2001

A study of the accuracy, completeness, and efficiency of artificial neural networks and related inductive learning techniques

Craig Garlin Carmichael
Iowa State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A study of the accuracy, completeness, and efficiency of artificial neural networks and related inductive learning techniques

by

Craig Garlin Carmichael

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Major Professor: Eric B. Bartlett

Iowa State University

Ames, Iowa

2001

Graduate College

Iowa State University

This is to certify that the Doctoral dissertation of

Craig Garlin Carmichael

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

Signature was redacted for privacy.

For the Graduate College

DEDICATION

This work is dedicated to my mother-in-law, Marilyn Giles, who sacrificed many hundreds of hours of her time watching our two young daughters so that my wife and I could make it through tough financial waters, and so I could find the inner peace necessary to succeed in this endeavor. Without her enormous help, this document and obtaining my doctoral degree would not have been possible.

TABLE OF CONTENTS

CHAPTER 1. GENERAL INTRODUCTION

The Inductive Learning Hypothesis (ILH) provides a backbone for all inductive learning machines: Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples. Yet, there is ongoing debate regarding the best way to learn and compare hypotheses from limited data [Mitchell, 1997]. In practice, sufficiently large data sets to one observer can seem limited to another. Therefore, regardless of the size of the data at which ILH breaks down and the debate begins, all inductive learning machines should attempt to not only approximate the target function but also assess the uncertainty in the approximation with some probability distribution or confidence interval.

Inductive learners such as feedforward neural networks do not directly provide probability distributions on the output values. With decision trees and other logical representations, the output can be explained as a logical derivation and by appeal to a specific set of cases that support the decision. This has been an elusive task for neural networks [Russell and Norvig, 1995]. Without a way to probe the networks for these supportive cases, a confidence interval on the network cannot be accurately constructed. However, a neural network's output can in fact be explained in terms of a specific set of cases that support the network's decision, as described in the next section. With this information, a confidence interval can be more accurately constructed to estimate the uncertainty in the output in relation to the supportive cases so that:

$$P(\text{decision error} < \text{bound} \mid \text{supportive cases of decision}) \neq P(\text{decision err.} < \text{bound} \ \forall \ \text{cases})$$

Introduction

Artificial Neural Networks (ANNs) have in the past been described as black boxes. This was often an adequate description, since they were far less understood than they are today. A neural network would be trained on the known data, and then its skills in generalization would be tested on some test set. Depending on the sample size and density of the training data, the reliability of the resulting model would be questioned. Every so often it would generate very inaccurate predictions, and no one knew when to trust it. Therefore, until neural networks could accurately estimate their own confidence in their own predictions, they would always be considered as black boxes to statisticians.

Inductive learners such as feedforward neural networks do not directly provide probability distributions on the output values. With decision trees and other logical representations, the output can be explained as a logical derivation and by appeal to a specific set of cases that support the decision. This has been an elusive task for neural networks [Russell and Norvig, 1995]. Without a way to probe the networks for these supportive cases, a confidence interval on the network cannot be accurately constructed.

However, multiple techniques exist for estimating the saliency, importance, relevance, or decision-making power (DMP), of individual elements within the structure of a trained neural network, including the input nodes connected to the normalized data. Various complexity regularization techniques have been set forth in the last decade for this kind of estimate. With a DMP estimate of each input, the supportive cases for a network's decision can be estimated, a confidence interval constructed, and the reliability and confidence in a network's decision increased, resulting in a box that is no longer completely black and mysterious.

Here, the DMP input vector is calculated using five separate methods: weight magnitude, signal variation (new method), input elimination, sensitivity analysis, and second order sensitivity. The input space is then renormalized based on the DMP estimates to scale up important inputs and scale down unimportant ones. Then, a general regression neural network, GRNN, is trained on the modified data to mimic the original neural network model. In this way, the more accurate DMP estimates should result in more accurate GRNN results. This, in turn, directly reveals which saliency estimates are more precise in the calculation of input importance. The five DMP ranking methods are tested against each other using a GRNN on four separate data sets, renormalized according to the DMP calculation of the trained neural network's inputs. If a DMP method can be shown to be universally superior to the other four techniques, then it will most accurately renormalize the sample space to pick the supportive cases of a network's decision, through the local weighting function approximation of a general regression neural network.

If model accuracy is important and the functionality underlying the corresponding data is unknown, then training an artificial neural network to learn the data is a common first step. After this global modeling approach, it should be a natural step to try various local modeling techniques in search of greater accuracy. Karl Steinbuch proposed neural network designs that explicitly remember the training experiences and used a local representation to do nearest neighbor lookup. They used a layer of hidden units to compute an inner product of each stored vector with the input vector. A winner-take-all circuit then selected the hidden unit with the highest activation. This type of network can find nearest neighbors or best matches using a Euclidean distance metric (Atkeson and Schaal, 1995). Here, instead a

straightforward scaling of the inputs based on the trained neural network will allow for an improved weighted distance metric (WDM) to be fed to local weighting methods.

The goal here is to tie together some concepts that are scattered throughout the neural network literature. Inductive learners such as feedforward neural networks do not directly provide probability distributions on the output values. With decision trees and other logical representations, the output can be explained as a logical derivation and by appeal to a specific set of cases that support the decision. This has been an elusive task for neural networks [Russell and Norvig, 1995]. Without a way to probe the networks for these supportive cases, a confidence interval on the network cannot be accurately constructed. Substitute the words "nearest neighbor" for "supportive" and it can be seen that the supportiveness of a training pattern to the current case is directly related to their proximity to each other as viewed by the trained neural network. This is one very good reason why local modeling techniques should supplement artificial neural networks.

Also, according to the literature, the main drawback of a GRNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. A GRNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. However, if a trained ANN is already present, then a very simple approach is available to squash the irrelevant inputs and stretch the relevant ones so that the renormalized data can be fed to local modeling techniques, eliminating the curse of dimensionality. Locally weighted learning is critically dependent on the distance function, and probing the ANN provides a simple solution for this.

Seven methods are described which have varying degrees of local and global properties, three of which are standard (KNN, GRNN, and ANN). After the ANN is probed

and the features (inputs) are weighted (according to the viewpoint of the neural network), the ANN-based WDM is used by KNN and GRNN to produce the methods FWKNN and FWGRNN. The last two methods (LMLR and NNA) are more advanced hybrid networks. Localized multivariate linear regression (LMLR) uses the local pattern-weighting scheme of the FGRNN to allow for multivariate linear regression in the local region specified by the WDM. The Neural Neighbors Algorithm (NNA) is introduced as a way to combine the strengths of FWGRNN, ANN, and LMLR in a stacking approach that is more general, and more computationally intensive, than an ANN. The goal is improved accuracy.
Four test problems are used to compare the performance of each new modeling process. The first problem is derived from the Macky-Glass equations. The second is a chaotic time series consisting of 13 inputs. The third problem is called the spooky particle data set, which is introduced as a candidate benchmark for comparison between inductive learning methods. The fourth is a classification problem.

The Scaled Conjugate Gradient training algorithm (SCG) is an extremely fast training method for the supervised learning of artificial neural networks (ANN), and therefore it is the basis of the generated neural models described in the following chapters. The problem is, however, that SCG by itself doesn't reach its full generalization potential because it will tend to over fit the data unless some adaptive constraint is placed on the number of free parameters in the network. Other complexity regularization methods exist for just this purpose, but most tend to be relatively slow in comparison to the already slow standard of backpropagation. Therefore, the purpose of the neural training algorithms is to derive methods for constraining SCG to areas of weight space that are most likely to produce good ANN models. Both constructive and destructive structural learning approaches are used.

