TITLE: Interval Neural Networks

AUTHOR(S): Rajendra B. Patil
Computer Research and Applications
CIC-3, MS B256
Los Alamos National Laboratory, Los Alamos, NM 87545

SUBMITTED TO: APIC'95 Extended Abstracts, A Supplement to the International Journal of Reliable Computing, El Paso

By acceptance of this article, the publisher recognizes that the U. S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U. S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U. S. Department of Energy.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
Interval Neural Networks

Rajendra B. Patil

Abstract— Traditional neural networks like multi-layered perceptrons (MLP) use example patterns, i.e., pairs of real-valued observation vectors, \((\vec{x}, \hat{y})\), to approximate function \(\hat{f}(\vec{x}) = \hat{y}\). To determine the parameters of the approximation, a special version of the gradient descent method called back-propagation [2] is widely used.

In many situations, observations of the input and output variables are not precise; instead, we usually have intervals of possible values. The imprecision could be due to the limited accuracy of the measuring instrument or could reflect genuine uncertainty in the observed variables. In such situations, input and output data consist of mixed data types; intervals and precise numbers.

Function approximation in interval domains is considered in this paper. We discuss a modification of the classical back-propagation learning algorithm to interval domains. Results are presented with simple examples demonstrating few properties of nonlinear interval mapping as noise resistance and finding set of solutions to the function approximation problem.

I. INTRODUCTION

A. Problem Definition: Function Approximation

In modeling a complex process, in rare cases, one knows about the underlying process to be able to set up a rather exact and complete mathematical description of the process. But in most cases, our knowledge about the process is incomplete and one has to select the form of the mathematical model, collect input-output observations of the process (training data) and estimate the model parameters empirically in order to obtain an acceptable predictor \(\hat{y} = f(\vec{x})\) of the observed process. It is implicitly assumed that the prediction objects (future data or test data) obey the same model structure as those used (training data) in the modeling process. There exist many methods (e.g., least squares) which calculate estimates of the model parameters. Such estimates are called point estimates.

Random errors are present in all measurements, and no mathematical model accounts for all facets of a physical process. Therefore we cannot hope to obtain point estimates exactly to the true values of parameters (if such exist). Nor can we expect point estimates calculated from different data samples over different experiments of the same process to be equal, even if the samples were obtained under similar situations. The point estimates differ because the input-output observations collected from the same process under similar situations differ due to measurement errors, randomness and uncertainty. In such problems one can consider the set of observations to be interval valued input-output observations. In some situations, variables are uncertain variables, e.g., consumption, expenses, etc., and are naturally represented as intervals. Many such modeling problems exist in operations research, and economics [7,10]. In such cases, where input-output observations are interval valued, interval estimates of the model parameters are required. In order to develop the model relating interval input variables to interval output variables interval computations [1] can be incorporated into any traditional modeling approach to handle interval valued data.

The objective of this paper is to extend one well know class of data modeling method in Neural Networks (NN) called Multilayered Perceptron (MLP) [2,3] to process interval valued data. The problem of modeling is common to applied statistics and neural networks. Neural networks has provided statistic with flexible multivariate nonlinear models. It also inspires statistical science with the notions of learning, self-organization, dynamics, etc. which statistics has so far paid little attention [15]. Though considerable amount of controversy exist between the areas of applied statistics and neural networks [15,16,5] in solving the problems of data modeling, neural networks have certainly proved their application potential in the areas of pattern
classification and pattern recognition [9] like speech recognition, prediction in finance, process control, target recognition, robotics, computer vision, and cognitive modeling [2,3]. Detail theoretical and empirical comparison of neural network methods and standard statistical techniques for modeling is given in [15,16,5]. Neural networks have also been extensively studied in the context of theoretical foundations of neurocomputing, with search for the fundamental principles of parallel distributed information systems with learning capabilities [2]. The success of neural networks in the data modeling context over wide variety of modeling problems can be attributed to two factors. First is that the basis elements have desirable properties. They are smooth functions of linear functions and nicely bounded above and below. Their form, being close to zero in one portion of the space and close to one in another portion makes them particularly good for approximating conditional probabilities and for approximating local ripples in the data. The other unique element is the idea of simultaneously selecting all basis elements using a learning algorithm like back-propagation [15].

In the following, modeling problem with interval variables is formulated. Given a process whose underlying mathematical form is unknown and can only be observed through its input-output behavior (e.g. how economic factors affect stock prices), the task of modeling is to construct a procedure (model) from the given set of input-output observations, such that the constructed procedure can be used as a predictor for future input values. In case of interval input-output observations the task is to find a functional relation of a response (output) interval variable, \( \bar{y} \) on one or more predictor (input) interval variables \( \bar{x} = (\bar{x}_1, \ldots, \bar{x}_n) \), given a set of \( \mathcal{N} \) interval input-output observations \( \{(\bar{x}_i; \bar{y}_i)\}_{i=1}^\mathcal{N} \), from a physical process, where \( \bar{x} \) is an interval vector with \( n \) interval components, \( \bar{x}_i = (\bar{x}_i^l, \bar{x}_i^u) \), \( i = 1, \ldots, n \). \( \bar{x}_i^l \) and \( \bar{x}_i^u \) are the lower and upper end points of interval \( \bar{x}_i \) respectively.

