P), \tag{A.1}$$

where "$\cdot$" denotes the matrix product, sometimes called inner product $W \cdot P$, or the dot product. The "$\cdot$" is also known as the scalar product since its result is a number, rather than a vector. In the case of a single unit, W is a row matrix of the weights:

$$W = [w_1, \ldots, w_s] \tag{A.2}$$

and P is a column matrix of the probabilities of the weights being addressed:

$$P = \begin{bmatrix} P_1 \\ \bullet \\ \bullet \\ \bullet \\ P_n \end{bmatrix}, \tag{A.3}$$

where $\mu \in \{1, \ldots, n\}$ is the index and σ a sigmoid function:

$$Y = \sigma(U) = 1/(1 + e^{-U/\rho}), \tag{A.4}$$

given ρ defines the shape of the sigmoid, where as $\rho \to 0.0$ the a–o function approaches a step function, which is analogous to the threshold logic function or Heaviside function. $\rho \to 0.5$ would make the function the more normal semi-linear or squash function. U is equal to the scalar product, $W \cdot P$. Given $W \cdot P$ is the matrix scalar dot product of the units' weights, w_μ, dot their associated probabilities, P_μ, of being addressed which is

$$[w_1 \quad \ldots \quad w_n] \cdot \begin{bmatrix} P_1 \\ \bullet \\ \bullet \\ \bullet \\ P_n \end{bmatrix} = w_1 P_1 + \cdots + w_n P_n \tag{A.5}$$

and this may be reinterpreted as

$$w_1 P_1 + \cdots + w_n P_n = \sum_{\mu=1}^{n} w_\mu P_\mu. \tag{A.6}$$

Appendix B. Expansion of internal state-space of sigma–pi units

In order to relate how the sigma–pi unit's internal state-space may be expanded, the dynamics of the unit need to be understood. The units themselves are considered and the state variables of each unit are defined as a function of the weights $F\left(w_{\mu_{(i)}}\right)$, then the point in an N-dimensional state-space where the unit is presently located is a function of the weights. The size of each unit's internal state-space is a function of the number of inputs to that unit. The basic dynamics of a sigma–pi unit is related first. A single n-cube (sigma–pi unit) when presented with a binary input stimulus instantaneously addresses only one weight for each feedforward pass. Real-valued inputs to the sigma–pi unit address multiple weights. These are required to estimate the net's real-valued output. Hence, the sigma–pi unit effectively *weights* each weight in proportion to the probability of the weight being addressed, $w_\mu \times P_\mu$. Refer to Appendix B, reference equations (B1), (B.5), and (B6). If one considers a 1-cube (a 1-input sigma–pi unit) it has two

possible weights, $\{w_0,w_1\}$. The actual weight address is selected probabilistically in the case of the sigma–pi unit.

In the following example the weights are viewed as a set of quantized levels. This is done for analysis only, and can be generalized to the case where the weights are real-valued and continuous over a bounded interval. The possible combinations of weight addresses in a *1*-cube are defined as the combination ${}^2C_1$ of one possible weight out of two. Each weight has the possible combination of states of ${}^{2w_m+1}C_1$, which is one possible state out of $2w_m+1$ states. We interpret the weights as ranging over a set of discrete levels $w_m \in \{-2,\ldots,+2\}$, where w_m is represented in polarized notation. This means the size of the 1-cube's internal state-space is defined as

$$ {}^2C_1 \times {}^{2w_m+1}C_1 = \frac{2!}{1!(2-1)!} \times \frac{(2w_m+1)!}{1!(2w_m+1-1)!} = 2 \times (2w_m+1). \quad \text{(B.1)} $$

In the case of a 2-cube (2-input sigma–pi unit) the size of the internal state-space is

$$ 2^2 \times (2w_m+1). \quad \text{(B.2)} $$

Then for the *n*-cube (*n*-input sigma–pi unit) case the size of the internal state-space is

$$ 2^n \times (2w_m+1). \quad \text{(B.3)} $$

One may take advantage of this if a unit is not able to communicate the required bandwidth of information by increasing the size of the unit's internal state-space. For example if a 1-cube is required to learn a given function to a given accuracy and it does not meet the accuracy requirement, then one enlarges the unit's internal state-space by increasing the number of inputs to the unit. One should of course note that all the inputs are connected to the same input state vector x_i. Each input to the unit has the effect of providing a set of probabilities $P(\mu) = 1/2^n \prod_{i=1}^{i=n}(1+\overline{\mu}_j z_i)$ that effectively sample the weights w_μ. Hence, if we add one more input to the original unit, it becomes a 2-cube and the size of its internal state-space has doubled. If we repeat the process by adding one more input to the unit it becomes a 3-cube and the size of its internal state-space has increased by $2^{(3-1)}$ times that of the original. This can be viewed as over sampling the weights, all be it an expanded set of weights.