The goal of chapter 2 is to fit the data well in terms of generalization and speed, and then use the lessons learned as a basis for the neural modeling techniques employed throughout the remainder of this dissertation.

Dissertation Organization

The organization of this dissertation is based on the submission of three separate journal articles, which comprise chapters 3-5. Chapter 1 is the general introduction and chapter 6 comprises the general conclusions. Chapter 2, titled "In Search of a Neural Network Cross Validation Driven Regularization," was adapted from the MS Thesis of Carmichael, 1997. Here, the memorization and generalization performance of the scaled conjugate gradient (SCG) training algorithm is benchmarked alongside two automated structural learning techniques based on SCG, regularized by the difference in performance between the training set and cross-validation set. Chapter 3 is titled "Inside the Black Box: Reconstructing a Neural Network's Decisions." Here, trained neural networks are probed for an assessment of the decision-making power of each of their inputs, using 5 separate techniques. These techniques are compared against each other by reconstructing the decisions of the corresponding ANN through a modified GRNN. Chapter 4 is titled "Combining Neural Networks with Nearest Neighbors." Here the best technique from chapter 3 is used to generate weighted distance functions that improve ordinary locally weighted modeling techniques. Hybrid approaches that combine local and global modeling methods are introduced and compared against one another. Chapter 5 is titled "Stacking Diverse Models to Achieve Reliable Error Response Distributions." After local methods are

improved with the ANN-based weighted distance metric (introduced in chapter 3, and reused in chapter 4), the supportiveness of the training data in the local neighborhood is described in a way that demonstrates how to construct accurate confidence intervals about the neural network output. Also, a framework is put in place to compare the accuracy of the modeled confidence intervals. This entire process is a study of the accuracy, completeness, and efficiency of artificial neural networks and related inductive learning techniques. It is intended to transform neural networks into something that is less of a black box. The appendix is for reference, which apply techniques discussed in chapters 2-5 to real world problems.

Literature Review

This section describes the major techniques used in the field of machine learning: function approximation, neural network, non-parametric regression (local modeling), k-Nearest Neighbor (KNN), lazy learning, Bayesian learning, statistical learning, and support vector machines (SVM). The chapters that follow will delve into detailed discussion involving many of these modeling techniques.

Function approximation

Many general learning tasks, especially concept learning, may be regarded as function approximation. The aim is to find a hypothesis (a function as well), that can be used for predicting the function values of yet unseen instances, e.g. to predict future events. Good hypotheses are those that often predict an accurate function value. The quality of a

hypothesis for approximating a specific function is measured by a loss function, increasing as the differences between predicted and true function values increase, where it is increasingly important to predict well on frequently observed instances. The aim of function approximation is to find a hypothesis that minimizes the expected error (or expected risk) problems involving classification (the examples are divided into a given set of classes) and/or regression (a real value function shall be approximated), where a loss function is chosen such as the squared difference between the predicted value h(x) and the correct value t(x).

Neural networks

A neural network is composed of a number of nodes connected by links. Each link has a numeric weight associated with it. Weights are the primary means of long-term storage in the network, and learning takes place by updating the weights. Some of the nodes are connected to the external environment, and can be designated as input or output nodes. Artificial Neural Networks (ANN) form a family of inductive techniques that mimic biological information processing. Most animals possess a neural system that processes information. Biological neural systems consist of neurons, which form a network through synaptic connections between axons and dendrites. With these connections, neurons exchange chemo-electrical signals to establish behavior. Neural networks can adapt their behavior by changing the strength of these connections; more complex learning behavior can occur by forming new connections. Changes in connections, and new connections, grow under influence of the signal exchange between neurons.

Artificial neural networks are simplified models of the biological counterpart. They consist of the following elements: Processing units (models of neurons), weighted

interconnections (models of neural connections), an activation rule, to propagate signals through the network (a model of the signal exchange in biological neurons), and a learning rule, specifying how weights are adjusted on the basis of the established behavior. The most common ANN are structured in layers with an input layer where data is coded as a numeral pattern, one or more hidden layers to store intermediate results, and an output layer that contains the output result of the network.

In contrast to symbolic techniques such as decision trees the representation of knowledge in a neural network is not easy to comprehend. The set of weights in a network, combined with input and output coding, realizes the functional behavior of the network. Such data does not give humans an understanding of the learned model. An alternative approaches for representing a neural network is showing the weight matrix. This is not very informative, but it illustrates why a neural network is called sub-symbolic. The knowledge is not represented explicitly, but contained in a distributed representation over many weight factors. A second alternative approach for representing a neural network is to show that a neural network performs well on the concepts in the data set, even better than other techniques, and do not try to interpret de model. Techniques exist to extract the model captured in a neural network, e.g. by running a symbolic technique on the same data or by transferring a NN model into an understandable form. Techniques exist to extract statistical models and symbolic models from NN.

Backpropagation

The backpropagation algorithm is applied to feedforward networks and uses the Delta rule, which starts with the calculated difference between the actual outputs and the desired

outputs. Using this error, connection weights are increased in proportion to the error times a scaling factor for global accuracy. But the system must determine which input contributed the most to an incorrect output in order to correct the error (error is corrected by changing the weights of that element). To solve this problem, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During the learning process, a forward sweep is made through the network, and the output of each element is computed layer by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function, and the connection weights are normally adjusted using the Delta Rule. This process proceeds for the previous layer(s) until the input layer is reached.

Backprop with momentum (enhanced backpropagation)

An enhanced version of backpropagation uses a momentum term and flat spot elimination. The momentum term introduces the old weight change as a parameter for the computation of the new weight change. This avoids oscillation problems common with the normal backpropagation algorithm by helping the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum. The new weight change is computed by a constant specifying the influence of the momentum. The effect of these enhancements is that flat spots of the error surface are traversed relatively rapidly with a few big steps (learning speed is increased significantly as a result of this), while the step size is decreased, as the surface gets rougher.

Scaled Conjugate Gradient Algorithm

The Scaled Conjugate Gradient Algorithm [Moller, 1993], SCG, is a derivative-based learning algorithm that uses a second order approximation along a conjugate gradient direction $\tilde{p}_k$ to estimate the step size α_k from a point in weight space $\tilde{w}_k$ needed to reach a minimum in RMS along that direction during iteration k. This is similar to second-order backprop, where a direction and a step size are calculated. The conjugate gradient direction is a cumulative function of steepest descent vectors from previous iterations and the current steepest descent vector $\tilde{r}_k$. The algorithm combines the model-trust region approach with the conjugate gradient approach in order to give better convergence. This introduces a scalar λ_k that scales the Hessian matrix in an artificial way. The SCG algorithm is described as follows:

1. Choose weight vector $\tilde{w}_1$ and scalars $0 < \sigma \le 10^{-4}, 0 < \lambda_1 \le 10^{-6}, \bar{\lambda}_1 = 0$.