The process that generates the data is presumed to be described by \( \bar{y} = f(\bar{x}_1, \ldots, \bar{x}_n) + \epsilon \) over some domain \( (\bar{x}_1, \ldots, \bar{x}_n) \in D \subset \mathbb{R}_n \) containing the interval valued data, where \( f \) is the unknown process model to be approximated. The expected value of the random noise \( \epsilon \) is presumed to be zero and reflects the unknown variables that are neither controlled nor observed but affect \( y \). In some problems, \( y \) values are category labels. For example, good or bad, buy or sell etc. In some problems, \( y \) is a continuous variable. In either case, the aim of modeling is to use the given data to construct a function (model) \( f(\bar{x}_1, \ldots, \bar{x}_n) \), that can serve as a reasonable approximation to \( f(\bar{x}_1, \ldots, \bar{x}_n) \) over the domain \( D \) of interest.

The model building process partitions (disjoint) the set of given input-output observations \( \{(\bar{x}_i; \bar{y}_i)\}_{i=1}^\mathcal{N} \), as training set \( \{(\bar{x}_i; \bar{y}_i)\}_{i=1}^{\mathcal{N}_1} \) and validation set \( \{(\bar{x}_i; \bar{y}_i)\}_{i=1}^{\mathcal{N}_2} \), where \( \mathcal{N}_1 + \mathcal{N}_2 = \mathcal{N} \). A model is built using the training set by minimizing the objective function \( L(\bar{\alpha}, \bar{\beta}) \), where \( \bar{\alpha} \) and \( \bar{\beta} \) are the interval-valued model parameters. The prediction performance is then tested using the validation set. Let the sigmoid function \( \sigma(z) = \frac{\exp(z)}{1 + \exp(z)} \). Then build the model by linear combinations of \( \sigma \) (linear combinations of input interval variables), \( \hat{y}(\bar{x}) = \sum_k \bar{\alpha}_k \sigma(\bar{\beta}_k \bar{x}) \). When \( y \) is a continuous variable, estimate the interval parameters \( \bar{\alpha}, \bar{\beta} \) of the model by minimizing,

\[
L(\bar{\alpha}, \bar{\beta}) = \sum_{n=1}^{\mathcal{N}_1} \left( \bar{y}_n - \sum_k \bar{\alpha}_k \sigma(\bar{\beta}_k \bar{x}_n) \right)^2
\]

(1)

In modeling problems, where for each input \( \bar{x} \), the corresponding \( y \) value is any one of the \( J \) different category label i.e., \( y = \{1, \ldots, J\} \), the problem is known as a pattern classification problem. The problem here concerns the construction of a procedure that will be applied to continuing sequences of cases (input vectors), in which each new case must be assigned to one of the set of pre-defined classes on the basis of observed input variables. Such procedure is constructed using a set of data (training) for which the true classes are known. In this problem, for each input vector \( \bar{x} \), the conditional probability for each category is estimated by a function of the form,

\[
\bar{p}(y | \bar{x}) = \sigma \left( \sum_k \bar{\alpha}_k \sigma(\bar{\beta}_k \bar{x}) \right)
\]

(2)

Here, if we define \( x_{y_n} = 1 \) if \( y_n = y \) otherwise 0, then the number of misclassifications are minimized by minimizing,

\[
L(\bar{\alpha}, \bar{\beta}) = \sum_{y,n} \left( x_{y_n} - \sigma \left( \sum_k \bar{\alpha}_k \sigma(\bar{\beta}_k \bar{x}_n) \right) \right)^2
\]

(3)

In the following, a class of flexible nonlinear regression models called Multi-layered Perceptrons (MLP) is introduced followed by the motivation to extend such a class to interval domains. In Section II, this model is extended for nonlinear regression in interval domains with a learning algorithm based the extension of the well known backpropagation learning algorithm [2]. Such an extension is able to handle mixed (intervals and precise numbers) input-output patterns. Section III. gives few examples demonstrating nonlinear interval mapping, finding sets of solutions to the function ap-
approximation problem and performance with noisy data.

B. Artificial Neural Networks

Artificial neural networks (ANN) [2,3] represent an emerging technology studied in many disciplines. They are endowed with the unique attributes of universal approximation (input-output mapping), and the ability to learn from and adapt to their environment. In the most general form, ANN's are non-programmed adaptive information processing systems, that develop associations between objects in response to their environment. ANN's employ a massive interconnection of simple computing cells referred to as "neurons" or processing units. The wide variety of learning laws incorporated, the processing unit's transfer function, the topology of connections, and the weights assigned to these connections, suggest different types of ANN architectures, each suitable for a different task. ANN's acquire knowledge by using a method of encoding information called the learning process. The acquired knowledge is stored in the strengths of interconnections (weights) also called long term memory and is "distributed". One such configuration called multilayered perceptron (MLP) is shown in Figure 1.

A neuron is an information processing unit that is fundamental to the operation of an ANN. An artificial neuron \( j \) takes input a vector, \( \vec{z} = (z_1, \ldots, z_i, \ldots, z_n) \), where \( z_i \) is the activity level of the \( i \)th neuron. Associated with each connected pair of neurons is an adjustable value called weight. The collection of weights that abuts the \( j \)th neuron form a vector, \( \vec{w}_j = (w_{j1}, \ldots, w_{ji}, \ldots, w_{jn}) \), where \( w_{ji} \) represents the connection strength from neuron \( i \) to neuron \( j \). Sometimes there is an additional parameter \( \Theta_j \) modulated by weight \( w_{j0} \), that is associated with the input called threshold. Mathematically,

\[
b_j = f \left( \sum_{i=1}^{n} z_i w_{ji} - w_{j0} \Theta_j \right) .
\]

The commonly used activation functions, \( f(\cdot) \), are linear, ramp, step, and sigmoid or hyperbolic tangent functions.