This means sigma–pi units have the ability to expand their internal state-space in order to learn functions to a required level of accuracy. This is shown to be the case in the experimental work in Section 11 [78] (re. Tasks 7, 8 and 9 which show the effects of expanding the hidden units' internal state-space). In the example, the final training accuracies were $e_{\mathrm{rms}}|_{1-input} = 0.012$, $e_{\mathrm{rms}}|_{2-input} = 0.0068$, and $e_{\mathrm{rms}}|_{3-input} = 0.006$.

The above expansion of internal state-space is not the same as increasing the number of connection weights to a semi-linear [87] unit, as the input would then have '*n*' weights connected to it. This means that '*n*' weights are utilized to obtain an instantaneous output of the semi-linear unit. While in the case of a cubic node, a set of weights is sampled, i.e. 1-input implies $2^n = 2^1 = 2$ weights, 2-input implies $2^n = 2^2 = 4$ weights, and n-input implies 2^n weights. Remembering that as $n \to \infty$ the sigma–pi units (single-cube) would be split into multiple-cube units (MCUs) to enable each to address a subset of the input state-space [31,32,34–37].

References

[1] M. Abramowitz, I.A. Stegun, Inverse Circular Functions, in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, ninth printing, (Section 4.4), Dover, New York, 1972.

[2] W.V. Anshelevich, B.R. Amirikian, A.V. Lukashin, M.D. Frank-Kamenetskii, On the ability of neural networks to perform generalization by induction, Biol. Cybern. 61 (1989) 125–128.

[3] A. Atiya, J. Chuanyi, How initial conditions affect generalization performance in large networks, IEEE Trans. Neural Networks 8 (1997) 1045–1052.

[4] J. Austin, A review of RAM based neural networks, in: The Presentation at the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, Turin, Italy, 1994.

[5] E.B. Baum, D. Haussler, What size net gives valid generalization?, Neural Comput. 1 (1989) 151–160.

[6] G. Bebis, M. Georgiopoulos, Improving generalization by using genetic algorithms to determine the neural network size, in: The Presentation of the Conference Record of Southcon/95, Fort Lauderdale, FL, USA, 1995.

[7] G. Birkhoff, S. Mac Lane, A Survey of Modern Algebra, vol. 12th printing, The Macmillan Company New York, 1963.

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

[9] M. Bodén, A guide to recurrent neural networks and back-propagation, SICS, Kista, Sweden, 2001.

[10] K.D. Bollacker, J. Ghosh, Knowledge reuse in multiple classifier systems, Pattern Recogn. Lett. 18 (1997) 1385–1390.

[11] B. Chakraborty, Effect of a sparse architecture on generalization behaviour of connectionist networks: a comparative study, in: The Presentation of the 1999 IEEE International Conference on Systems, Man, and Cybernetics, Tokyo, Japan, 1999.

[12] P. Chandra, Y. Singh, Regularization and feedforward artificial neural network training with noise, in: The Presentation of the IEEE 2003 International Joint Conference on Neural Networks, Double-tree Hotel-Jantzen Beach, Portland, Oregon, USA, 2003.

[13] P.C.Y. Chen, J.K. Mills, A methodology for analysis of neural network generalization in control systems, in: The Presentation at the Proceedings of the 1997 American Control Conference, Albuquerque, NM, USA, 1997.

[14] P.C.Y. Chen, J.K. Mills, Neural network generalization and system sensitivity in feedback control systems, in: The Presentation of the Proceedings of the 1997 IEEE International Symposium on Intelligent Control, Istanbul, Turkey, 1997.

[15] M.H. Christiansen, Improving learning and generalization in neural networks through the acquisition of multiple related functions, in: The Presentation of the Proceedings of the Fourth Neural Computation and Psychology Workshop, 1998.

[16] T. Clarkson, D. Gorse, J. Taylor, C. Ng, Learning probabilistic RAM nets using VLSI structures, IEEE Trans. Comput. 6 (1992) 1552–1561.

