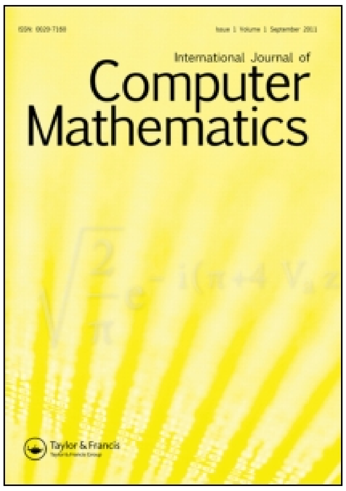

International Journal of Computer Mathematics
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713455451

Function evaluation with feedforward neural networks

Doug Logan[a]; Panos Argyrakis[b]
[a] Department 48B/428, Scientific and Engineering Computations, IBM Corporation, Kingston, New York, USA [b] Department of Physics, University of Thessaloniki, Thessaloniki, Greece

To cite this Article Logan, Doug and Argyrakis, Panos(1998) 'Function evaluation with feedforward neural networks', International Journal of Computer Mathematics, 67: 1, 201 — 222
To link to this Article: DOI: 10.1080/00207169808804660
URL: http://dx.doi.org/10.1080/00207169808804660

Intern. J. Computer Math., Vol. 67, pp. 201–222
Reprints available directly from the publisher
Photocopying permitted by license only

FUNCTION EVALUATION WITH FEEDFORWARD NEURAL NETWORKS

DOUG LOGAN[a] and PANOS ARGYRAKIS[b]

[a]*Scientific and Engineering Computations, IBM Corporation, Department 48B/428, Kingston, New York 12401, USA;* [b]*Department of Physics, University of Thessaloniki, 54006 Thessaloniki, Greece*

(Received 14 April 1997)

Much effort has previously been spent in investigating the decision making/object identification capabilities of feedforward neural networks. In the present work we examine the less frequently investigated abilities of such networks to implement computationally useful operations in arithmetic and function evaluation. The approach taken is to employ standard training methods, such as backpropagation, to teach simple three-level networks to perform selected operations ranging from one-to-one mappings to many-to-many mappings. Examples considered cover a wide range, such as performing reciprocal arithmetic on real valued inputs, implementing particle identifier functions for identification of nuclear isotopes in scattering experiments, and locating the coordinates of a charged particle moving on a surface. All mappings are required to interpolate and extrapolate from a small sample of taught exemplars to the general continuous domain of possible inputs. A unifying principle is proposed that looks upon all such function constructions as expansions in terms of basis functions, each of which is associated with a hidden node and is parameterized by such techniques as gradient descent methods.

Keywords: Function Evaluation; feedforward ANN; mappings

INTRODUCTION

Neural networks are collections of neurons that receive inputs and deliver outputs. A neuron is the smallest unit of the net; typically a net contains thousands or millions of such units. Their foremost characteristic is that all neurons are connected in a complex way to form the structure of the net. These simple ideas are some of the governing principles of the biological brain, and they are currently used extensively in such studies. In the last few years they have inspired and opened the way for the artificial nets, which are networks that are constructed and used to perform computations, to make decisions, or solve other applied problems. The common element that all nets have is that they "learn" from examples, and not via the formulation of an algorithm that addresses the particular process. A small number of examples are used to develop a mapping from inputs to outputs, that hopefully generalizes to correctly map all the other possible inputs to their correct outputs.

There have been several different mechanisms and structures of nets reported recently in the literature [1–3]. In the present work we utilize a feedforward net which is described as successive layers of neurons, where the first layer receives as many individual inputs as there are neurons at this layer, and passes these signals through successive intermediate layers, until the output is delivered from the last layer to the "outside world". Neurons at layer i are constrained to be directly connected only to neurons at layer $(i+1)$ and $(i-1)$, with either full or limited connectivity, as dictated by the goals of the model. The operation of the system is synchronous in that at each "time step i" layer i receives its inputs from layer $(i-1)$ and delivers its output to layer $(i+1)$. It is in this sense that these networks are termed "feedforward".

Fundamental to this process is the definition of the operation of a neuron. Although inspired by neurological studies, the function of a neuron is greatly oversimplified by defining its output to be either active or passive, if it receives from neurons connected to it, either excitatory or inhibiting signals, respectively, i.e. the output is bounded between some saturated maximum and minimum value. To put this operation on a mathematical basis, one can model this behavior with any number of suitably bounded functions, called activation or transfer functions. Popular models include the hyperbolic tangent and the logistic (Fermi) function. The latter takes the following form with maximum and minimum outputs of 1 and 0:

$$o = \frac{1}{1+e^{-x}} \tag{1}$$

where o is the neuron output and x represents the sum of all inputs into this neuron. Thus, x is given by:

$$x = \sum_{i=1}^{n} i_i w_i + b \tag{2}$$

where, for neurons at the second or higher layers, i_i is the output of the i^{th} neuron feeding into this neuron, and w_i is the weight attached to the "synapse" associated with this connection. The weights w_i can be any positive or negative number. B is a unique "threshold" or "bias" for each neuron, and may also be any positive or negative number. Neurons at the first layer simply act to pass the inputs, without modification, into the network. The network is defined by its number of layers, the number of neurons (nodes) within each layer, the connectivity between layers, the weights associated with each neuron to neuron connection and each neuron bias.

The usefulness of such networks lies in the fact that it is often possible to "teach" them to make arbitrarily complex mappings between a set of inputs and desired outputs. Examples include the recognition of phonomes [4, 5] (for speech recognition), and pattern recognition [6, 7] (such as decoding noisy inputs of alphanumeric data). It has also been demonstrated that such disparate mappings as chaotic time series predictors [8] to developing expertise at playing backgammon [9] can be mastered with some degree of success.

Teaching a netwrok involves determining optimal (not necessarily unique) numeric values for the weights of each synapse and bias. The most popular training algorithm to achieve this goal is an iterative gradient descent method called backpropagation [10]. Here an error measure or objective function can be defined as:

$$E = \sum_{p=1}^{P} e_p \tag{3}$$

where **P** is the total number of distinct mappings to be taught and e_p is an error measure for each such pattern, defined usually as the 2 norm of the vector difference between the correct output and the measured output from the output layer of neurons:

$$e_p = \sum_{i=1}^{n} (t_i - o_i)^2 \tag{4}$$

where n is the number of output neurons, t_i is the desired (target) value of the output from neuron i, and o_i is the measured output. Each iteration of this method concludes with an update of each weight and bias value, and continues until the error measure is sufficiently small. Treating the bias in an equivalent manner, this involves updating each weight w_{ij} at iteration $(t+1)$

in terms of w at iteration t:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial E}{\partial w_{ij}} \tag{5}$$

The partial derivative for the dependence of the total error E with respect to any weight w_{ij} connecting neuron i to neuron j, can be calculated by a straightforward application of the chain rule. Parameter η is a positive number that indicates the extent of projecting the correction in the gradient's direction.