   Set $\tilde{p}_1 = \tilde{r}_1 = -E'(\tilde{w}_1)$, $k = 1$, and *success* = *true*

2. If *success* = *true*, then calculate second order information:

$$\sigma_k = \sigma / |\tilde{p}_k|$$

$$\tilde{s}_k = \frac{E'(\tilde{w}_k + \sigma_k \tilde{p}_k) - E'(\tilde{w}_k)}{\sigma_k}$$

$$\delta_k = \tilde{p}_k^T \tilde{s}_k$$

3. Scale $\delta_k : \delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|\tilde{p}_k|^2$.

4. If $\delta_k \le 0$ then make the Hessian matrix positive definite:

$$\bar{\lambda}_k = 2\left(\lambda_k - \frac{\delta_k}{|\tilde{p}_k|^2}\right)$$

$$\delta_k = -\delta_k + \lambda_k |\tilde{p}_k|^2$$

$$\lambda_k = \bar{\lambda}_k$$

5. Calculate the step size:

$$\mu_k = \tilde{p}_k^T \tilde{r}_k$$

$$\alpha_k = \frac{\mu_k}{\delta_k}$$

6. Calculate the comparison parameter:

$$\Delta_k = \frac{2\delta_k [E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_k \tilde{p}_k)]}{\mu_k^2}$$

7. If $\Delta_k \geq 0$ then a successful reduction in error can be made:

$$\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k$$
$$\tilde{r}_{k+1} = -E'(\tilde{w}_{k+1})$$
$$\bar{\lambda}_k = 0$$
$$success = true$$

If (k mod $N = 0$) then restart algorithm:

$$\tilde{p}_{k+1} = \tilde{r}_{k+1}$$

Else

$$\beta_k = \frac{|\tilde{r}_{k+1}|^2 - \tilde{r}_{k+1}^T \tilde{r}_k}{\mu_k}$$
$$\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$$

End if

If $\Delta_k \geq 0.75$, then reduce the scale parameter:

$$\lambda_k = \tfrac{1}{4}\lambda_k$$

End if

Else

(A successful reduction in error cannot be made)

$\bar{\lambda}_k = \lambda_k$
success = *false*

End if

8. If $\Delta_k \leq 0.25$, then increase the scale parameter:

$$\lambda_k = \lambda_k + \left(\frac{\delta_k (1 - \Delta_k)}{|\tilde{p}_k|^2} \right)$$

9. If the steepest descent direction $\tilde{r}_k \neq 0$, then set $k = k + 1$ and go to 2, else terminate and return $\tilde{w}_k$ as the desired minimum.

Structural learning

Structural learning methods attempt to maximize the generalization potential of an ANN model by simultaneously minimizing both the training set RMS and the number of trainable parameters. They can be roughly classified into two categories: *destructive* learning and *constructive* learning. Since the former starts with a large-sized network and the latter starts with a small-sized one, the computational cost for the former is larger than that for the

latter. On the other hand, the former is expected to have larger generalization ability than the latter, because the former optimizes the entire connection weights simultaneously, whereas the latter optimizes a part of them sequentially. Although destructive methods can be further broken down into different classes, the important criterion for an effective structural learning method is the effective minimization of both the training set RMS and the number of trainable parameters while doing so in an amount of time somewhat comparable to that of training without structural learning. In structural learning with forgetting [Ishikawa, 1996], the algorithm is composed of three algorithms: Learning with forgetting, hidden units clarification, and learning with selective forgetting. These fall in the category of destructive learning approaches, where weight decay (pruning) is combined with a methodology to force each hidden unit to be fully active or inactive.

The Cascade Correlation Learning Algorithm [Fahlman & Lebiere, 1990] is one of the earliest dynamic neural (constructive) learning algorithms. The algorithm starts with a minimal architecture and attempts to build an optimal architecture by adding one hidden neuron at a time. The added neurons are never removed. It starts with only the inputs and outputs and no hidden neurons. Each iteration the current network is trained with a standard learning algorithm like backprop or SCG. After this training, the algorithm generates some candidate units. These candidate units have no connections to the output neurons but have connections to all the inputs and hidden neurons. These new connections are trained with some standard learning algorithm to maximize the correlation between the candidate units' output and the error function of the neural network. This training is stopped when there is no significant improvement in the above correlations. The candidate unit with the maximum correlation is selected and added to the network as a hidden neuron. The weights coming into

this neuron are not changed from this point and connections are added to the output neurons. After the neuron is added the weights from this neuron to the outputs are modified with some standard algorithm. The above process is repeated until some desired accuracy is reached.

An improvement of the cascade correlation learning algorithm is suggested by Phatak & Koren (1994). In the original algorithm each new neuron added is connected to all the inputs, outputs and also other hidden neurons. In effect this is like adding a new layer each time a neuron is added. The algorithm suggested by them adds neurons in a layer-by-layer basis. There is a limit to the number of neurons in a layer. Only after a layer is full another layer is created. When creating a new layer the output neurons are made part of the new layer, a new output layer is created and the old and new outputs are all connected. The weights between old and new weights are first trained before adding new neurons. The tradeoffs of this approach are also discussed in the paper. A modified cascade correlation algorithm for classification is suggested in [Lehtokangas, 2000]. In the ordinary cascade correlation learning algorithm each time a new neuron should be added, some candidate neurons are considered, the weights into them trained and the one with the best value for the covariance criteria is added to the network. But this is computationally expensive. A fast weight initialization for cascade correlation learning based on stepwise regression is suggested in [Lehtokangas, 2000].

Kwok and Yeung (1997) surveys various network growing algorithms for regression problems. Apart from various dynamic node creation algorithms, which start from a small architecture and keep adding hidden neurons, algorithms based on projection pursuit regression and cascade correlation learning are reviewed. A Dynamic Node Architecture algorithm using information theory is proposed by Bartlett (1994). In this algorithm

information theory is used to find the importance of a hidden neuron. The algorithm starts with a small architecture, keeps adding neurons until a target is reached and then removes least important neurons to find the smallest architecture which can learn to the desired accuracy. Setiono and Hui (1995) describe a network construction algorithm using a Quasi-Newton method. This algorithm is similar to other dynamic node creation algorithms except that it uses a variant of the quasi-Newton method as the algorithm to learn with a given architecture. A constructive backpropagation algorithm based on the cascade correlation learning algorithm is proposed by Lehtokangas (1999). In this algorithm a new hidden neuron in connected to both the inputs and outputs and these weights are trained by backpropagation algorithm. After finishing training these weights (both inputs to hidden and hidden to outputs) are kept frozen. New neurons are added until the desired accuracy is reached. Carmichael (1997) in his thesis proposes a new importance function for the hidden neurons and weights. A measure of local importance of a weight into a neuron, in comparison to other weights into the same neuron, is based on the weight and the variance of the signal entering the weight over all the patterns. From this local importance, a global importance is calculated which estimates the importance of this weight or neuron to the network as a whole. A cross validation set is used in the algorithm to measure how well each ANN architecture is doing.

Various pruning algorithms are surveyed in [Reed, 1993]. The pruning algorithms start with a high architecture, larger than required, and then remove unimportant weights and/or nodes. In the paper, the pruning algorithms are divided into two broad groups. In one group, the algorithm measures the sensitivity of the error function to the removal of weights and/or nodes of the network. In the other, algorithms add a penalty term to the error function

of the network to penalize unimportant weights. The algorithms described in the paper differ in how the importance of the elements of the network is calculated and in the penalty terms added to the error function.

In Optimal Brain Damage [Le Cun et al, 1990] the importance or saliency of a weight is measured by estimating the second derivative of the error function with respect to the weight. The algorithm assumes that all the off-diagonal terms of the Hessian matrix are zero. The diagonal terms can be calculated by a modified backpropagation rule. The Optimal Brain Surgeon algorithm [Hassibi and Stork, 1993] is a further development of the Optimal Brain Damage algorithm. In this algorithm the Hessian matrix is fully calculated iteratively. But this requires calculation of the inverse of the Hessian matrix that is computationally very expensive. In the network pruning method described in [Mozer and Smolensky, 1989], the saliency or relevance of a unit in the network is the difference in the error when the unit is removed and when the unit is in the network. Instead of calculating the relevance of each unit in the network, the relevance is approximated by using a gating term for each unit. When the relevance of a unit falls below a threshold that unit is removed. Karnin (1990) also measures the relevance of a unit in a similar way but simplifies the calculation of the relevance by using the terms already available in the forward and backward pass and avoiding a separate relevance calculation pass. Dantaluri (2000) gives a review of various constructive and destructive structural learning approaches. Here, dynamic node architecture heuristics generate optimal network architectures through a modified cost function.