[17] J. Coplien, L. Zhao, Symmetry breaking in software patterns, in: Springer Lecture Notes in Computer Science, series no. 2177, Springer, Berlin, 2001, pp. 37–56.

[18] H.S.M. Coxeter, S.L. Greitzer, Geometry Revisited, Mathematical Association of America, Washington, DC, 1967.

[19] G. Cybenko, Approximation by superposition of a sigmoidal function, Math. Control, Signals, Syst. 2 (1989) 303–314.
[20] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, Large automatic learning, rule extraction, and generalization, Complex Syst. 6 (1987) 877–922.
[21] J.L. Elman, Finding structure in time, Cogn. Sci. 14 (1990) 179–211.
[22] H. Elsimary, S. Mashali, S. Shaheen, Generalization ability of fault tolerant feedforward neural nets, in: The Presentation of the IEEE International Conference on Systems, Man and Cybernetics—'Intelligent Systems for the 21st Century', Vancouver, BC, Canada, 1995.
[23] A. Ferguson, Learning in RAM-Based Artificial Neural Networks, Department of Engineering, University of Hertfordshire, Hatfield Campus, College Lane, Hatfield, Herts, UK, 1995.
[24] A. Ferguson, L.C. Dixon, H. Bolouri, Learning algorithms for RAM-based neural networks, Ann. Math. Artif. Intell. (1996).
[25] L. Franco, S.A. Cannas, Generalization properties of modular networks: implementing the parity function, IEEE Trans. Neural Networks 12 (2001) 1306–1313.
[26] S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma, Neural Comput. 4 (1992) 1–58.
[27] A.A. Ghorbani, K. Owrangh, Stacked generalization in neural networks: Generalization on statistically neutral problems, Presented at the Proceedings of IJCNN.01 International Joint Conference on Neural Networks, Washington, DC, USA, 2001.
[28] J. Ghosh, A.C. Nag, Knowledge enhancement and reuse with radial basis function networks, in: The Presentation at the 2002 IEEE World Congress on Computational Intelligence, Hilton Hawaiian Village Hotel, Honolulu, Hawaii, 2002.
[29] G.L. Giles, G.W. Omlin, Pruning recurrent neural networks for improved generalization performance, IEEE 5 (1994) 848–851.
[30] D. Gorse, J. Taylor, Training strategies for probabilistic RAMs, Parallel Process. Neural Syst. Comput. (1990) 161–164.
[31] K. Gurney, Learning in nets of structured hypercubes, Ph.D., Department of Electrical Engineering, Uxbridge, Brunel, UK, 1989.
[32] K. Gurney, Weighted nodes and RAM-nets: a unified approach, J. Int. Syst. 2 (1992) 155–185.
[33] K. Gurney, Training nets of stochastic units using system identification, Neural Networks 6 (1993) 133–145.
[34] K. Gurney, An Introduction to Neural Networks, University College London Press, UCL, UK, 1997.
[35] K. Gurney, Information processing in dendrites: I. Input pattern generalisation, Neural Networks 14 (2001) 991–1004.
[36] K. Gurney, Information processing in dendrites: II. Information theoretic complexity, Neural Networks 14 (2001) 1005–1022.
[37] K.N. Gurney, Training nets of hardware realisable sigma–pi units, Neural Networks 5 (1992) 289–303.
[38] F. Harary, Graph Theory, Addison Wesley Publishing Company, Reading, MA, 1994, pp. 161.
[39] J.W. Harris, H. Stocker, Handbook of Mathematics and Computational Science, Springer, New York, 1998.
[40] S.S. Haykin, Neural Networks: A Comprehensive Foundation, second ed, Prentice Hall, 1998.
[41] R. Hecht-Nielsen, Kolmogorov's mapping neural network existence theorem," in: The Presentation of the Proceedinsg of the IEEE First International Conference on Neural Networks, San Diego, USA, 1987.
[42] R. Hecht-Nielsen, Theory of backpropagation neural network, in: The Presentation of the Proceedings of the International Joint Conference on Neural Networks, New York, USA, 1989.
[43] R. Hecht-Nielsen, Neurocomputing: The Technology of Non-Algorithmic Information Processing, Addison-Wesley, New York, 1990.