Nevertheless, assuming that a specific neural net may be optimally parameterized, it is not often clear whether such techniques are robust enough to discover them. Thus, as with any gradient descent method, there arise questions about whether the solution can become trapped in local minima (the gradient is zero at some point in weight space for which our objective function E is not zero), or, indeed whether local minima exist for the particular mapping. Use of simulated annealing [11] is often cited as a method for escaping such minima. A related issue, that is often crucial in determining how rapidly convergence is approached, involves how far to proceed along the gradient path (are exact or approximate line searches adequate to ensure a reasonable convergence rate?). An exact line search along the gradient may determine an "optimal" value of η (see equation 5) to be used to adjust the weights for a particular iteration. Often [10] however, because line searches are time consuming, the value of η is set to some arbitrarily small value hoping that, while not necessarily optimal, at least the error is decreased. Another fear is that a "pure" gradient descent method may be subject to "cross stitching", [12] i.e. the phenomena where successive iterations over weight space simply cross back and forth across a steep ravine without making much progress down the ravine towards to global minimum. To avoid this problem, it has be suggested [13] that conjugate gradient methods be employed. Often [10] this approach is implicitly invoked, in somewhat heuristic fashion, by adding a term to the gradient increment called the "momentum", which is calculated as some (usually fixed) fraction of the gradient from the previous iteration. Thus, teaching neural networks may be viewed [13] as simply an unconstrained optimization problem and may borrow all the (less than ideal) techniques that have been developed over many years to deal with such problems.

In the following we examine feedforward neural networks mainly from a computational aspect, i.e. we examine whether we can teach neural networks to implement a number of operations in arithmetic and function evaluation. It has been recently shown [14] that for any square integrable function there always exists a 3-layer backpropagation neural network that can approximate this function to within a specified mean square error accuracy. The types of functions we consider can be categorized in terms of the number of separate input and output variables, i.e. one-to-one mappings, two-to-one, etc. Examples considered range from performing reciprocal arithmetic on real valued inputs, to implementing particle identifier functions for identification of nuclear isotopes in medium energy scattering experiments. All mappings are required to interpolate (or extrapolate) from a small sample of taught exemplars to the general continuous domain of possible inputs.

Our goal is to demonstrate that neural networks can be constructed to implement useful computationally complex function evaluations. Our conclusion is that they are able to do so because of their equivalence to expansions of functions in terms of elementary basis functions (e.g., weighted logistic functions). These types of basis functions prove to be quite flexible for achieving satisfactory mappings whether such mappings are functional in form, or of the image identification form. Needless to say, there are several other techniques for function approximation, such as, for example, sequential learning schemes using minimal radial basis function neural nets [15], and also analogous techniques for the approximation of functionals [16]. In the present study we limit ourselves to feedforward nets.

COMPUTATIONS

We restrict ourselves for simplicity to examining only three layer networks, where the first layer is simply the presentation of the input and the next two layers include neurons as defined using the logistic function. We employ full connectivity between adjacent layers only, with no direct connections between input and output layers. Each neuron in the hidden layer and output layer has a unique bias unit.

We investigate whether such networks can be taught a few exemplars from a number of smooth functions, and from these, accurately interpolate over appropriate continuous domains of all inputs. The algorithm used to teach examples is the standard backpropagation method [10] without conjugate gradients or "momentum". The projection parameter, η (see equation 5), is calculated on each iteration using Newton's method for "root finding". This is based on expanding the error or objective function to only first order and correcting the weights using the one norm of the gradient vector.

Thus, η is calculated as:

$$\eta = \frac{E}{\sum_{i=1}^{N} \left| \frac{\partial E}{\partial w_{ij}} \right|} \tag{6}$$

Here N is the total number of weights within the network (including bias units). Because the convergence rate of any root finding algorithm is sensitive to initial conditions [17] it is important that the initial weights be selected appropriately. This can be done by assigning small random weights such that the inner product of weights and inputs to any neuron is initially close to zero. In this case, the partial derivative of the output of a neuron with respect to a given weight, will tend to be large (it is maximized identically for zero input) and the iterative procedure starts with a direction along which significant progress can be made. If Newton's method is initialized with larger weights, it is often found that the projection along weight space is too haphazard and convergence can be poor.

This method is used, rather than a second order method involving computation of the Hessian error weight matrix and solving large systems of linear equations [17], because it is significantly less computationally complex, is more parallel and thus closer to a "real" neural network, and makes direct use of the error magnitude. The error can be assumed to be exact if we a priori assume that our network is capable of finding the global minimum.

The types of functions which are to be taught are constrained given that the output of any neuron (and specifically the output neurons) is bounded between 0 and 1. This consideration affects the type of functions we now consider and/or the required scaling implicit in order to attempt such mappings. The functions are categorized in terms of the number of neurons required to describe the input and the output. Thus, we consider a number of functions of the following from: one-to-one, two-to-one, and many-to-many.

ONE-TO-ONE FUNCTIONS

Whether it is useful to build neural networks to implement such simple functions is debatable. However, by examining such functions as $f(x) = 1/x$

and others, it is possible to look in more detail at exactly what is being accomplished, which in the case of more complex mappings is often obscured. Here both the input and output layers contain one neuron only. The number of hidden neurons is taken to be variable and full connectivity between adjacent layers is assumed. It is also assumed that each neuron, with the exception of the input neuron, has connected to it a bias unit that is adjustable, like any other weight. Some insight into how a given mapping is represented is provided by the following considerations. In general we want to be able to achieve some mapping:

$$y = f(x) \tag{7}$$

where x is the input variable and $f(x)$ the output. If we take the output as given by values generated by the logistic function (subsuming all signs into parameters) we have:

$$y = \frac{1}{1+e^{x}} \tag{8}$$

and then we can represent the output equivalently as

$$y' = \text{In}\left(\frac{1}{y} - 1\right) \tag{9}$$

in which case we can isolate the contribution of each hidden neuron as

$$y' = \sum_{i=1}^{n} f_i + b_y \tag{10}$$

where each f_i, in a sense, is a basis function described uniquely by the i^{th} neuron in the hidden layer:

$$f_i = \frac{w_i}{1 + e^{-(\Sigma_i z_i i_i - b_i)}} \tag{11}$$

In the one-to-one mappings w_i is the weight connecting the i^{th} hidden neuron to the output neuron and z_i is the weight connecting the input i to this particular neuron. The biases are represented by b_y for the output neuron and b_i for each hidden neuron. The process of "teaching" the network to map x to $f(x)$ is then equivalent to developing optimal basis functions, each parameterized by weights and biases.