A large number of these neurons is usually arranged in the form of layers connected through a multiplicity of pathways. Layers are referred to as input, output and hidden layers depending on their interface to the environment. The topology of the connections among and between the layers is an important feature of ANN governing its dynamics. Characteristics of the connection topologies include connection types, connection schemes, and field configurations. There are two primary connection types, excitatory and inhibitory. Three major primary interconnection schemes are intra-layer, inter-layer, and recurrent. Layer configurations include lateral feedback, layer feed-forward, and layer feedback.

Two primary mechanisms used in ANN mapping are auto-associative and hetero-associative mapping. An autoassociator ANN stores the patterns (vectors), \( \vec{z}_1, \vec{z}_2, \ldots, \vec{z}_N \) in the so-called distributed memory consisting of the weight matrix \( W \), of the interconnections. A heteroassociator ANN stores the input and output pattern pairs, \( (\vec{z}_1, y_1), \ldots, (\vec{z}_N, y_N) \), where \( \vec{z}_i \) and \( y_i \) are vectors of \( m \) and \( p \) dimensions respectively.

The recall function takes \( W \), the learned weights (memory) and a \( \vec{z}_i \) as input and returns a response \( \vec{y}_i \) as output. Two primary recall mechanism used in ANN are nearest neighbor recall and interpolative recall. Interpolative recall interpolates, in possibly nonlinear fashion, from the entire set of stored patterns to produced an output.

In learning, shown a set of input observations (perhaps with desired outputs), ANN's learn to approximate and generalize to a certain extent the input and output mapping to produce a consistent response. This ability to learn and generalize is achieved by dynamically changing the underlying connection topology and/or the weights using a learning algorithm. Supervised learning (training) laws incorporate an external teacher, whereas unsupervised learning is more like a self-organization process.
C. Motivation

Interval extension of MLP has been previously proposed in [11] with the motivation of fuzzy regression. In this paper we extend MLP to handle mixed data types (intervals and precise numbers) and to generate set of solutions to the nonlinear regression and pattern classification problems. Several other advantages of such an extension are demonstrated.

The most obvious statistical interpretation of MLP's is that they are a class of flexible nonlinear regression models. These models are developed to handle real-valued information. In many data modeling problems, observations over set of variables are more naturally considered to be interval vectors. This could be due the accuracy of measuring instrument or genuine uncertainty (range of values) in the variable being measured or represent data values over a certain time period (eg. in economics and finance) Flexible nonlinear regression methods like MLP can be extended to handle intervals. Apart from being able to handle intervals, such interval extensions can be further extended to handle fuzzy variable where variables are not only intervals but have a "preference" associated with values within the intervals. As interval arithmetic subsumes classical arithmetic, such modifications of existing techniques may add more flexibility and are in effect generalizations of the original concepts. Example of nonlinear interval mapping is given in Section III. Figure 3.

In many situations, the observed variables are modulated with high frequency noise from unknown sources. One possible alternative in modeling such noisy input-output systems is to convert the noisy input-output real-valued variables as interval variables with interval widths proportional to the noise. Modeling with interval variables in such noisy environments is easier because one can now spend efforts building a model to capture the true underlying structure in the given data and not worry about the over-fitting problem (due to noise) common to most nonlinear non-parametric function approximation methods. This interval transformation of the original problem leads to set of solutions to the regression problem. The sets of solutions thus obtained automatically does the sensitivity analysis on the parameters without any extra computations. Interval extension of MLP automatically deals with such noisy situations without having to transform the problem to interval domain. Example of such a situation is given in Section III. Figure 5b and 7b.

The advantage of interval mathematics to computing is of finding sets containing unknown solutions and to make these sets as small as possible; and to do all this as efficiently as possible. This not only has natural applications in dealing with uncertainty but also can give a set of possible alternative solutions to choose from. In the present context of multi-variable regression and pattern classification, usually a large number of equally good solutions exist. These set of solutions can be found using interval extension of MLP. Section III. Figure 5b and 7b demonstrates this situation.

In many situations in data modeling, "don't care" and missing data over certain variables in few observed examples is common. For example, in building a model from available patient profiles to help diagnose a disease, it is possible that for few patients, certain test results are missing or are "don't care". In many data modeling techniques, for a "don't care" attribute \( D \), one has to include in the training set, all the vectors which results from coding \( D \) with all its possible values. This results in exponential increase in the size of training set if one has more then one "don't care" attributes. In interval MLP the "don't care" or missing value of an attribute is simply coded by using its realistic range as an interval.

II. INTERVAL NEURAL NETWORKS

In the following discussion we consider a widely used class of feed-forward neural network called Multi Layered Perceptrons (MLP) trained using a version of error-correction learning rule (back-propagation) and its interval extension. Typically MLP's consist of 3 or more layers of neurons. Input and output layers interface with the external environment. Typically the interconnections are strictly of feed-forward nature. The signal propagate from input layer to the hidden layer and from hidden layer to output layer (Figure 1. and 2.) Such networks are shown to be universal approximators [12]. The interval learning algorithm is a straightforward extension of the standard back-propagation algorithm with interval weights and biases.