[44] S.B. Holden, P.J.W. Rayner, Generalization and learning in Volterra and radial basis function networks, in: The Presentation at the IEEE International Conference on Acoustic, Speech, and Signal Processing, San Francisco, CA, USA, 1992.
[45] K. Hornik, Some new results on neural network approximation, Neural Networks 6 (1993) 1069–1072.
[46] K. Hornik, M. Stinchombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Networks 3 (1990) 551–560.
[47] G. Horvath, T. Szabo, CMAC neural network with improved generalization property for system modeling, in: The Presentation of the Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference, 2002.
[48] S.J. Huang, S.N. Koh, H.K. Tang, Image compression and generalization capabilities of backpropagation and recirculation networks, in: The Presentation at the IEEE International Symposium on Circuits and Systems, Singapore, 1991.
[49] K. Hyeoncheol, F. LiMin, Generalization and fault tolerance in rule-based neural networks," Presented at the 1994 IEEE International Conference on Computational Intelligence, Orlando, FL, USA, 1994.
[50] J.L.S. Jang, S.-Y. Lee, S. Shin, An optimization network for matrix inversion, in: The Presentation at the Neural Information Processing Systems, 1988.
[51] S.A. Janowsky, Pruning versus clipping in neural networks, Phys. Rev. 'A' 39 (1989).
[52] A. Jeffrey, Inverse Trigonometric and Hyperbolic Functions, Handbook of Mathematical Formulas and Integrals, second ed, Academic Press, Orlando, FL, 2000.
[53] C. Ji, Is the distribution-free sample bound for generalization tight? in: The Presentation of the IJCNN'92 International Joint Conference on Neural Networks, Baltimore, MD, USA, 1992.
[54] R. Kamimura, Improving generalization by teacher-directed learning, in: The Presentation of the IEEE 2003 International Joint Conference on Neural Networks, Doubletree Hotel-Jantzen Beach, Portland, Oregon, USA, 2003.
[55] R. Kamimura, Teacher-directed information maximization: supervised information-theoretic competitive learning with Gaussian activation functions, in: The Presentation of the IEEE IJCNN'04, 2004 International Joint Conference on Neural Networks, Budapest, Hungary, 2004.
[56] L. Kindermann, A. Lewandowski, P. Protzel, A framework for solving functional equations with neural networks, in: The Presentation at the Eighth International Conference on Neural Information Processing, Shanghai, 2001.
[57] K. Knopp, Multiple-Valued Functions (Section II), Theory of Functions Parts I and II, Two Volumes Bound as One, Dover, New York, 1996.
[58] A.N. Kolmogorov, On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition, Dokl. Akad. Nauk SSSR 114 (1957) 369–373.
[59] J.K. Kruschke, Improving generalization in backpropagation networks with distributed bottlenecks, in: The Presentation at the 1989 IJCNN International Conference on Neural Networks, Washington, DC, USA, 1989.
[60] I. Kushchu, Learning, evolution and generalisation, in: The Presentation at The 2003 Congress on Evolutionary Computation, CEC '03, 2003.
[61] J. Larsen, A generalization error estimate for nonlinear systems, in: The Presentation of the Proceedings of the 1992 IEEE-SP Workshop on Neural Networks for Signal Processing, Helsingoer, Denmark, 1992.
[62] J. Larsen, L.K. Hansen, Generalization performance of regularized neural network models, in: The Presentation of the Proceedings of the 1994 IEEE Workshop on Neural Networks for Signal Processing, Ermioni, Greece, 1994.
[63] S. Lawrence, A.C. Tsoi, N.J. Giles, Local minima and generalization, in: The Presentation at the IEEE International Conference on Neural Networks, Washington, DC, USA, 1996.
[64] T.S. Lin, J. Meador, Classification-accuracy monitored backpropagation, in: The Presentation of the ISCAS'92—IEEE International Symposium on Circuits and Systems, San Diego, CA, USA, 1992.