Non-parametric regression

Non-parametric regression belongs to a data analytic methodology usually known as local modeling [Fan, 1995]. The basic idea behind local regression consists of obtaining the prediction for a data point x by fitting a parametric function in the neighborhood of x. This means that these methods are "locally parametric." According to Cleveland and Loader (1995) local regression traces back to the 19th century. These authors provide a historical survey of the work done since then. The modern work on local modeling starts in the 1950's with the kernel methods introduced within the probability density estimation setting [Rosenblatt, 1956; Parzen, 1962] and within the regression setting [Nadaraya, 1964; Watson, 1964]. Local polynomial regression is a generalization of this early work on kernel regression. In effect, kernel regression amounts to fitting a polynomial of degree zero (a constant) in a neighborhood. Summarizing, we can state the general goal of local regression as trying to fit a polynomial of degree p around a query point (or test case) x_q using the training data in its neighborhood. This includes the various available settings like kernel regression (p=0), local linear regression (p=1), etc. Local regression is strongly related to the work on instance-based learning [Aha, 1991], within the machine learning community. Given a case x for which we want to obtain a prediction, these methods use the training samples that are "most similar" to x to obtain a local model that is used to obtain the prediction. This type of inductive methodologies do not perform any kind of generalization of the given data and "delay learning" until prediction time.

k-Nearest Neighbor algorithm

The k-nearest neighbor approach (KNN) is a simple algorithm that stores all available examples and classifies new instances of the example language based on a similarity measure. A variant of this algorithm addresses the task of function approximation. Examples are described by numerical attribute-values. The complete example set is simply stored in the "training phase". Calculations are delayed until queries occur, no hypothesis in the usual sense is returned. Hypotheses are implicit defined by the stored example set and the rules new instances are classified by. If there are n attributes all vectors can be interpreted as instances of $|R^n$. The distance $d(\mathbf{x}_1, \mathbf{x}_2)$ of two example vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ is defined as their usual vector distance. The distance between two example vectors is regarded as a measure for their similarity. To classify a new instance e from the set of stored examples, the k examples most similar to e are determined. The new instance is assigned the class that most of the k examples belong to. This approach is suited for function approximation as well. Instead of assigning the most frequent classification among the k examples most similar to an instance e, an average of the function values of the k examples is calculated as the prediction for the function value e. A variant of this approach calculates a weighted average of the nearest neighbors, which is similar in nature to general regression neural networks (GRNN). Given a specific instance e that shall be classified, the weight of an example increases with increasing similarity to e. A major problem of the simple approach of KNN is that the vector distance will not necessarily be suited for finding intuitively similar examples, especially if irrelevant attributes are present. Therefore, with the performance of KNN being very sensitive to the chosen distance function, a non-Euclidean distance metric is often more suitable.

Weight Adjusted *k*-Nearest Neighbor algorithm

A major drawback of the similarity measure used in KNN is that it uses all features in computing distances. In many data sets, only a smaller number of the total inputs may be useful in classification or function mapping. A possible approach to overcome this problem is to learn weights for different features before feeding this modified distance metric to KNN. Here, the Weight Adjusted k-Nearest Neighbor (WAKNN) classification algorithm that is based on the k-NN classification paradigm. In WAKNN [Karypis and Kumar, 1991] the weights of features are learned using an iterative algorithm. In the weight adjustment step, the weight of each feature is perturbed in small steps to see if the change improves the classification objective function. The feature with the most improvement in the objective function is identified and the corresponding weight is updated. The feature weights are used in the similarity measure computation such that important features contribute more in the similarity measure. The drawback is that each perturbation can be very computationally expensive as the number of patterns grows.

General regression neural network

A general regression neural network (GRNN) [Specht, 1991] can be thought of as the weighted average variant of KNN described above. It is a feed-forward neural network based on Nadaraya-Watson kernel regression, also reinvented in the neural network literature by Schioler and Hartmann. (Kernels are also called "Parzen windows".) They can be thought of as normalized RBF networks with a hidden unit centered at every training pattern. These RBF units are called "kernels" and are usually Gaussian pdf's. The output is just a weighted average of the target values of the training patterns (cases) close to the given input

pattern, the closeness being determined by the Euclidean distance between the two patterns. For a GRNN, the expected (modeled) value(s) of the output(s) given the input(s) are determined by:

$$E[y \mid \vec{x}] = \frac{\int_{-\infty}^{\infty} y \cdot f(\vec{x}, y)\, dy}{\int_{-\infty}^{\infty} f(\vec{x}, y)\, dy}$$

Where y is the output and $\vec{x}$ is the input vector. The above representation for a GRNN that models a finite set of points can be reduced to the following discrete formula for an estimation of $E[y \mid \vec{x}]$.

$$E[y \mid \vec{x}] \approx y(\vec{x}) = \frac{\sum_{p=1}^{n} v_p y_p}{\sum_{p=1}^{n} v_p}$$

Where v_p is defined as

$$v_p = \exp\left(\frac{-D_p^2}{2\sigma_G^2}\right)$$

Also, D_p^2 is the Euclidean distance metric between the current $\vec{x}$ and that of pattern p in the training set, defined as

$$D_p^2 = (\vec{x} - \vec{x}_p)^T (\vec{x} - \vec{x}_p)$$

The single free parameter σ_G is determined by minimizing the predicted error sum of squares (PRESS) in an N-folding process. Where an artificial neural network doesn't determine the supportive cases of its decision explicitly, a GRNN does this directly through the weighted averaging process, so that there is v_p support for case p in relation to $\vec{x}$. The main drawback of a GRNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. A GRNN cannot ignore irrelevant inputs without major modifications to the basic algorithm.

Lazy learning

The notion of lazy learning [Aha, 1997] subsumes a family of algorithms, that store the complete set of given (classified) examples of an underlying example language and delay all further calculations, until requests for classifying yet unseen instances are received. The time required for classifying an unseen instance will be higher, if all calculations have to be done when (and each time) a request occurs. The advantage of such algorithms is, that they do not have to output a single hypothesis, assigning a fixed class to each instance of the example language, but they can use different approximations for the target concept/function, which are constructed to be locally good. In this way examples similar to a requested

instance receive higher attention during the classification process. This method requires a similarity measure, to evaluate the importance of examples, for classifying an unseen instance. If examples are described by a real-valued vector, then similarity could be measured by the usual vector distance, which raises the question, if all attributes should have the same impact. To reduce the unbeneficial impact of irrelevant attributes, and to assign each attribute the proper degree of impact, are key issues, using this method.

Bayesian learning

Bayesian Learning constitutes a probabilistic view of learning, based on Bayes Theorem. The underlying assumption is, that there is a set of hypotheses, each having a certain probability of being correct. Receiving more information changes the probabilities from a learner's point of view. For instance an observation might contradict a hypothesis, or strengthen the belief in it. The aim in this setting is to be able to find a hypothesis with highest probability of being correct, given a specific set of data / piece of information.

Statistical learning

Given a set of training examples the error of a learning result is estimated by the empirical error, a measure based on the training data. The statistical learning theory focuses on two major questions. First is the asymptotic analysis: Can it be proven, that with an increasing number of examples the empirical error converges to the real error? Second, is the question regarding learning rate: if point (question) 1 has been proved, then how fast does the empirical error converge to the real error?