Viewed in this way, function approximation with neural networks is entirely analogous to such techniques as Fourier expansions. With the latter we know that including successive terms will reduce the residual error of the approximation. To first order, this is also true for this case, i.e. increasing the number of hidden neurons (basis functions) also will provide the capability of refining the approximation. However, this would greatly increase the cpu time required, so this limits the number of hidden neurons that can be employed, which in turn limits the precision of the mapping.

These basis functions are not orthogonal but have a number of desirable properties. They are continuously differentiable, can be either positive or negative with variable magnitude (depending on w_i), and they are either monotonically increasing or decreasing over the bounded region between 0 and w_i The dependence on the input i can be rapidly or slowly changing depending on the magnitude and sign of z_i, and shifting capability is provided by the sign and magnitude of the bias b_i. In short, they provide flexible and economic basis in which general smoothly varying functions may be represented. Unlike other expansion techniques that use orthogonal basis functions and Gram-Schmidt like procedures, here we attempt to concurrently parameterize a fixed number of non-orthogonal functions (using algorithms such as backpropagation). This may not be the best manner to get the optimal expansion (as opposed to incrementally adding on new basis functions and tuning them to eliminate residual erros from the current set of optimized functions); however, at least it is well defined.

Our goal in attempting simple one-to-one mappings has not been to do the best possible job, but rather to see if reasonably accurate results (1 to 2% error) can be obtained at the expense of minimal computer time. A number of mappings that have been found to be easily taught to this level of accuracy include:

$$f(x) = \frac{1}{x} \qquad O(100) > x > 1 \tag{12}$$

$$f(x) = \log_{10} x \qquad 10 > x > 1 \tag{13}$$

$$f(x) = e^{-x} \qquad O(10) > x > 1 \tag{14}$$

$$f(x) = \sin x \qquad \pi/2 > x > 0 \tag{15}$$

$$f(x) = x(1-x) \qquad 1 > x > 0 \tag{16}$$

where the ranges of the mapping taught are indicated. These constraints are dictated by the fact that the output of the nodes at the output level is bounded between 0 and 1.

As a representative case we consider the mapping $f(x) = 1/x$ for $100 > x > 1$, where a network with 7 hidden neurons was trained to calculate the reciprocals of 20 values of x. The ability of the network to interpolate for all values of x over this range is indicated in Figure 1, where we plot the absolute % error for values of x not included in the training set. The training was achieved in 280000 iterations with minimal computer time. Although better accuracy may be obtained by performing more back-propagation steps, significiantly diminishing returns on accuracy improvement was observed at this level of training. Increasing the number of hidden neurons (basis functions) uniformly improves accuracy to about 7 neurons

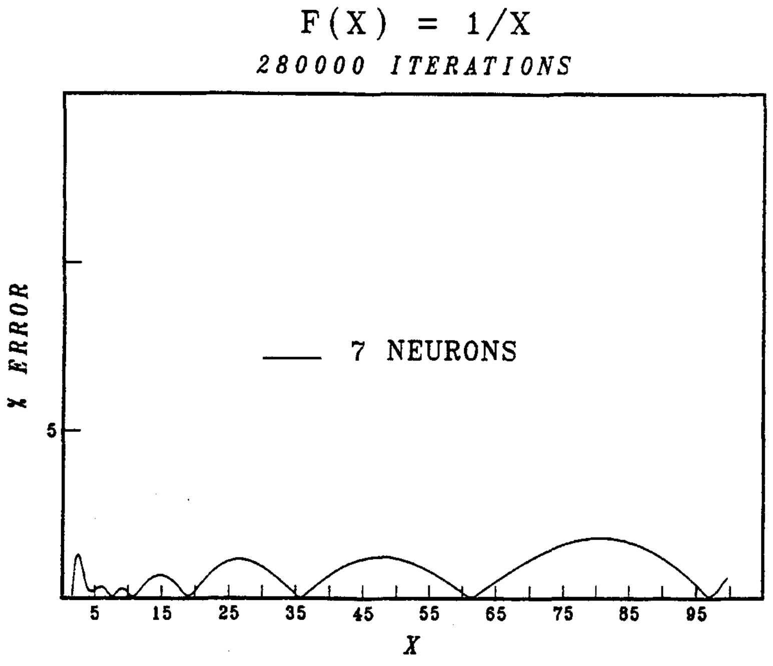

FIGURE 1 Absolute percent error in computing $f(x) = 1/x$.

for the same number of iterations. Thereafter, performance degrades somewhat (although accuracy continues to improve) as more neurons are added. The asymptotically best performance achievable is difficult to predict given the non-orthogonal basis used and the manner in which the basis functions are adjusted.

The greater relative accuracies for small values of x is expected in that we have defined our error function in terms of the square of the absolute difference betwen correct and measured values of the reciprocal. This means that the gradient corrections tend to favor faster convergence for small values of x, where for the same relative percent error, small values of x will have much larger absolute errors compared to larger values of x. Because our partial derivatives are proportional to the absolute differences, the gradients are more aggressive in correcting errors for small values of x. It is possible to use alternative objective functions, such as entropy expressions [18] or of non 2 norm form [19] to promote convergence in different regions at different rates.