[65 B. Liskov, Data Abstraction and Hierarchy, in: The Presentation of the Addendum to the Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Orlando, 1988.
[66] E. Littmann, H. Ritter, Generalization Abilities of Cascade Network Architectures, Kaufmann, Morgan, 1993.
[67] F.L. Luo, B. Zheng, Neural network approach to computing matrix inversion, Appl. Math. Comput. 47 (1992) 109–120.
[68] D.J.C. MacKay, A practical Bayesian framework for backpropagation networks, Neural Comput. 4 (1992) 448–472.
[69] D.J.C. MacKay, Bayesian interpolation, Neural Comput. 4 (1992) 415–447.
[70] D. K. Milligan, Annealing in RAM-based learning networks, Technical Memorandum CN/R/142, Brunel University, Uxbridge, West London, Middlesex, UK, 1988.
[71] J.E. Moody, Note on generalization, regularization and architecture selection in nonlinear learning systems," in: The Presentation of the Proceedings of the 1991 IEEE Workshop on Neural Networks for Signal Processing, Princeton, NJ, USA, 1991.
[72] H. Mori, T. Ogasawara, A recurrent neural network approach to short-term load forecasting in electric power systems, in: The Presentation at the The World Congress on Neural Networks, WCNN-93, Portland, Oregon, USA, 1993.
[73] S. Mukherjee, P. Niyogi, T. Poggio, R. Rifkin, Statistical learning: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization, Center for Biological Computation and Learning, Artificial Intelligence Lab, Brain Science Department, Massachusetts Institute of Technology, Department of Computer Science and Statistics, January 2004.
[74] M. Negnevitsky, Artificial Intelligence—A Guide to Intelligent Systems, Addison-Welsley, Edinburgh, 2002.
[75] R.S. Neville, S. Eldridge, Transformations of sigma–pi nets: obtaining reflected functions by reflecting weight matrices, Neural Networks 15 (2002) 375–393.
[76] R. Neville, J. Stonham, Generalisation in sigma–pi networks, Connect. Sci.: J. Neural Comput., Artif. Intell. Cogn. Res. 7 (1995) 29–60.
[77] R.S. Neville, T.J. Stonham, Adaptive critic for sigma–pi networks, Neural Networks 9 (1996) 603–625.
[78] R.S. Neville, T.J. Stonham, R.J. Glover, Partially pre-calculated weights for the backpropagation learning regime and high accuracy function mapping using continuous input RAM-based sigma–pi nets, Neural Networks 13 (2000) 91–110.
[79] H. Ninomiya, A. Sasaki, A study on generalization ability of 3-layer recurrent neural networks, in: The Presentation of the IJCNN'02 Proceedings of the 2002 International Joint Conference on Neural Networks, Honolulu, HI, USA, 2002.
[80] H. Ogawa, E. Oja, Optimally generalizing neural networks, in: The Presentation of the 1991 IEEE International Joint Conference on Neural Networks, Singapore, 1991.
[81] D.A. Pados, P. Papantoni-Kazakos, A note on the estimation of the generalization error and prevention of overfitting, in: The Presentation of the IEEE World Congress on Computational Intelligence, 1994 IEEE International Conference on Neural Networks, Orlando, FL, USA, 1994.
[82] D.S. Phanak, Relationship between fault tolerance, generalization and the Vapnik–Chervonenkis (VC) dimension of feedforward ANNs, in: The Presentation of the JCNN'99 International Conference on Neural Networks, Washington, DC, USA, 1999.
[83] T. Poggio, R. Rifkin, S. Mukherjee, P. Niyogi, General conditions for predictivity in learning theory, Nature 428 (2004) 419–422.
[84] L.Y. Pratt, Experiments on the transfer of knowledge between neural networks, in: R. Rivest (Ed.), Computational Learning Theory and Natural Learning Systems, Constraints and Prospects, vol. 19, MIT Press, 1994, pp. 523–560.
[85] M.F. Redondo, C.H. Espinosa, Generalization capability of one and two hidden layers, in: The Presentation of the IJCNN'99 International Conference on Neural Networks, Washington, DC, USA, 1999.
[86] R. Reed, Pruning algorithms: a survey, IEEE 4 (1993) 740–747.
[87] D.E. Rumelhart, J.L. McClelland, a.t.P.R. Group, Parallel Distributed Processing, MIT Press, Cambridge, MA, USA, 1986.
[88] S. Russell, P. Norvig, Artificial Intelligence A Modern Approach, Prentice-Hall Inc., Englewood Cliffs, NJ, 1995.
[89] D. Sarkar, Randomness in generalization ability: a source to improve it?, in: the Presentation of the 1994 IEEE World Congress on Computational Intelligence, IEEE International Conference on Neural Networks, Orlando, FL, USA, 1994.
[90] W.S. Sarle, Stopped training and other remedies for overfitting, in: The Presentation at the Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics, Convention Center and Vista Hotel, Pittsburgh, PA, 1995.
[91] A.V. Shubnikov, V.A. Koptsik, Symmetry in science and art, in: G.D. Archard (Ed.), Translated from Russian, Plenum Press, New York, 1974.
[92] S. Sigurdsson, J. Larsen, L.K. Hansen, On comparison of adaptive regularization methods, in: The Presentation of the Proceedings of the 2000 IEEE Signal Processing Society Workshop on Neural Networks, Sydney, NSW, Australia, 2000.
[93] T. Szabo, G. Horvath, Improving the generalization capability of the binary CMAC, in: The Presentation of the IJCNN 2000 Proceedings of the IEEE–INNS–ENNS International Joint Conference on Neural Networks, Como, Italy, 2000.
[94] M.J. Turmon, Assessing generalization of feedforward neural networks, Doctoral Thesis, Cornell University, 1995, pp. 88.
[95] Y. Wada, M. Kawato, Estimation of generalization capability by combination of new information criterion and cross validation, in: The Presentation of the IJCNN'91 International Joint Conference on Neural Networks, Seattle, WA, USA, 1991.
[96] J. Wang, A recurrent neural network for real-time matrix inversion, Appl. Math. Comput. 5 (1993) 89–100.
[97] L. Wang, J.M. Mendel, Structured trainable networks for matrix algebra, IEEE Int. Joint Conf. Neural Networks 42 (1990) 125–128.
[98] L. Wang, J.M. Mendel, Parallel structured networks for solving a wide variety of matrix algebra problems, J. Parallel Distrib. Comput. 14 (1992) 236–247.
[99] M. Wann, T. Hediger, N.N. Greenbaum, The influence of training sets on generalization in feed-forward neural networks, in: The Presentation at the IJCNN'90 International Joint Conference on Neural Networks, San Diego, CA, USA, 1990.
[100] E. Watanabe, H. Shimizu, A learning algorithm for improving generalization ability of multi-layered neural network for pattern recognition problem, in: The Presentation of the 1994 IEEE World Congress on Computational Intelligence, IEEE International Conference on Neural Networks, Orlando, FL, USA, 1994.
[101] Weigend A.S., Rumelhart D.E., Huberman B.A., Generalization by weight-elimination with application to forecasting, in: The Presentation of the Proceedings of the 1990 conference on Advances in neural information processing systems, Denver, Colorado, United States, 1990.
[102] P.J. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Harvard University, Cambridge, MA, 1974.
[103] S. Wermter, R. Sun, An overview of hybrid neural systems, in: S. Wermter, R. Sun (Eds.), Hybrid Neural Syst., Springer, 2000, pp. 1–13.
[104] D. Whitley, N. Karunanithi, Generalization in feedforward neural networks, in: The Presentation at the IJCNN'91 International Joint Conference on Neural Networks, Seattle, WA, USA, 1991.
[105] P.M. Williams, Improved generalization and network pruning using adaptive Laplace regularization, in: The Presentation of the Third International Conference on Artificial Neural Networks, Brighton, UK, 1993.
[106] P. Zegers, M. K. Sundareshan, Systematic testing of generalization level during training in regression-type learning scenarios, in: The