1. Notation

For simplicity we only consider 3 layer MLP with \( n \) neurons in the input layer, \( h \) neurons in the hidden layer, and \( p \) neuron in the output layer, indexed using \( i, j, \) and \( k \) respectively (Figure 2). Also, all input and output interval vectors are assumed to be positive and are normalized in \([0,1]\). Values of weights, activations values of neurons, and biases are updated at every iteration. Iteration index \( t \) is used only where necessary for simplicity. Also sample number index is not used and is assumed in the following discussion.

An \( i \)th interval input vector of \( n \) dimension is represented as \( \bar{x}_i = (\bar{x}_{i1}, \bar{x}_{i2}, \ldots, \bar{x}_{in}) \) (Figure 2). An \( i \)th interval output vector of \( p \) dimension is repre-
sent as \( \tilde{y}_i = (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_p) \). Interval \( \tilde{z}_i = [\tilde{z}_i^L, \tilde{z}_i^U] \) denotes the \( i \)th interval component of the interval vector \( \tilde{z}_i \), \( \tilde{z}_i^L \) and \( \tilde{z}_i^U \) are the lower and the upper end points of an interval \( \tilde{z}_i \) respectively, such that \( \tilde{z}_i^L \leq \tilde{z}_i^U \).

The interval weights from all \( n \) input layer neurons to the \( j \)th hidden neuron are represented by a interval vector \( \tilde{w}_j = (\tilde{w}_{j1}, \tilde{w}_{j2}, \ldots, \tilde{w}_{jn}) \), where \( \tilde{w}_{ji} = [\tilde{w}_{ji}^L, \tilde{w}_{ji}^U] \). Similarly, weights from all \( h \) hidden neurons to an output neuron \( k \) is \( \tilde{w}_k = (\tilde{w}_{k1}, \tilde{w}_{k2}, \ldots, \tilde{w}_{kh}) \), where \( \tilde{w}_{kj} = [\tilde{w}_{kj}^L, \tilde{w}_{kj}^U] \), \( i = 1, \ldots, n; j = 1, \ldots, h \) and \( k = 1, \ldots, p \).

Activations values of the input, hidden and the output layer neurons are represented by \( \tilde{a}_i = [\tilde{a}_i^L, \tilde{a}_i^U] \), \( \tilde{a}_j \), and \( \tilde{a}_k \) respectively, where \( i = 1, \ldots, n; j = 1, \ldots, h \) and \( k = 1, \ldots, p \). The interval biases are denoted as \( \Theta_j = [\Theta_j^L, \Theta_j^U], j = 1, \ldots, h \) and \( \Theta_k, k = 1, \ldots, p \) for the hidden and the output layer nodes respectively. There are no biases for the input nodes.

The net input to each hidden layer neuron \( j \), is represented as, \( \tilde{N}_j = \tilde{N}_j^L, \tilde{N}_j^U \) \( = \sum_{i=1}^{n} \tilde{w}_{ji} \tilde{a}_i + \Theta_j \) similarly, for each output neuron \( k \), the net input \( \tilde{N}_k = \tilde{N}_k^L, \tilde{N}_k^U \) \( = \sum_{j=1}^{h} \tilde{w}_{kj} \tilde{a}_j + \Theta_k \).

The sigmoidal activation function, \( f(\tilde{x}) = (1 + \exp(-\alpha \cdot \tilde{x}))^{-1} \) is used for all the hidden and the output nodes producing interval outputs. Input layer acts as a buffer and has activation function of unity.

The symbol \( \tilde{E}_j = [\tilde{E}_j^L, \tilde{E}_j^U] \) and \( \tilde{E}_k = [\tilde{E}_k^L, \tilde{E}_k^U] \) refer to instantaneous interval error signal at the hidden layer neuron \( j \) and the output layer neuron \( k \) respectively. Also, \( E_j^s \) and \( E_k^s \) is the total error at the output of the hidden layer neuron \( j \) and the output layer neuron \( k \) respectively.

If \( \tilde{a} = [\tilde{a}_1, \tilde{a}_2] \) and \( \tilde{b} = [\tilde{b}_1, \tilde{b}_2] \) are intervals, following interval arithmetic operations are used.

\[
\tilde{a} + \tilde{b} = [\tilde{a}_1 + \tilde{b}_1, \tilde{a}_2 + \tilde{b}_2]
\]

\[
c \cdot \tilde{a} = c \cdot [\tilde{a}_1, \tilde{a}_2]
\]

\[
c \cdot [\tilde{a}_1, \tilde{a}_2] = \begin{cases} 
  [c \tilde{a}_1, c \tilde{a}_2] & \text{if } c \geq 0 \\
  [c \tilde{a}_2, c \tilde{a}_1] & \text{otherwise.}
\end{cases}
\]

\[
\tilde{a} \cdot \tilde{b} = [\tilde{a}_1, \tilde{a}_2] \cdot [\tilde{b}_1, \tilde{b}_2]
\]

\[
= \begin{bmatrix}
  \min(\tilde{a}_1 \tilde{b}_1, \tilde{a}_2 \tilde{b}_1, \tilde{a}_1 \tilde{b}_2, \tilde{a}_2 \tilde{b}_2), \\
  \max(\tilde{a}_1 \tilde{b}_1, \tilde{a}_2 \tilde{b}_1, \tilde{a}_1 \tilde{b}_2, \tilde{a}_2 \tilde{b}_2)
\end{bmatrix}.
\]

Figure 2: Signal flow diagram in interval MLP

.2. Interval back-propagation and its Derivation

Using simple interval arithmetic operations given above, the interval back-propagation algorithms and its derivation is given below. Input layer nodes, hidden layer nodes and output layer nodes are indexed using \( i, j, \) and \( k \) respectively.