The basis functions and the desired mapping are indicated in Figure 2 for the 7 hidden neuron network. Here, the exact reciprocal function, as defined

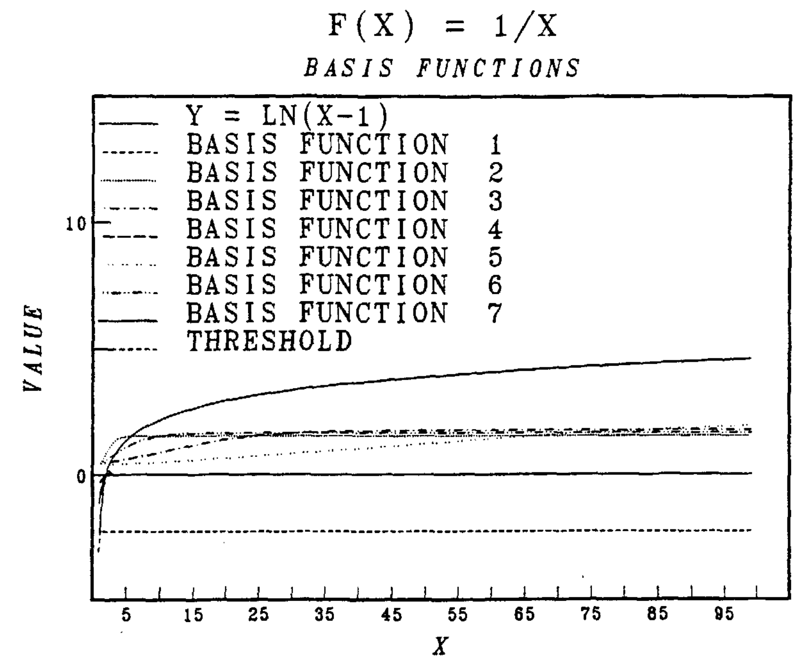

FIGURE 2 Basis function representation for $f(x) = 1/x$.

equivalently in equation 9, is plotted along with the basis functions (equations 10, 11) whose parameters were optimized using the back-propagation method. The seven basis functions, as noted, add up to within 2% or so of the exact function over the range from $x = 1$ to $x = 100$. Three of the basis functions are negative, four are positive, and the majority have developed strong nonlinearities to represent the strong curvature over the range of $x = 1$ to $x = 10$. One basis function (number 5) has developed a very gradual nonlinearity to represent the more gradual curvature for larger values of x.

It is difficult to make quantitative statements relating the optimal number of layers, number of neurons within each, the asymptotic residual error and the rate of error minimization to the number of training iterations and cumulative cpu time. Regarding these issues the following comments may be made. It has been shown [14] that any square integrable function can be represented by a three layer network with asymptotically large number of hidden neurons. However, the optimal number of neurons to achieve the smallest residual error must be defined in terms of how much computer time we allotted. Even if this number were known, the success of the training procedure is quite sensitive to a number of additional variables. First among these is the number and choice of the examples to be taught. While it is clear that they should be chosen to represent the aggregate behavior of the function as well as possible, it is difficult to refine this further. Second, like any gradient descent technique, the rate of progress is very sensitive to initial conditions i.e. the starting guesses for all weights and biases. Third, while we would like to make as much progress towards the global minimum as possible per iteration, there accrues a large expense in computer time if we use techniques more complex than simple Newton methods. For these reasons, we do not include measurements of error reductions as a function of number of iterations and cpu cycles, number of hidden neurons, etc. Our purpose in this work is simply to point out the underlying basis function expansions that are being carried out, and to illustrate this with a number of functions. With this perspective, the above issues can perhaps be more fruitfully pursued on an analytic basis rather than experimentally.

TWO-TO-ONE FUNCTIONS

Single valued functions of two variables may also be implemented on these networks. For example, mapping the exclusive Boolean OR is one of the prototype examples cited [10] in support of the abilities of multilayer perceptrons to successfully categorize decision regions that are linearly nonseparable. However, the principal types of functions we consider here are those with real continuous valued (not binary) inputs from which we wish to compute a real valued numeric output. In this, like the one-to-one mappings discussed, the same considerations apply in determining which types of functions can be treated, and over what range, given that the output is constrained in the region (0,1). Similarly, we require that the network accurately generalize, when trained only with a small set of exemplars. The type of network we require now has two nodes in the input level and one node in the output level. One additional consideration is whether we can implement two variable functions that are invariant to permutation of the inputs on the respective input neurons, i.e. can we train our neural networks to implement functions that satisfy:

$$f(x_1, x_2) = f(x_2, x_1) \tag{17}$$

where the ordering refers to the respective input neuron on which the value of x_1 and x_2 are applied. In general, such invariance implies that all weights connecting the inputs to the first hidden layer must be the same, as each input must be treated equally. For the simple 3-layer networks we consider here, this leads to a diffiicult constraint in being able to train a variety of functions for which there is a relational constraint between the inputs. Thus, we consider only two varible functions in which the order of inputs to input neurons is significiant. In this case a number of functions have been constructed with the same degree of success as in the one variable functions. These include such functions as:

$$f(x) = e^{-(x_1 - x_2)} \qquad x_1 > x_2 \tag{18}$$

$$f(x) = e^{-(x_1 - x_2)^2} \qquad x_1 > x_2 \tag{19}$$

The first function may be recognized as the one used in the Metropolis criterion [20] for accepting or rejecting a state transition for a state at energy x_2 going to a higher energy x_1, for a system obeying Boltzmann statistics. The second function is the probability density at point x_1 for a normal distribution with mean x_2 and unit variance. It is possible to teach many variations of such functions with equal success.

More complex functions may also be implemented. For example, consider the area of medium energy heavy ion nuclear physics. Experiments in this field typically employ a heavy ion accelerator to bombard a given target with a heavy ion beam. The nuclear reaction products are then detected with arrays of Silicon semiconductor detectors. Such products usually constitute a full spectrum of isotopes at various kinetic energies. The first order of business is to separate and identify the isotopes. This requires a minimum detector telescope defined by two silicon detectors; the second detector immediately following the first and the telescope oriented in the direction in which the products are emitted. The first detector that a particle passes through is very thin (relative to the total range of the energetic particle) and is called a delta E detector. The second detector is of sufficient thickness to fully stop the particle and is called an E (or stopping) detector. The rate of energy loss measured by the thin detector can be approximated as a function of the total energy of the particle (measured by both the thin and stopping detector) and its mass and nuclear charge [21]:

$$\frac{dE}{dx} = C_1 \frac{mz^2}{E} \ln\left(C_2 \frac{E}{m} \right) \tag{20}$$

where dE/dx is the rate of energy loss in the thin detector, E is the total energy, m is the mass of the particle, z is its nuclear charge, and C_1 and C_2 are constants. For nonrelativistic energies, the product of $E(dE/dx)$ is a fairly sensitive indicator of the isotope identified by mz^2. A plot of dE/dx versus E for a large number of isotopes indicates that their loci are approximately convex. For isotopes of similar values of mz^2 identification is often difficult because their loci are only slightly separated. However, we can train a feedforward network to implement a particle identifier function that computes a suitably scaled value of the mz^2 product. Thus, rather than have as many output neurons as there are isotopes, and use "image" identification training to uniquely set a given neuron "on" in response to a given $(dE/dx, E)$ pair applied to input neurons, we ask instead that only one output neuron output a value that can be unambiguously associated with each isotope. It is evident that most image identification problems can be cast in this function evaluation form.