Support vector machine (SVM)

The Support Vector Machine is a new type of learning machine for pattern recognition and regression problems that constructs its solution in terms of a subset of the training data, the Support Vectors. Learning can be thought of as inferring regularities from a set of training examples. Much research has been devoted to the study of various learning algorithms that allow the extraction of these underlying regularities. If the learning has been successful, these intrinsic regularities will be captured in the values of some parameters of a learning machine; for a polynomial classifier, these parameters will be the coefficients of a polynomial, for a neural net they will be weights and biases, and for a radial basis function classifier they will be weights and centers. This variety of different representations, however, conceals the fact that no matter how different the outward appearance of these algorithms is, they all must rely on intrinsic regularities of the data. The Support Vector Learning Algorithm (Boser, Guyon & Vapnik, 1992, Cortes & Vapnik, 1995) is a promising tool for studying these regularities (i.e. for studying learning) in pattern classification. It allows the construction of various learning machines by the choice of different dot products. Thus the influence of the set of functions that can be implemented by a specific learning machine can be studied in a unified framework. It builds on results of statistical learning theory, namely on the structural risk minimization principle (Vapnik, 1979) guaranteeing high generalization ability. Thus there is reason to believe that decision rules constructed by the support vector algorithm do not reflect incapabilities of the learning machine (as in the case of an overfitted artificial neural network) but rather regularities of the data.

Support vector machines were developed by Vapnik et al. based on the Structural Risk Minimization principle from statistical learning theory. They can be applied to

regression, classification, and density estimation problems. In their basic form, SVM classifiers learn binary, linear decision rules described by a weight vector w and a threshold b. According to which side of the hyperplane the attribute vector **x** lies on, it is classified into class +1 or -1. The idea of structural risk minimization is to find a hypothesis **h** for which one can guarantee the lowest probability of error. For SVMs, Vapnik shows that this goal can be translated into finding the hyperplane with maximum margin for separable data. For separable training sets SVMs find the hyperplane **h**, which separates the positive and negative training examples with maximum margin. The examples closest to the hyperplane are called Support Vectors. Popular kernel functions are polynomial classifiers, radial basis function (RBF) classifiers, and two layer sigmoid neural nets. Such kernels calculate an inner-product in some feature space.

References

Aha, D. (1997). *Lazy Learning.* Kluwer Academic Publishers.

Aha, D., Kibler, D., Albert, M. (1991). "Instance-based learning algorithms," *Machine Learning 6*, 37-66.

Atkeson, C. G. and S. Schaal (1995). "Memory-based neural networks for robot learning," *Neurocomputing*, Vol 9, 1-27.

Atkeson, C. G., A.W. Moore, and S. Schaal (1997). " Locally Weighted Learning," *Artificial Intelligence Review*, Vol 11, 11-73.

Bartlett, E. B. (1994). "Dynamic node architecture learning: an information theoretic approach", *Neural Networks 7*, 129-140.

Bartlett, E. B. and A. Basu (1991). "A dynamic node architecture scheme for back-propagation neural networks," *Intelligent Engineering Systems Through Artificial Neural Networks*, 101-106.

Bartlett, E. B. and A. Whitney (1999). "On The Use Of Various Input Subsets For Stacked Generalization," *Intelligent Engineering Systems Through Artificial Neural Networks*, Vol 9, 117-124.

Bartlett, E. B. and K. Kim, (1993). "Error Bounds on the Output of Artificial Neural Networks," *ANS Trans.*, 69,197-199.

Bartlett, E. B. and R. E. Uhrig (1992). "Nuclear power plant status diagnostics using an artificial neural network," *Nuclear Technology* 97, 272-281.

Baum, E. B. (1989). "What size net gives valid generalization," *Neural Computation, vol. 1*, pp. 151-160.

Bhat, N., and T. McAvoy, (1990). "Use of Neural Nets for Dynamic Modeling And Control of Chemical Process Systems," *Computers Chem. Engng.* 14, 573-583.

Blum, E. K. and L. K. Li, (1991). "Approximation Theory and Feedforward Networks," *Neural Networks*, 4, 511.

Bryson, A. and Y. Ho (1969). *Applied Optimal Control*, Blaisdell.

Carmichael, C. G. (1997). "Experiments in advancing supervised-learning ANN techniques," Thesis(M.S), Iowa State University.

Cherkassky, S. and F. Mulier, (1998). *Learning From Data: Concepts, Theory, and Methods (Adaptive and Learning Systems for Signal Processing, Communications and Control Series).* John Wiley & Sons, New York, New York.

Cleveland, W. and C. Loader (1995). "Smoothing by Local Regression: Principles and Methods (with discussion)," Computational Statistics.

Cortes, C. and V. Vapnik (1995). "Support Vector Networks," *Machine Learning 20*, pp. 273-297

Cun, Y. Le, J. S. Denker, and S. A. Solla (1990). "Optimal brain damage," Advances in Neural Information Processing Systems 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 598-605.

Cybenko, G. (1989). "Approximation by superposition of a sigmoidal function," *Mathematics of Control, Signals, and Systems 2,* 303-314.

Dantaluri, Varma (2000). "A cost function based dynamic node architecture algorithm," Thesis(M.S.), Iowa State University.

Fahlman, S. and C. Lebiere (1990). "The cascade-correlation learning architecture," in Advances in Neural Information Processing Systems 2, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp.524-532.

Fan, J. (1995). "Local Modeling," *Encyclopedia of Statistical Science.*

Frean, M. (1992). "The upstart algorithm: A method for construction and learning feedforward neural networks," *Neural Computation*, vol. 4, pp. 946-957.

Geman, S., E. Bienenstock, and R. Doursat (1992). "Neural Networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1-58.

Guyon, I., V. Vapnik, B. Boser, L. Bottou, and S.A. Solla (1992). "Structural risk minimization for character recognition," *Advances in Neural Information Processing Systems 4*, 471-479, San Mateo CA, Morgan Kaufmann.

Han, E., Karypis, G., Kumar, V. (1991). "Text Categorization Using Weight Adjusted k-Nearest Neighbor Classication," *Proceedings of The Twelfth International Joint Conference on Artificial Intelligence.*

Hassibi, B. and D. G. Stork (1993). "Second-order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5*. San Mateo, CA: Morgan Kaufmann, pp. 164-171.

Haykin, S., (1999). *Neural Networks: A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, New Jersey.

Hecht-Nielsen, R. (1990). *Neurocomputing.* Addison-Wesley, Reading, Mass.

Ishikawa, M. (1996). "Structural Learning with Forgetting," *Neural Networks*, Vol. 9, No. 3, pp. 509-521.

Johansson, E., F. Dowla and D. Goodman (1991). "Back-propagation learning for multi-layer feed-forward neural networks using the conjugate gradient method," *International Journal of Neural Systems*, *vol. 2*, pp. 291-302.

Karnin, E. D. (1990). "A simple procedure for pruning back-propagation trained neural networks," *IEEE Trans. Neural Networks*, *vol. 1*, pp. 239-242.

Kim, K. and E. B. Bartlett, (1996). "Nuclear Power Plant Fault Diagnosis Using Neural Networks with Error Estimation by Series Association," *IEEE Transactions on Nuclear Engineering Vol 43*, No 4, pp 2373 - 2388

Krogh, A. and J. A. Hertz (1992). "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds., pp. 951-957.

Kurkova, V. (1992). "Kolmogorov's theorem and multilayer neural networks," *Neural Networks*. 5,501-506

Kwok, T. Y. and D. Y. Yeung (1997). "Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems," IEEE Trans. Neural Networks, vol. 8, pp. 630-645.

Lapades, A., and R. Farber, (1987). *Nonlinear signal processing using neural networks: prediction and system modeling.* Los Alamos National Laboratory Technical Report LA-UR-87-2662.

Lehtokangas, M. (1999). "Fast Initialization for Cascade-Correlation Learning," *IEEE Trans. Neural Networks*, vol. 10, pp. 410-414.