- **Forward Pass:**

Assign random intervals from \([-1,1]\) to all initial interval weights \( \tilde{w}_{ji} \) from input to hidden layer, to interval weights \( \tilde{w}_{kj} \) from hidden to output layer and to interval biases \( \Theta_j, j = 1, \ldots, h \), \( \Theta_k, k = 1, \ldots, p \) (Figure 2.)

For each given training pattern pair \((\tilde{x}, \tilde{y}) = (\tilde{x}_1, \ldots, \tilde{x}_n; \tilde{y}_1, \ldots, \tilde{y}_p)\), present the \( \tilde{x} \) values to the input layer neurons. Propagate the input layer neuron activations through \( \tilde{W}_{ji} \) (input to hidden layer interval weight matrix) and calculate the activations \( \tilde{a}_j, j = 1, \ldots, h \), of the hidden layer neurons using,

\[
\tilde{a}_j = [\tilde{a}_j^L, \tilde{a}_j^U] = [f(\tilde{N}_j^L), f(\tilde{N}_j^U)]
\]

\[
= f \left( \sum_{i=1}^{n} \tilde{w}_{ji} \tilde{a}_i - \Theta_j \right).
\]

where \( f(.) \) is sigmoid function. Here we assume that all input interval vectors are positive. As \( f(.) \) is monotonic, \( \tilde{a}_i^L = f(\tilde{N}_j^L) \) and \( \tilde{a}_i^U = f(\tilde{N}_j^U) \).

Propagate the hidden neuron activations \( \tilde{a}_j, j = 1, \ldots, h \), through weights \( \tilde{W}_{kj} \) to the output layer and calculate the activations \( \tilde{o}_k, k = 1, \ldots, p \) of the output layer neurons using,

\[
\tilde{o}_k = [\tilde{o}_k^L, \tilde{o}_k^U] = [f(\tilde{N}_k^L), f(\tilde{N}_k^U)]
\]

\[
\tilde{N}_k^L = \sum_{j=1}^{h} \tilde{w}_{kj} \tilde{a}_j^L - \Theta_k
\]

\[
\tilde{N}_k^U = \sum_{j=1}^{h} \tilde{w}_{kj} \tilde{a}_j^U - \Theta_k
\]

Next, compute the error \( \tilde{E}_k = [\tilde{E}_k^L, \tilde{E}_k^U] \) and \( \tilde{E}_k \) from the computed output \( \tilde{o}_k = [\tilde{o}_k^L, \tilde{o}_k^U] \) and the desired
output \( \tilde{y}_k = [\tilde{y}_k^1, \tilde{y}_k^2] \) of the output layer neurons

\( k = 1, \ldots, p. \)

\[
\tilde{e}_k^1 = \frac{1}{2}(\tilde{o}_k^1 - \tilde{y}_k^1)^2
\]

\[
\tilde{e}_k^2 = \frac{1}{2}(\tilde{o}_k^2 - \tilde{y}_k^2)^2
\]

\[
\varepsilon_k = k_1 \cdot \tilde{e}_k^1 + k_u \cdot \tilde{e}_k^2
\]

where, \( t \) is the current iteration, \( N \) is a large integer and \( 0 < p \leq 4 \). The error is large when \( \delta_k \not\subseteq \tilde{y}_k \) [11]. This error calculation forces the learning algorithm to achieve the inclusion relationship \( \delta_k \subseteq \tilde{y}_k \). This way of error calculations also permits the algorithm to transform the solution discovered before \( N \) iterations to an interval solution (set of solutions).

- **Learning Rule for \( w_{kj} \):**

Error \( E_k = \tilde{e}_k^1 + \tilde{e}_k^2 \) is used to derive a learning rule to adjust interval weights \( \tilde{w}_{kj}, k = 1, \ldots, p \) and \( j = 1, \ldots, h \) between the hidden and the output neurons in the manner similar to the LMS algorithm.

The interval back-propagation algorithm applies a correction \( \Delta \tilde{w}_{kj} = [\Delta \tilde{w}_{kj}^1, \Delta \tilde{w}_{nj}^u] \) to the weights \( \tilde{w}_{kj} \), which is proportional to the instantaneous gradient \( \partial E_k / \partial \tilde{w}_{kj} = [\partial E_k / \partial \tilde{w}_{kj}^1, \partial E_k / \partial \tilde{w}_{kj}^u] \).

\( \partial E_k / \partial \tilde{w}_{kj}^1 \) can be derived as follows.

**Case (i) if \( \tilde{w}_{kj}^1 \geq 0 \) then**

\[
\partial E_k / \partial \tilde{w}_{kj}^1 = \partial \tilde{e}_k^1 / \partial \tilde{w}_{kj}^1
\]

\[
= \frac{\partial \tilde{e}_k^1}{\partial k_1} \frac{\partial k_1}{\partial \tilde{w}_{kj}^1}
\]

\[
= -(\tilde{y}_k^1 - \delta_k^1)\delta_k^1(1 - \delta_k^1)\delta_j^1
\]

\[
= -\delta_k^1\delta_j^1
\]

where,

\[
\delta_k^1 = (\tilde{y}_k^1 - \delta_k^1)\delta_k^1(1 - \delta_k^1).
\]

**Case (ii) if \( \tilde{w}_{kj}^1 < 0 \) then,**

\[
\partial E_k / \partial \tilde{w}_{kj}^1 = -(\tilde{y}_k^1 - \delta_k^1)\delta_k^1(1 - \delta_k^1)\delta_j^1
\]

\[
= -\delta_k^1\delta_j^1
\]

\( \partial E_k / \partial \tilde{w}_{kj}^u \) can be derived as follows.