In the present case we teach the particle identifier function for only a few isotopes at a few selected energies, but require it to identify correctly a much larger set of isotopes that can have energies over the continuous domain. The particular examples were He^4, Li^6, and C^{12} each at energies of 2.5,5.0 and 10.0 MeV/nucleon. The dE/dx values for these examples were calculated using stopping powers extracted from the data of Reference 22. The capability of the network to generalize to all other energies of these isotopes as well as all other feasible isotopes over this mass and charge region was then examined. This is indicated in Figure 3, where the value of the single output neuron is plotted. It may be seen that taught isotopes are clearly separated and identified as are a large number of untaught isotopes including, **Li**^{7}, Li^{8}, Be^{7}, Be^{9} (Be^{8} is too unstable to exist for the length of time to traverse any conventional detector telescope), **Be**^{10}, B^{10}, B^{11} and C^{11}. Thus, we can construct a particle identifier function to uniquely identify a large spectrum of isotopes. In principle this technique could be implemented on-line with an experimental set-up, and the identification could be performed instantaneously.

MANY-TO-MANY FUNCTIONS

We next examine whether a neural net can implement functions or transformations of many variables of the form:

$$y_1, y_2, \ldots y_{n3} = f(x_1, x_2, \ldots x_{n1}), \tag{21}$$

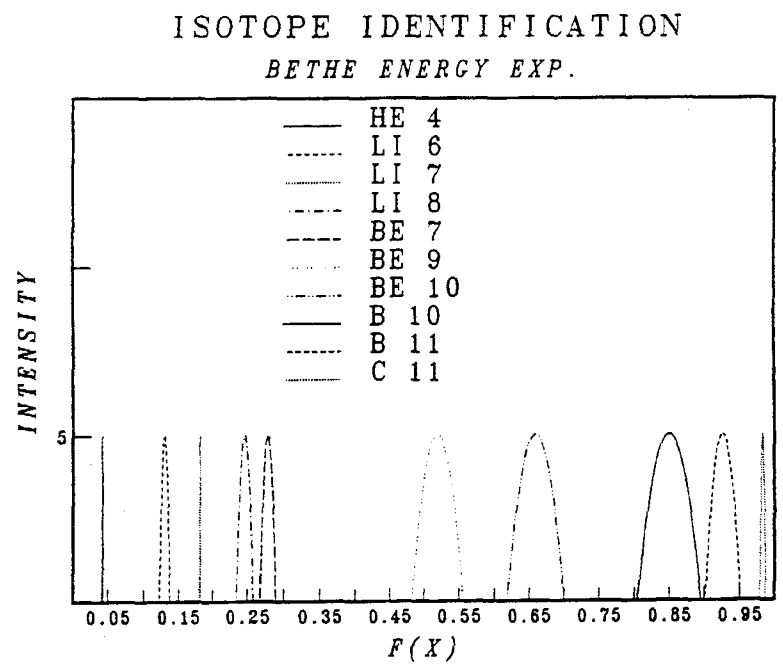

FIGURE 3 Particle identifier output for various isotopes sampled over energies from 1 Mev/nucleon to 10 Mev/nucleon.

where $y_k(k = 1, \ldots, n3)$ is the k^{th} ordered output and x_i $(i = 1, \ldots, n1)$ is the i^{th} ordered input. Thus the type of network to be constructed would have a relatively large number of units in all three layers, with connectivity, weights, and biases as defined above. The input and output layers have the requisite number of neurons to represent the mapping of all inputs to the necessary number of outputs, while the hidden layer may have a variable number of neurons as is appropriate for the particular mapping sought. As usual, scaling the function to the domain (0, 1) is required.

We begin an examination of such mappings with the following problem. Consider that we have a bounded two dimensional domain over which some particle or source can move. The particle may generate some field, or, perhaps emit some additional particles. It could be a charged particle that creates a Coulomb field or an isotropically emitting light source. The problem that we wish to solve is the following: given a set of detectors that measure the strength of the field (or light intensity) at defined locations, calculate the coordinates of the particle. The bounded region is rectangular with one detector at each of the four corners. If we consider a charged particle then the strength of the field measured at each detector is proportional to $1/r$, where r is the distance from the particle to detector. If we consider a light emitting source, then the strength of the measured intensity at each detector is proportional to the solid angle subtended by the detector with respect to the emitter, or equivalently, to $1/r^2$. We would like to construct a neural network that would take as input the four detector measurements, and output the coordinates of the particle. To scale this problem, we define our grid over the domain (0, 1) in both x and y directions.

For both types of computations, a limited amount of computer time (several 100,000 iterations) is required to train a minimal 4:4:2 network to compute reasonably accurate coordinates over the full domain when taught only a small number of test positions. Of the two output neurons, one served to output the x coordinate value, the other the y value. The field problem was trained with 50 positions uniformly placed over the region $x = 0.05$ to 0.95 and $y = 0.05$ to 0.95, with all detector measurements normalized such that the largest measurement corresponded to unity. The relatively accurate ability of the network to calculate coordinates may be appreciated from Figures 4 and 5 which show, respectively, the true and calculated positions of 81 untaught positions.

It is instructive to examine the outputs of each of the four hidden neurons in the equivalent basis set representation as expressed in Equations 9–11. Holding y constant at 0.5, and varying x over the range of (0.05, 0.95), we expect the four basis functions that contribute to the output of the x output

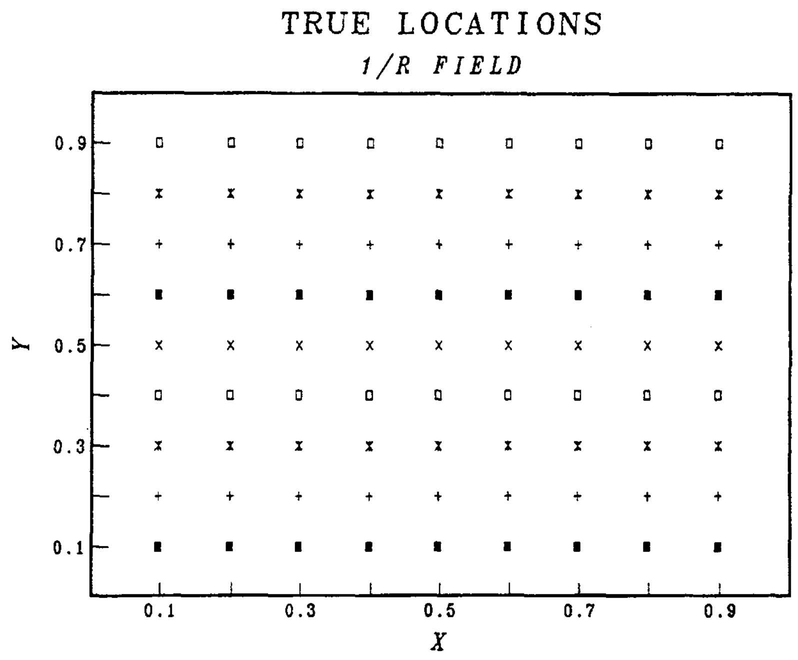

FIGURE 4 A set of untaught particle positions.