Presentation of the IEEE IJCNN'04, 2004 International Joint Conference on Neural Networks, Budapest, Hungary, 2004.
[107] S. Zhang, H.-X. Liu, D.-T. Gao, W. Wang, Surveying the methods of improving ANN generalization capability, in: The Presentation of the 2003 International Conference on Machine Learning and Cybernetics, Sheraton Hotel, Xi'an, China, 2003.
[108] L. Zhao, J.O. Coplien, Symmetry in class and type hierarchy, in: The Presentation at the 40th International Conference on Technology of Objects, Languages and Systems (TOOLS), Sydney, Australia, 2002.
[109] L. Zhao, J.O. Coplien, Understanding symmetry in object-oriented languages, J. Object Technol. 2 (2003) 123–134.
[110] S. Zhong, V. Cherkassky, Factors controlling generalization ability of MLP networks, in: The Presentation of the IJCNN'99 International Conference on Neural Networks, Washington, DC, USA, 1999.
[111] D. Zwillinger, Inverse Circular Functions (Section 6.3), CRC Standard Mathematical Tables and Formulae, CRC Press, Boca Raton, FL, 1995.

Dr. Richard Neville has worked in the area of Artificial Intelligence for over two decades. His research interest cover: Artificial Intelligence; Computational Intelligence; Fuzzy Logic; Neural Networks; Evolutionary Computing (Genetic Algorithms); Agents (Soft, Intelligent); Parallel Processing/Distributed processing; and Data mining. He has written approximately seventy scientific papers in the area of neural networks, Genetic algorithms, Computational Intelligence and Advanced Devices.