**Case (i) if \( \tilde{w}_{kj}^u \geq 0 \) then**

\[
\partial E_k / \partial \tilde{w}_{kj}^u = \partial \tilde{e}_k^u / \partial \tilde{w}_{kj}^u
\]

\[
= \frac{\partial \tilde{e}_k^u}{\partial k_1} \frac{\partial k_1}{\partial \tilde{w}_{kj}^u}
\]

\[
= -(\tilde{y}_k^u - \delta_k^u)\delta_k^u(1 - \delta_k^u)\delta_j^u
\]

\[
= -\delta_k^u\delta_j^u
\]

where,

\[
\delta_k^u = (\tilde{y}_k^u - \delta_k^u)\delta_k^u(1 - \delta_k^u).
\]

**Case (ii) if \( \tilde{w}_{kj}^u < 0 \) then,**

\[
\partial E_k / \partial \tilde{w}_{kj}^u = -(\tilde{y}_k^u - \delta_k^u)\delta_k^u(1 - \delta_k^u)\delta_j^u
\]

\[
= -\delta_k^u\delta_j^u
\]

The correction of \( \Delta \tilde{w}_{kj} \) is applied to \( \tilde{w}_{kj} \)

\[
\tilde{w}_{kj}(t+1) = \tilde{w}_{kj}(t) + \Delta \tilde{w}_{kj}(t+1)
\]

where,

\[
\Delta \tilde{w}_{kj}(t+1) = -\eta \frac{\partial E_k}{\partial \tilde{w}_{kj}}
\]

and \( \tilde{w}_{kj}^u \) is given by the above cases.

- **Backward Pass:**

In the above formulation, error \( E_k \) is used to update the weights \( \tilde{w}_{kj} \). For hidden nodes there are no target values available to calculate the error. Similar to the standard back-propagation, backward propagation of error \( E_k \), through the hidden layer to output layer weights \( \tilde{w}_{kj} \), is used to derive the learning rule for the input to hidden layer interval weights \( \tilde{w}_{ji} \). The interval back-propagation algorithm applies a correction \( \Delta \tilde{w}_{ji} = [\Delta \tilde{w}_{ji}^1, \Delta \tilde{w}_{ji}^u] \) to the weights \( \tilde{w}_{ji} \), which is proportional to the instantaneous gradient \( \partial E_k / \partial \tilde{w}_{ji} = [\partial E_k / \partial \tilde{w}_{ji}^1, \partial E_k / \partial \tilde{w}_{ji}^u] \).

- **Learning rule for \( \tilde{w}_{ji} \):**

\( \partial E_k / \partial \tilde{w}_{ji}^1 \) can be derived as follows.

**Case (i) if \( \tilde{w}_{ji}^1 \geq 0 \) then,**

\[
\partial E_k / \partial \tilde{w}_{ji}^1 = \partial \tilde{e}_j^1 / \partial \tilde{w}_{ji}^1
\]

\[
= \frac{\partial \tilde{e}_j^1}{\partial k_1} \frac{\partial k_1}{\partial \tilde{w}_{ji}^1}
\]

\[
= -(\tilde{y}_j^1 - \delta_j^1)\delta_j^1(1 - \delta_j^1)\delta_k^1
\]

\[
= -\delta_j^1\delta_k^1
\]

**Case (ii) if \( \tilde{w}_{ji}^1 < 0 \) then,**

\[
\partial E_k / \partial \tilde{w}_{ji}^1 = -(\tilde{y}_j^1 - \delta_j^1)\delta_j^1(1 - \delta_j^1)\delta_k^1
\]

\[
= -\delta_j^1\delta_k^1
\]

The correction of \( \Delta \tilde{w}_{ji} \) is applied to \( \tilde{w}_{ji} \)

\[
\tilde{w}_{ji}(t+1) = \tilde{w}_{ji}(t) + \Delta \tilde{w}_{ji}(t+1)
\]

where,

\[
\Delta \tilde{w}_{ji}(t+1) = -\eta \frac{\partial E_k}{\partial \tilde{w}_{ji}}
\]

where \( 0 < \eta \leq 1 \) is the learning rate and \( \partial E_k / \partial \tilde{w}_{ji} \) is given by the above cases.
\[ \frac{\partial E_k}{\partial w_{ji}^u} \text{ can be derived as follows.} \]

**Case (i) if** \( w_{kj}^u \geq 0 \) **then,**

\[ \frac{\partial E_k}{\partial w_{ji}^u} = \frac{\partial e_k^u}{\partial w_{ji}^u} = \frac{\partial e_k^u}{\partial \tilde{y}_k^u} \frac{\partial \tilde{y}_k^u}{\partial w_{ji}^u} \]

\[ = (\tilde{y}_k^u - \bar{y}_k^u) \sigma_j^u(1 - \sigma_j^u) \cdot w_{kj}^u \sigma_j^u(1 - \sigma_j^u) \eta_x^i. \]  \hspace{1cm} (29)