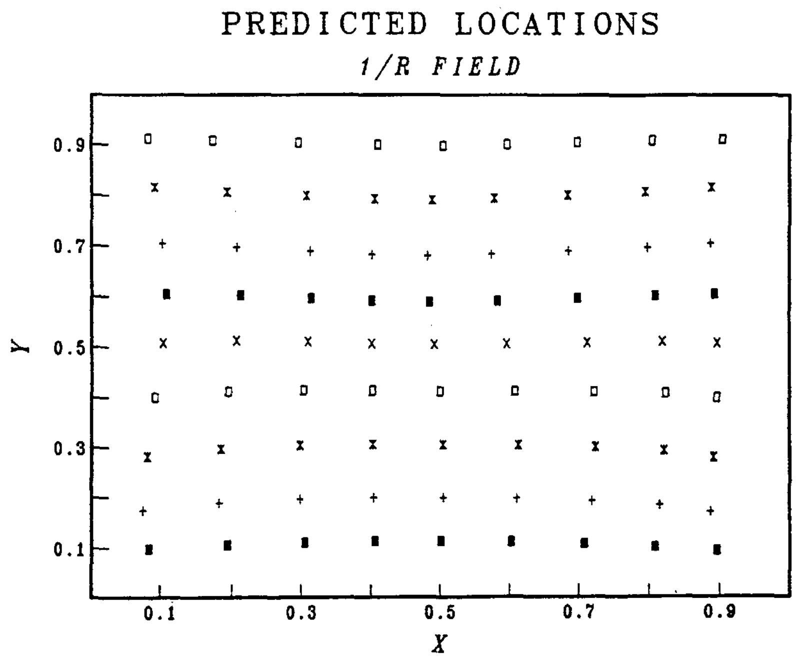

FIGURE 5 Computed positions for the set in Figure 4.

neuron, when added to the bias of that neuron, to sum to the decreasing function illustrated in Figure 6. For y constant at 0.5, the biased sum of the four basis functions that contribute to the y output neuron, should equal zero (see Fig. 6). Given the symmetry of the problem, we expect the same corresponding behavior when x is held constant, and y varied. Note that the basis function from a given hidden neuron that contributes to either the x or y output neuron, differs parametrically only in the weight attaching that neuron to the appropriate output neuron.

Taking the case of holding $y = 0.5$ constant, and plotting the outputs of our 4 basis functions feeding the x output neuron, we observe in Figure 7 that 2 hidden neurons (the second and fourth) have developed the ability to calculate the x^{th} position, while the other two basis functions effectively do not contribute (the constant bias of the x output neuron has been added in to these 4 basis functions such that their sum is correct, and to illustrate which hidden neurons are actually measuring change in this coordinate). If we look at the basis functions for the same case, but instead examine their contributions to the y output neuron, we observe the behavior shown in Figure 8, where it is seen that the sum of all basis functions add to

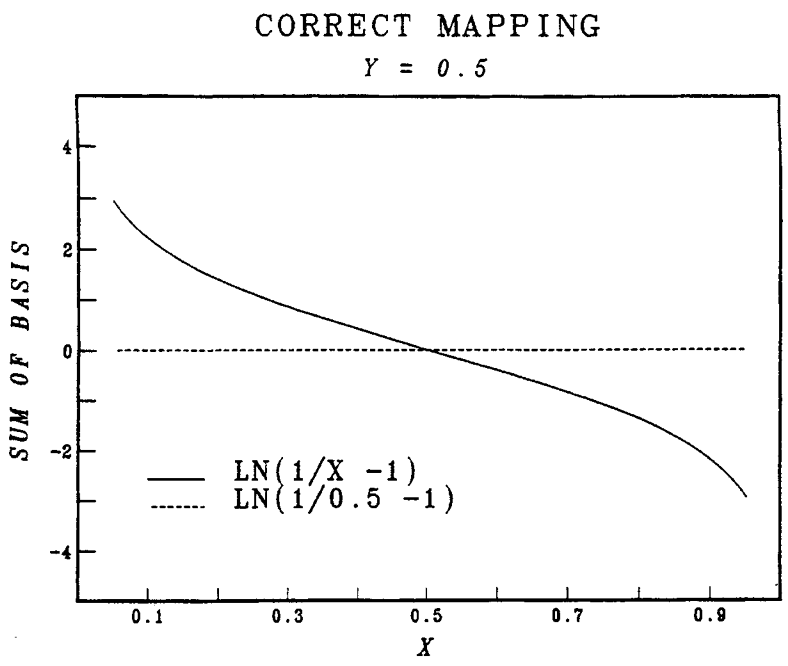

FIGURE 6 Correct representation of coordinate x and y ($y = 0.5$ constant).

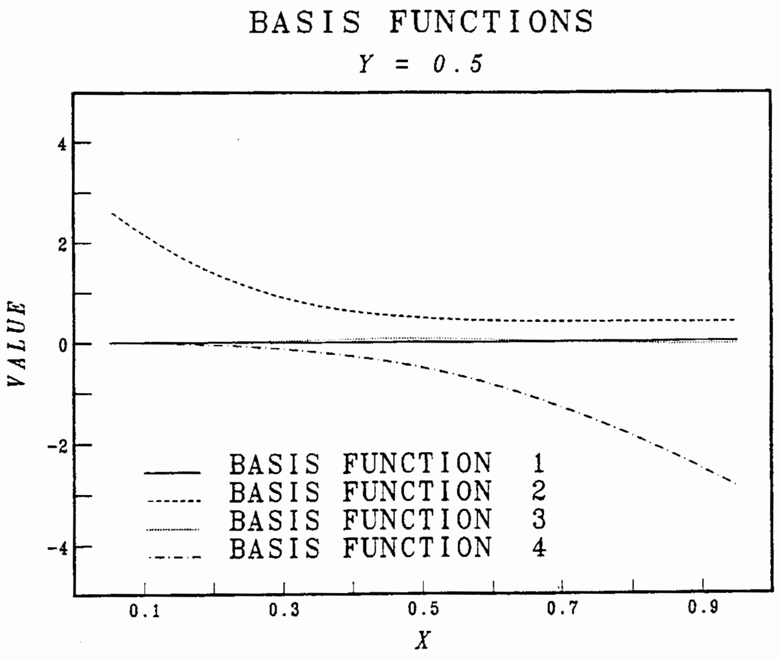

FIGURE 7 Basis function representation for coordinate $x(y=0.5)$.