Lehtokangas, M. (2000). "A Modified Cascade-Correlation Learning for Classification," *IEEE Trans. Neural Networks*, vol. 11.

Lippmann, R. P., (1987). "An Introduction to Computing with Neural Nets," *IEEE Acoustics Speech and Signal Processing Magazine*, 4, 4.

Mackey, M. C. and L. Glass (1977). "Oscillations and Chaos in Physiological Control Systems," *Science*, 197, 287-289.

Mandilwar, D. K. and H. Qammar (1993). "Prediction of Chaotic Time Series: Neural Versus Fuzzy", *Proceedings of the 1993 International Fuzzy Systems and Intelligent Control Conference,* 189-199.

Miller, W. T., R. S. Sutton, and P. Werbos (Eds.), (1990). *Neural Networks for Control.* Press. Cambridge, Mass.

Mitchell, T., (1997). *Machine Learning.* McGraw-Hill, New York, New York.

Moller, M.F. (1993). "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", *Neural Networks 6*, 525-533.

Moody, J. E. (1992). "The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems," in *Advances in Neural Information Processing Systems 4*, San Mateo, CA: Morgan Kaufmann, pp. 847-854.

Mozer, M. C. and P. Smolensky (1989). "Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in Neural Information Processing Systems, 1, D. S. Touretzky , Ed. (Denver 1988), pp. 107-115.

Nadaraya, E.A. (1964). "On estimating regression", Theory Probab. Applic.10, 186-90.

Narendra, K. S. and K. Parthasarathy, (1990). "Identification and Control of Dynamic Systems Using Neural Networks," *IEEE Trans. Neural Networks,* 1, no. 1, 4.

Parzen, E. (1962). "On estimation of a probability density function and mode," *Annals Mathematical Statistics 33*, 1065-1076.

Phatak, D. and I. Koren (1994). "Connectivity and performance tradeoffs in the cascade correlation learning architecture," *IEEE Trans. Neural Networks, vol. 5*, pp. 930-935.

Reed, R. (1993). "Pruning Algorithms – A Survey," *IEEE Trans. Neural Networks, vol. 4*, pp. 740-747.

Rosenblatt, M. (1956). "Remarks on some nonparametric estimates of a density function," *Annals Mathematical Statistics 27*, 832-837.

Rumelhart, D., G. Hinton and R. Williams (1986). "Learning representations of back-propagation errors," *Nature*(London), vol. 323, pp. 533-536.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, (1986). *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol. 1 & 2, MIT Press, Cambridge, Massachusetts.

Russell, S. and P. Norvig, (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey.

Schioler, H. and U. Hartmann (1992). "Mapping Neural Network Derived from the Parzen Window Estimator," *Neural Networks*, 5, 903 - 909.

Schmitz, G. P. J. and C. Aldrich (1999). "Combinatorial Evolution of Regression nodes in Feedforward Neural Networks," *Neural Networks*, 12, 175-189.

Setiono, R. and L. C. K. Hui (1995). "Use of a quasi-Newton method in a feedforward neural network construction algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 273-277.

Sietsma, J. and R. J. F. Dow (1991). "Creating artificial neural networks that generalize," Neural Networks, vol. 4, pp. 67-79.

Specht, D. F., (1991). "A General Regression Neural Network," *I&E Transactions on Neural Networks*, 2(6), 568-576.

Sridhar, D. V., (1996). "Process Modeling Using Stacked Neural Networks," *Ph.D. Dissertation*, Iowa State University, Ames, Iowa.

Upadhyaya, B. R., and E. Eryurek, (1992). "Application of neural networks for sensor validation and plant monitoring," *Nuclear Technology* 97, 170-176.

Uhrig, R. E. (1989). "Use of neural networks in nuclear power plant diagnostics," *Proc. Int. Conf on Availability Improvements in Nuclear Power Plant,* 310-315 Madrid, Spain.

Vapnik, V. (1979). *Estimation of Dependences Based on Empirical Data.* Nauka, Moscow.

Vapnik, V. (1998). *Statistical Learning Theory.* Springer.

Venkatasubramanian, V., and K. Chan, (1989). "A neural network methodology for process methods," *Journal of Applied Meteorology* 31, 405-420.

Watson, G. S. (1964). "Smooth Regression Analysis," *Sankhya – The Indian Journal of Statistics 26,* 359-372.

Weigend, A. S., D. E. Rumelhart, and B. A. Huberman (1991). "Generalization by weight-elimination with application to forecasting," in *Advances in Neural Information Processing Systems 3*, R. Lippmann, J. Moody, and D. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, pp. 875-882.

Werbos, P. (1974). "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D Thesis, Harvard University, Cambridge, MA.

CHAPTER 2. IN SEARCH OF A NEURAL NETWORK CROSS VALIDATION DRIVEN REGULARIZATION

A paper prepared for submission to the Journal of Neural Networks

Craig G. Carmichael and Eric B. Bartlett

Department of Electrical and Computer Engineering,

Iowa State University, Ames, IA, 50011

Abstract

The Scaled Conjugate Gradient algorithm (SCG) [Moller, 1993] is implemented for its potential in supervised learning. An importance calculation is introduced as a new estimate for determining the overall importance of each weight, node, and input in an ANN. Two new complexity-regularization methods are implemented which use this importance calculation. The first, a dynamic node architecture heuristic with feedback (DNAF), adapts the network architecture by adding and cutting nodes. The second, called the Noise Feedback Descent algorithm (NFD), biases the conjugate gradient direction so that unimportant weights decay in a very small number of iterations. Unlike other learning algorithms using complexity penalization, NFD does not require the use of a modified cost function, and it does not seem to increase the training time. The performance of DNAF and NFD are benchmarked against SCG.

Keywords

Artificial Neural Network, Complexity Regularization, Conjugate Gradients, Feedforward Networks, Data Modeling

Introduction

The Scaled Conjugate Gradient training algorithm (SCG) is an extremely fast training method for the supervised learning of artificial neural networks (ANN). The problem is, however, that SCG by itself doesn't reach its full generalization potential because it will tend to overfit the data unless some adaptive constraint is placed on the number of free parameters in the network. Other complexity regularization methods exist for just this purpose, but most tend to be relatively slow in comparison to the already slow standard of backpropagation. Therefore, the purpose here is to derive methods for constraining SCG to areas of weight space that are most likely to produce good ANN models. Both constructive and destructive structural learning approaches are used here. The goal is to fit the data well in terms of generalization and speed.

ANN importance estimates

This section introduces a useful calculation to artificial neural network techniques. That is, it provides a new estimate for how important each weight and node is in the network. With this estimate, two input nodes or weights can be directly compared to each other in terms of their importance. In the same way, unimportant nodes and weights can be targeted for structural learning purposes. The calculation is an extension of simple traditional

methods. Complexity regularization procedures typically assign the importance of connection weight w_{ij} according to some monotonically increasing function of the magnitude of the weight. Here, an observation is made that probably a more accurate estimate involves the addition of another term: the variance of the signal exiting neuron *i* and entering w_{ij}, which connects neuron *i* to neuron *j*. This variance is determined by using the distribution of the signal exiting neuron *i* across all patterns.