**Case (ii) if** \( w_{kj}^u < 0 \) **then,**

\[ \frac{\partial E_k}{\partial w_{ji}^u} = \frac{\partial e_k^u}{\partial w_{ji}^u} = \frac{\partial e_k^u}{\partial \tilde{y}_k^u} \frac{\partial \tilde{y}_k^u}{\partial w_{ji}^u} \]

\[ = -(\tilde{y}_k^u - \bar{y}_k^u) \sigma_j^u(1 - \sigma_j^u) \cdot w_{kj}^u \sigma_j^u(1 - \sigma_j^u) \eta_x^i \]

\[ = -\delta_k^u \delta_j^u(1 - \sigma_j^u) \eta_x^i. \]  \hspace{1cm} (30)

**Case (iii) if** \( w_{kj}^l < 0 \) **then,**

\[ \frac{\partial E_k}{\partial w_{ji}^l} = \left( \frac{\partial e_k^l}{\partial w_{ji}^l} \right) + \left( \frac{\partial e_k^l}{\partial w_{ji}^u} \right) \]

\[ = -(\tilde{y}_k^u - \bar{y}_k^u) \sigma_j^u(1 - \sigma_j^u) \cdot w_{kj}^l \sigma_j^u(1 - \sigma_j^u) \eta_x^i \]

\[ - (\tilde{y}_k^l - \bar{y}_k^l) \sigma_j^l(1 - \sigma_j^l) \eta_x^i \]

\[ = \delta_k^l \delta_j^u(1 - \sigma_j^u) \eta_x^i. \]  \hspace{1cm} (31)

The correction applied to \( w_{ji} \) is,

\[ \tilde{w}_{ji}(t+1) = \tilde{w}_{ji}(t) + \Delta \tilde{w}_{ji}(t+1) \]  \hspace{1cm} (32)

\[ w_{ji}^u(t+1) = w_{ji}^u(t) + \Delta w_{ji}^u(t+1) \]  \hspace{1cm} (33)

where,

\[ \Delta \tilde{w}_{ji}(t+1) = -\eta \frac{\partial E_k}{\partial \tilde{w}_{ji}} \]  \hspace{1cm} (34)

\[ \Delta w_{ji}^u(t+1) = -\eta \frac{\partial E_k}{\partial w_{ji}^u} \]  \hspace{1cm} (35)

where \( 0 < \eta \leq 1 \) and \( \frac{\partial E_k}{\partial \tilde{w}_{ji}} \) and \( \frac{\partial E_k}{\partial w_{ji}^u} \) is given by the above cases.

After adjusting \( w_{kj} \) and \( \tilde{w}_{ji}, i = 1, \ldots, n, j = 1, \ldots, k \) and \( i = 1, \ldots, p \), the next sample is taken from the training set and the same process is repeated. Samples in the training set are used over and over again until the objective function (sum-of-squared errors) is minimized to a desired value.

**III. EXAMPLES**

In the following examples, initial interval weights were initialized in \([-1,1]\). Learning rate, \( \eta = 0.5 \) was used.

**Example 1: Nonlinear Interval Regression:**

In this example inputs are real numbers and outputs are intervals in \([0,1]\). Training data consisted of 32 samples. Testing data consisted of 63 samples. The nonlinear relation between the input and output is shown in Figure 3. Interval outputs are shown as vertical lines. A 1 input, 7 hidden and 1 output node interval MLP was trained for 2000 cycles. The error, \( e_1 = 0.0318 \) and \( e_1^u = 0.0263 \) was observed after 2000 cycles of training. The predicted intervals over the 63 test samples are shown with dotted lines.

**Example 2: Interpolation:**

In this example, inputs are real numbers and outputs are intervals in \([0,1]\). Training data consisted of 8 samples. Testing data consisted of 63 samples. The nonlinear relation between the input and output is shown in Figure 4a. Interval outputs are shown as vertical lines. Notice here the absence of data in the \([0.3, 0.7]\) part of the domain. A 1 input, 7 hidden and 1 output node interval MLP was trained for 2000 cycles. The error, \( e_1 = 0.031368 \) and \( e_1^u = 0.02920 \) was observed after 2000 cycles of training. The predicted output intervals over the 63 test samples in \([0,1]\) are shown with dotted lines in Figure 4a. Results over two standard MLP networks of same architecture trained on lower and upper end of the required output intervals are shown in Figure 4b. Notice here that in the region \([0.3, 0.7]\), where no data is available these two regression approaches have different interpolation effects.

**Example 3: Behavior on Noise data:**

In this example, the given input and output data are both real-valued vectors and is modeled using...
Figure 4: (a) Nonlinear Interval regression using Interval MLP (b) Two independent regression models using standard back-propagation

Figure 5: Interval MLP on noise data. (a) data points (.), fit after 2000 cycles (-.-) and enveloping effect at 2010 cycles (solid lines). (b) three randomly selected point solution from the interval solution producing envelop effect in (a)

interval MLP. This example demonstrates the capability of interval MLP to find a set of equally good possible solutions to nonlinear regression problems. It also avoids the over-fitting problem.

A data set of 51 sample modulated with noise was generated and is shown as dots in Figure 5a. Training set and test set consisted of alternate 26 and 25 sample respectively. A 1 input, 7 hidden and 1 output node interval MLP was trained for 2000 cycles. The error $\tilde{e}_1 = \tilde{e}_2 = 0.082209$ was observed after 2000 cycles of training. The approximated function over all 51 samples is shown with dashed lines in Figure 5a.