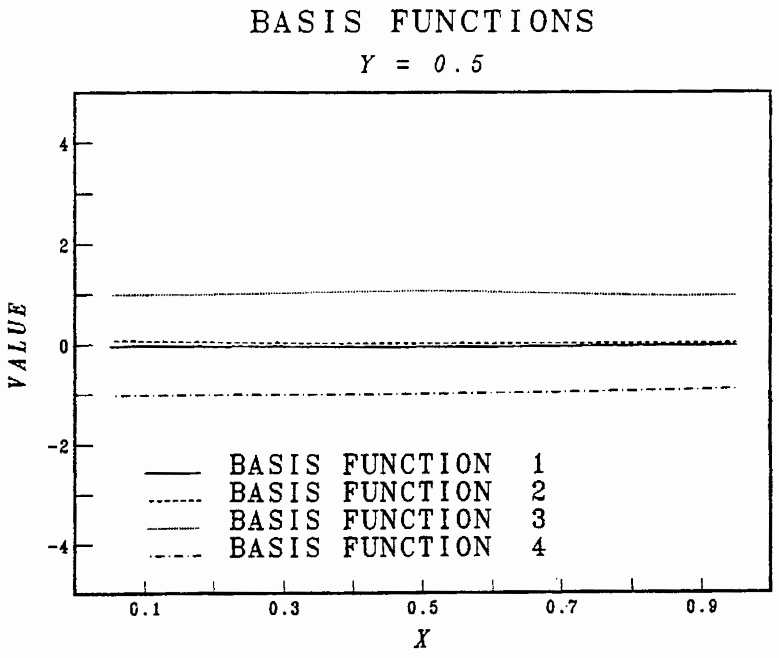

FIGURE 8 Basis function representation for coordinate $y(y=0.5)$.

approximately zero (which is the value required for ln (1/.5 − 1)). If we look at the basis functions for other values of y = constant we observe the same behavior.

Alternatively, if we examine the basis set representations holding $x = 0.5$ constant, we observe that the other two hidden neurons supply basis functions that are sensitive to variations of the y^{th} coordinate. This is shown in Figure 9. The plot corresponding to the basis functions for the x output neuron is shown in Figure 10, where the sum of all functions effectively equals zero (as it should) for this value of x.

In short, the backpropagation method has found an economic and accurate parameterization of a symmetric, but complicated function, using only 4 basis functions. The parameters are close to their optimal values, i.e. increased training iterations lead to very marginal improvement. However, it is expected that increasing the number of basis functions (number of hidden neurons) would afford increased accuracy.

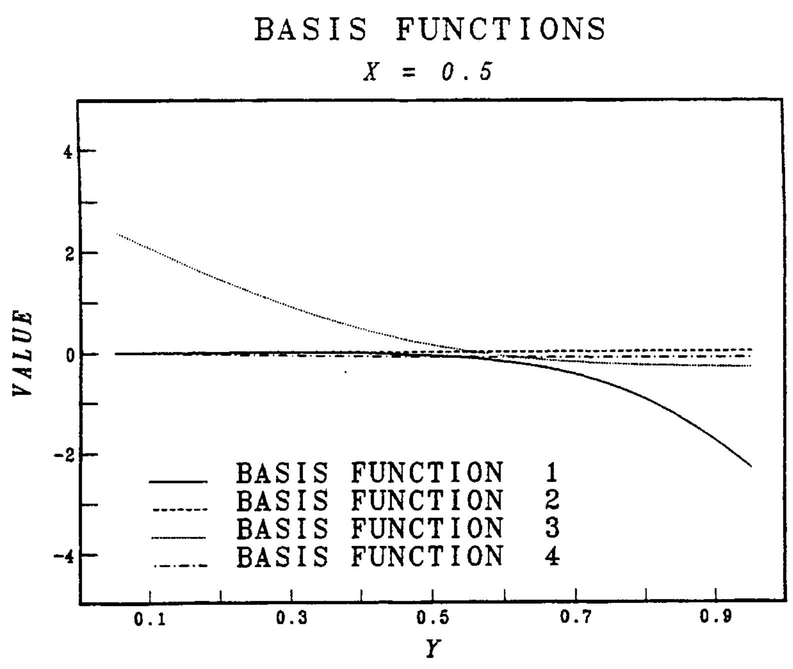

FIGURE 9 . Basis function representation for coordinate $y(x = 0.5)$.

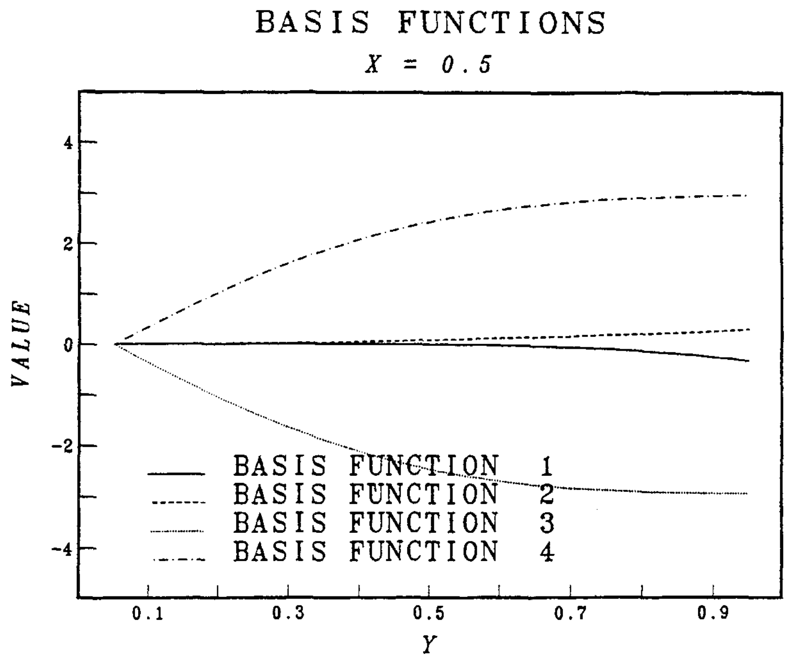

FIGURE10 Basis function representation of x coordinate ($x = 0.5$).