Let the static local importance of weight w_{ij} to node *j*, L_{Wij}, be directly proportional to the variance, or standard deviation, of the signal entering node *j*. This assumption in fact must be valid in the limit as the variance approaches zero. In this case, the signal can be replaced by a constant, which means that the weight can be completely eliminated without any loss as long as that constant becomes part of the threshold. So, let the local importance of weight w_{ij} to node *j*, be calculated by the following relation:

$$L_{Wij} = | w_{ij} \sigma[x_i] | \tag{1}$$

where x_i represents the signal exiting node *i*. To obtain the normalized local importance of the weight w_{ij} to node *j*, $\overline{L}_{Wij}$, it must be compared to the importance of other weights fanning into node *j*. Let the normalized local importance of weight w_{ij} to node *j* be normalized according to the following equation:

$$\overline{L}_{Wij} = \frac{L_{Wij}}{\sum_{k=1}^{n} L_{Wkj}}$$

(2)

where n is the number of weights fanning into node j. Now that each weight's local importance has been calculated, the global importance of each node and weight in the network can be estimated. The reason that local and global importance estimates differ is because the information transmitted through a weight must at least reach an output node for that weight to have any chance of being important.

A simple example can demonstrate this effect. Suppose that the normalized local importance of connection weight w_{ij} is large, due to a relatively large weight magnitude $|w_{ij}|$, a large signal variance $\sigma^2[x_i]$, or both. This means absolutely nothing if, say every weight fanning out of node j is zero. No information transmitted across w_{ij} can possibly reach the output node, so the value of $\overline{L}_{Wij}$ is meaningless.

For the importance estimate to be complete, the level of information successfully transmitted across the network from each weight w_{ij} must be accounted for. The calculations above dealt with how important a weight was to the node above it, and they were made possible by a forward recall on the data at the current point in weight space. The calculations below estimate how important each weight and node is to the network. They are made possible by a backward propagation of local importance calculations through the network using the previously calculated values for $\overline{L}_W$.

To begin the global importance estimates, we distribute the global importance, G_j^l, across the output layer l for each output j. Assuming each output is equally important, the following relation holds at the output layer:

$$G_j^l = \frac{1}{O} \tag{3}$$

where O is the number of outputs. Relatively speaking, each weight connected to an output node is only as important as the node it's connected to. Because of this dependency, we let the global importance of each weight, $G_{Wij} = G_j^l \overline{L}_{Wij}$, be determined by the following:

$$G_{Wij} = G_j^l \overline{L}_{Wij} \tag{4}$$

Then, working backwards from the output layer, we calculate the global importance for each node i in the previous layer, $l-1$, the same way. Once again the importance of the nodes below can only be as important as the weights connected to them from above. So, calculate the global importance of node i in layer $l-1$, G_i^{l-1}, according to the following relation:

$$G_i^{l-1} = \sum_{j=1}^{m} G_{Wij} \tag{5}$$

where m is the number of connections above node i. Proceed backwards from the output layer to the input layer until all global calculations have been completed. At this point, the global importance of each weight and node in the network has been estimated. Due to the normalization process mentioned above, the global importance of all of the inputs sum to unity.

It is important to note, however, that the importance of a node or weight is bounded by how well the network is generalizing. Therefore, a more robust definition of importance should be some function of G_{Wij}, G_j^l, and the cross-validation RMS, RMS_{cv}, corresponding to the point in weight space where the importance estimates were calculated. Therefore, let the following be one useful definition of importance:

$$I_j^l = \frac{G_j^l}{RMS_{cv}} \tag{6}$$

$$I_{Wij} = \frac{G_{Wij}}{RMS_{cv}} \tag{7}$$

where I is the overall importance estimate. This definition is best applied to input nodes where there is a basis for comparison between training iterations.

A dynamic node architecture heuristic with feedback (DNAF)

The following three subsections describe the DNAF process. First, select an initial network architecture and randomize all free parameters. Then run through the *DNAF* heuristic module. This calls both the *TRAIN* and *STOP* heuristic modules. It also invokes the importance calculation described above.

The memorization set mentioned below is the data used for gradient descent. The cross-validation or feedback set, is also part of the learning process. The combination of both sets is the learning set.

The DNAF process iteratively updates a memorization threshold, θ_{DNAF}. This corresponds to the amount of memorization allowed before DNAF rips out the most unimportant node. The hope is that by eliminating the network's ability to memorize when θ_{DNAF} is exceeded, the network will generalize better.

DNAF heuristic module

This is the dynamic node architecture heuristic with feedback module. This heuristic attempts to optimize the network architecture by adding and cutting nodes based on feedback from the cross-validation set. Proceed through the following steps without allowing DNAF to step outside the bounds of the minimum and maximum sizes of the network.

1. Set $\theta_{DNAF} = 9999$.
2. Run through *TRAIN* heuristic module (defined below).
3. Check the DNAF stopping criterion by running the *STOP* heuristic module (defined below).
4. If N_{cut} has been set to 1, cut the least important node according to the importance calculation described above. If it hasn't been set to 1, randomly select a layer and add a node to that layer.
5. Set $\theta_{DNAF} = \mid RMS_{cv,good} - RMS_{mem} \mid$. The values for RMS are calculated in the *TRAIN* heuristic module.
6. Go to step 2.

TRAIN heuristic module

In the following heuristic for training a network on a given architecture using the Scaled-Conjugate Gradient algorithm, SCG, the total number of iterations must be greater than the number of free parameters in the network before training is completed. If not, the network must be reinitialized and trained again. This is so training will be less likely to terminate at a bad point in weight space.

1. Set nodal cut flag, N_{cut} and iteration number to 0. The variable N_{cut} will be used by the *DNAF* heuristic module to decide whether to add or cut a node from the current architecture.
2. Increase the iteration number by one. Attempt to reduce the memorization set RMS, RMS_{mem} by applying the Scaled-Conjugate Gradient algorithm, SCG to the memorization set during this iteration.
3. Calculate the cross-validation RMS, RMS_{cv}. If it is smaller than the smallest cross-validation RMS seen while training on the current architecture, RMS_{cv}^{good}, set $RMS_{cv}^{good} = RMS_{cv}$. If it is smaller than the smallest cross-validation RMS seen so far, RMS_{cv}^{best}, then set the following variables:

$$RMS_{cv}^{best} = RMS_{cv}$$
$$H_{best} = H$$

where H is the current total number of hidden nodes.

4. Check the following conditions:

$$RMS_{cv} > RMS_{mem} + 1.5\theta_{DNAF}$$
$$RMS_{mem} < RMS_{cv}^{best}$$

If they both occur, the network is considered to be memorizing the data. In this case, exit training early and set $N_{cut} = 1$. Exit this *TRAIN* heuristic module.

5. Check if the average gradient is less than 0.00001 or if the progress in RMS_{mem} between successful iterations is less than 0.0000001. If either happens, then SCG is no longer making progress in reducing the memorization set RMS. So go to step 6. Otherwise, continue training by going to step 2.

6. Check the following conditions:

$$C1: \quad H > H_{best} + 2$$
$$C2: \quad RMS_{cv}^{good} > RMS_{cv}^{good,last}$$
$$C3: \quad RMS_{cv}^{good} > RMS_{mem} + \theta_{DNAF}$$
$$C4: \quad RMS_{mem} < RMS_{cv}^{best}$$

where $RMS_{cv}^{good,last}$ is last value of RMS_{cv}^{good} if the last DNAF action was to add a node. Otherwise it is a large number like 99999. If C1 or C2 is true, set $N_{cut} = 1$. This means that there has been no indication that the cross-validation RMS has improved by adding nodes. Therefore, cut a node. Also, if C3 and C4 are both true, set $N_{cut} = 1$. This means that the network has memorized an unacceptable amount of noise. Exit this *TRAIN* heuristic module.

STOP heuristic module

This module checks the DNAF stopping criterion and the result is passed back to the *DNAF* module.

1. Check if the cross-validation RMS hasn't seen any improvement for the last 25 DNAF iterations or if the best architecture (corresponding to RMS_{cv}^{best}) has been swept across three times without improvement.
2. If neither one of these occur, the DNAF stopping criterion hasn't yet occurred, so exit this *STOP* heuristic module.
3. Otherwise, the DNAF stopping criterion has occurred. Set the network architecture to the best architecture seen and randomize all parameters. Run through *TRAIN* heuristic module. Do this step five times, and then Exit this *STOP* heuristic module.