The same network when further trained transforms the point solution (Figure 5a. dotted) to an interval solution producing an envelop effect around the given values of y and is shown in the Figure 5a. The envelop has varying width in different parts of the domain depending on the amount of variation in y. This property can be used to model the true underlying structure in the data and ignore the high frequency variation (avoid over-fitting). The result using spline interpolation over the same data is shown in Figure 6. and shows some over-fitting problem.

Due to point-to-interval solution transformation the interval solution not only contains the previously found solution but also multiple solutions of different accuracy to the same problem. This is similar to sensitivity analysis in the parameters of the
approximation and is done automatically. In summary, any possible combination of points in the interval solution vector is a solution to the problem, giving a set of solutions to choose from. This is shown in Figure 5b, where 3 random combinations of parameters chosen from the interval solution and tested over the test set are shown to lie within the envelop. The MSE over these 3 randomly chosen solutions had $\hat{\sigma}_1 = \hat{\sigma}_y = 0.082209, \hat{\sigma}_1 = \hat{\sigma}_y = 0.082299$ and $\hat{\sigma}_1 = \hat{\sigma}_y = 0.080060$.

**Example 4: Set of solutions:**

All pattern classification problems solved using regression methods use a threshold of some type. Thresholding maps a range of values to the same class (e.g. if $y \geq 0.5$ then class is 1 otherwise class is 0). Due to thresholding, many solutions to a pattern classification problem are possible. Here we consider one such simple example.

The geometrical interpretation of a 3 layer MLP (Figure 1.) network with $n$ input, $h$ hidden and $k$ output neurons is that; each hidden node represents a hyperplane in $n-1$ dimensions forming a convex (open or close) region. After training the network, each such region partitions the input space such that the final placement minimizes the classification error of the training examples. The output node functions as logical “AND” or “OR” over the convex regions formed by the hidden nodes. This way class boundaries are formed for the pattern classification task.

Application of interval MLP for approximating such decision function can lead to a vector of interval parameters containing all or large number of possible solutions to the problem as shown in the example below (Figure 7a. and 7b.).

Consider a simple pattern classification problem of 4 patterns, $(\tilde{x}_1; y_1) = (0, 0; 0), (\tilde{x}_2; y_2) = (0, 1; 1), (\tilde{x}_3; y_3) = (1, 0; 1)$ and $(\tilde{x}_4; y_4) = (0, 0; 0)$. The underlying function to be approximated here is the parity (XOR) function. To approximate this function, two lines forming a convex region (open or close) including $\tilde{x}_2$ and $\tilde{x}_3$ are needed as shown in Figure 7a. Many such pairs of lines can form this required convex region and infinitely many solutions to this classification problem exist.

A 2 input, 2 hidden and 1 output interval MLP is used to model the required decision function. Using interval MLP, a point-solution developed after 500 cycles transforms into a set of solutions to the XOR problem after 1400 cycles as shown in Figure 7a. and 7b. and in the table below.
<table>
<thead>
<tr>
<th>$w_{ij}$</th>
<th>at 500</th>
<th>at 1400</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{30}$</td>
<td>[8.7, 8.7]</td>
<td>[9.9, 11.5]</td>
</tr>
<tr>
<td>$w_{31}$</td>
<td>[-5.9, -5.9]</td>
<td>[-7.7, -6.6]</td>
</tr>
<tr>
<td>$w_{32}$</td>
<td>[-5.9, -5.9]</td>
<td>[-7.7, -6.6]</td>
</tr>
<tr>
<td>$w_{40}$</td>
<td>[3.3, 3.3]</td>
<td>[2.9, 4.5]</td>
</tr>
<tr>
<td>$w_{41}$</td>
<td>[-7.7, -7.7]</td>
<td>[-10.5, -7.1]</td>
</tr>
<tr>
<td>$w_{42}$</td>
<td>[-7.9, -7.9]</td>
<td>[-10.5, -7.1]</td>
</tr>
<tr>
<td>$w_{50}$</td>
<td>[-4.2, -4.2]</td>
<td>[-10.0, -3.5]</td>
</tr>
<tr>
<td>$w_{53}$</td>
<td>[9.1, 9.1]</td>
<td>[11.6, 14.9]</td>
</tr>
<tr>
<td>$w_{64}$</td>
<td>[-9.5, -9.5]</td>
<td>[-13.8, -12.9]</td>
</tr>
</tbody>
</table>

Unit 0 is the bias unit, 1 and 2 are the input units, 3 and 4 are the hidden units and 5 is the output unit of the network. Weight $w_{ij}$ is the weight from unit $i$ to unit $j$. Any combination of parameters from the above interval weights is a solution to the XOR problem.

IV. CONCLUSIONS

An interval extension of the classical MLP and backpropagation is discussed in this paper. The utility of such an extension is demonstrated with simple examples of nonlinear interval regressions, finding a set of solutions, and modeling true structure of the data in noisy environments. Interval MLP also has advantage of handling "don't care" values of attributes more gracefully even when the standard data expansion approach. Furthermore, as interval representation are more natural to the representation of subjectivity an imprecision, interval MLP can integrate knowledge in a more flexible way in cases where strong subjectivity and imprecision exists.

This suggests that many of the classical search and optimization algorithms may have interval extensions that could prove to be their useful generalizations [13,14]. Also, interval versions of these search and optimization algorithms can be viewed as precursors to their fuzzy versions. Such interval extensions can be easily extended to fuzzy search and optimization methods containing the original concept as a special case.

REFERENCES

Intervals Neural Networks

Rajendra B. Patil
Computer Research and Applications
CIC-3, MS B256
Los Alamos National Laboratory
Los Alamos, NM 87545