CONCLUSIONS

Our intention in this work has been to examine the computational abilities of neural networks rather than their more often cited abilities applied to image identification. We required that the neural networks implement a number of representative functions of one or many variables and interpolate correctly over the continuous domain, when taught only a few correct correspondences of input to output. In this regard, the neural networks may be considered successful or not depending upon the degree of accuracy required. We have considered the functions, implemented with several percent accuracy, to be successful. Developing networks with greater accuracy corresponds to simply carrying out the training over longer periods (more iterations) and/or increasing the number of hidden nodes. However, the asymptotic best results achievable are difficult to predict and the expense in computer time may become prohibitive. Additionally, there also exist some inherent problems which may prohibit the feedforward network from learning some polynomial functions. This is not because there are problems in the training algorithm, but it may be due to inherent consequences of the role of the connection weights [23].

As might be expected, these techniques work best for approximation of smooth functions provided the examples taught adequately represent the overall behavior of the function. This observation was reinforced by examining the ability of a network to perform binary bit addition, multiplication, etc. For example, addition of two 3-bit numbers involves 64 possible pattern mappings. Each output is distinctly discontinuous for a different sum. Thus, teaching the net all permutations for every other sum but one, in general, left the net with poor behavior at generalizing correctly for the latter.

Our main conclusion is that neural network training is equivalent to expanding a function in terms of basis functions, where each hidden node is a basis function described, for example, by an optimally parameterized logistic function (or other such lower and upper bounded function). The backpropagation algorithm is an effective means of achieving the parameterization. However, it is not clear if it is the best algorithm to achieve this end.

Acknowledgment

We thank Dr. E. Clementi for his suggestion to investigate the feasibility to implement arithmetic operations and function evaluation.

References

[1] Anderson, J. A. (1995). *An Introduction to Neural Networks*, MIT Press (Cambridge, Mass. USA).
[2] Haykin, S. (1994). *Neural Networks and Comprehensive Foundations*, Prentice-Hall (Upper Saddle Point, NJ USA).
[3] Bose, N. K. and Liang, P. (1996). *Neural Network Fundamentals with Graphs, Algorithms, and Applications*, McGraw Hill (New York).
[4] Leung, H. C. and Zue, V. W. (1989). Applications of Error Back Propagation to Phonetic Classification, in: *Advances in Neural Information Processing Systems*, ed. D. Touretzky, Morgan Kaufmann Publ., San Mateo, Ca. 206.
[5] Bengio, Y., Cardin, R., De Mori, R. and Cosi, P. (1989). Use of Multilayered Networks for Coding Speech with Phonetic Features, in: *Advances in Neural Information Processing Systems*, ed. D. Touretzky, Morgan Kaufmann Publ., San Mateo, Ca.224.
[6] Mori, Y. and Yokosawa, K. (1989). Neural Networks that Learn to Discriminate Similar Kanji Characters, in: *Advances in Neural Information Processing Systems*, ed. D. Touretzky, Morgan Kaufmann Publ., San Mateo, Ca.332.
[7] Wilkinson, T. S., Mighell, D. A. and Goodman, J. W. (1989). Back Propagation and its Applications to Handwritten Signature Verification, in: *Advances in Neural Information Processing Systems*, ed. D. Touretzky, Morgan Kaufmann Publ., San Mateo, Ca.340.
[8] Lapedes, A. and Farber, R. (1987). How Neural Networks Work, in: *Neural Information Processing Systems*, ed. D. Z. Anderson, AIP, New York, 442.

[9] Tesauro, G. and Sejnowski, T. J. (1987). in: *Neural Information Processing Systems*, ed. D. Z. Anderson, AIP, New York, 794.

[10] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). Learning Internal Representations by Error Back Propagation, in: *Parallel Distributed Processing*, The MIT Press, Cambridge, Ma.318.

[11] Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing, *Science*, **220**, 671.

[12] Luenberger, D. G. (1986). *Linear and nonlinear programming*, Addison-Wesley, Reading, Mass.

[13] Kramer, A. H. and Sangiovanni-Vincentelli, A. (1989). Efficient Parallel Learning Algorithms for Neural Networks, in: *Advances in Neural Information Processing Systems*, ed. D. Touretzky, Morgan Kaufmann Publ., San Mateo, Ca.40.

[14] Hecht-Nielsen, R. (1989). in Proceedings of the International Joint Conference on Neural Networks, **1**, 593.

[15] Yingwei, L., Sundararajan, N. and Saratchandran, P. (1997). A sequential learning scheme for function approximation using minimal radial basis function neural nets, *Neural Computation*, **9**, 461.

[16] Mhaskar, H. N. and Hahm, N. (1997). Neural nets for functional approximation and system identification, *Neural Computation*, **9**, 143.

[17] Luenberger, D. G. (1969). *Optimization by vector space methods*, Wiley, N.Y.

[18] Baum, E. B. and Wilczek, F. (1987). in: Neural Information Processing Systems, ed. D. Z. Anderson, AIP, New York, 52.

[19] Hanson, S. J. and Burr, D. J. (1987). in: Neural Information Processing Systems. ed. D.Z. Anderson, AIP, Newyork, 348.

[20] Metropolis, N., Rosenbluth, A. W., Rosenbluth, N. M., Teller, A. H. and Teller, E. (1953). Equation of state calculations for fast computing machines, *J. Chem. Phys.*, **6**, 1087.

[21] Knoll, G. F. (1979). *Radiation Detection and Measurement*, Wiley, New York.

[22] Hubert, F., Fleury, A., Bimbot, R. and Gardes, D. (1978). Range and Stopping Power Tables for 2.5-12 MeV/Nucleon Heavy Ions in Solids, Technical Report of Centre d'Etudes Nucleaires de Bordeaux and Institute de Physique Nucleaire Paris, France.

[23] Cardell, N. S., Joerding, W. and Li, Y. (1994). Why some feed forward networks cannot learn some polynomials, *Neural Computation*, **6**, 761.