Noise feedback descent algorithm

One of NFD's novel features includes the use of a feedback set to give an estimate of the level of noise that has been memorized. Unlike a typical training algorithm, NFD uses this estimate of noise as feedback to the training algorithm in a way that tends not to memorize the feedback set. Also, unlike a typical training algorithm with complexity penalization, NFD biases the gradient direction towards noiseless areas of weight space instead of relying on a modified cost function with arbitrarily set constants to reduce the number of parameters in the ANN. In addition, NFD estimates the importance of each input during a given iteration using only the data and the current point in weight space. This estimate can be very useful for future data collection. It can also be useful as a future means of biasing the gradient direction so that the most unimportant inputs are completely eliminated first. This would be a more direct approach to online backwards elimination.

Feedback set

NFD's feedback set is very similar to a standard cross-validation set. However, the feedback set provides an estimate of the current noise level that will be fed back to the training algorithm. This allows for a simultaneous optimization of training set RMS and feedback set RMS. Used in conjunction with the following training algorithm, the feedback set does not provide a means for memorizing the data. Its use forces the training algorithm to search areas of weight space that exhibit good generalization characteristics.

Training

The Noise Feedback Descent algorithm, NFD, is derived from:

- The Scaled Conjugate Gradient algorithm described above. For purposes of clarity, some of SCG's most important features are highlighted below:

1. Initialize $\tilde{w}_1$, $\tilde{p}_1$, $\tilde{r}_1$, *scalars*
2. $\tilde{s}_k = \dfrac{E'(\tilde{w}_k + \sigma_k \tilde{p}_k) - E'(\tilde{w}_k)}{\sigma_k}$
3. $\alpha_k = \dfrac{\tilde{p}_k^T \tilde{r}_k}{\tilde{p}_k^T \tilde{s}_k}$
4. $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k$
5. $\tilde{p}_{k+1} = f(\tilde{r}_{k+1}, \tilde{r}_k, \tilde{p}_k, etc.)$
6. $k = k + 1$
7. Go to 2.

- Structural learning with complexity penalization. The idea here is to reduce the number of trainable parameters in order to hopefully improve generalization. A typical solution formulation involves a modified cost function with arbitrarily set constants as shown below:

$$E_s = E + \varepsilon \sum_{|w_{ij}|<N} | w_{ij} |$$

(8)

The Noise Feedback Descent algorithm inserts a crucial step into SCG between steps 3 and 4 shown above if, and only if, the following condition occurs:

$$E < E_{fb}^{best}$$

(9)

where E_{fb}^{best} is the lowest batch error seen in the feedback set during training. If this condition holds, NFD utilizes information calculated during the current iteration to bias the gradient direction towards a nearly equivalent, noiseless point in weight space as shown below:

1. $\alpha_k = \dfrac{\tilde{p}_k^T \tilde{r}_k}{\tilde{p}_k^T \tilde{s}_k}$
2. $\tilde{p}_k = f(\tilde{p}_k, \alpha_k, \tilde{r}_k, \tilde{s}_k, \tilde{w}_k, \tilde{X} \rightarrow E_k, \tilde{X}_{fb} \rightarrow E_{fb,k})$
3. $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k$

An extremely useful piece of information is available from a second-order training method such as SCG, useful in a way that is well suited for structural learning. That is, what is the estimated step size α_k from $\tilde{w}_k$ along $\tilde{p}_k$ needed to reach a minimum in RMS. Derived below is a method of using α_k in a way that selectively drives certain synaptic weights towards zero in a small number of iterations. It does so without sacrificing a significant reduction in training set RMS and does not necessarily slow down the training algorithm.

From the SCG algorithm described above, when a weight is updated, it is updated according to the following equation:

$$w_{k+1}^{ij} = w_k^{ij} + \alpha_k \, p_k^{ij} \tag{10}$$

where w_k^{ij} is the connection from neuron i in the last layer to neuron j in the current layer during the current iteration k. Before this step is encountered, however, it is important for the training algorithm to realize that as the training set batch error, E, becomes smaller than the feedback set batch error, E_{fb}, strictly going downhill in E usually results in going uphill in E_{fb}. Knowing when this happens and to what degree can be extremely useful in knowing when not to strictly follow the conjugate gradient direction $\tilde{p}$.

With this in mind, NFD decides to update $\tilde{w}$ by stepping a distance α along $\tilde{p}$ only when E and E_{fb} agree to some extent. This is acceptable because the data in the feedback set is suggesting that the network is generalizing. However, as E_{fb} starts

becoming larger than E, corrective measures are taken to eliminate the network's ability to memorize. This is accomplished by stepping a distance α from $\tilde{w}$ along a biased direction $\tilde{p}_B$ designed to eliminate noise without drastically affecting the reduction in training set RMS. Noise elimination is attempted through weight decay.

So, after α_k has been calculated, each connection weight w_k^{ij} can be driven exactly to zero in just one iteration by stepping a distance α_k from $\tilde{w}_k$ along an absolute pruning direction $\tilde{p}_{A,k}$, which is derived as follows:

$$w_{k+1}^{ij} = 0 = w_k^{ij} + \alpha_k \, p_{A,k}^{ij} \tag{11}$$

$$p_{A,k}^{ij} = -\frac{w_k^{ij}}{\alpha_k} \tag{12}$$

where $p_{A,k}^{ij}$ is the absolute pruning direction component for the weight connecting node *i* in the previous layer to node *j* in the current layer. However, updating along this direction for all weights would obviously result in very poor training performance. Also, it would be ideal for α_k to still be approximately the step size that corresponds to a minimum in the training set RMS along the updated direction, but radically biasing the gradient direction toward $\tilde{p}_{A,k}$ may radically reduce the accuracy of this step size estimate.

The goal, therefore, is to bias the gradient direction so that α_k remains a relatively good estimate and the reduction in training set RMS remains relatively unaffected while

driving certain weights towards zero along a biased gradient direction, $\tilde{p}_{B,k}$. For each weight, the biased direction is an interpolation of $\tilde{p}_k$ and $\tilde{p}_{A,k}$ determined by:

$$p^{ij}_{B,k} = p^{ij}_k + \Gamma^{ij}_k (p^{ij}_{A,k} - p^{ij}_k) \tag{13}$$

where Γ^{ij}_k is some parameter in the range [0,1]. As Γ^{ij}_k approaches zero, the biased direction becomes the conjugate gradient direction. Alternately, as it approaches unity, the biased direction becomes the absolute pruning direction. In other words, this parameter is directly proportional to how much the corresponding weight will be pruned. The question then is, "How is Γ chosen for each connection weight?"

Let I^{ij}_k be some measure of the importance of the signal transmitted through weight *ij* at iteration k, w^{ij}_k, through node *j*. If I^{ij}_k is relatively large in comparison to the Γs corresponding to the other weights fanning into neuron *j*, the signal is considered more important to neuron *j* at iteration *k* and $p^{ij}_{B,k}$ should be close to the unbiased conjugate direction component p^{ij}_k, so Γ^{ij}_k should be close to zero. On the other hand, if I^{ij}_k is relatively small, Γ^{ij}_k should be close to 1 and weight w^{ij}_k should be driven close to zero by iteration $k+1$ because it is not deemed very important at this time. For now, let I^{ij}_k be defined as follows:

$$I^{ij}_k = | w^{ij}_k \sigma_k [x^i] r^{ij}_k s^{ij}_k | \tag{14}$$

where x^i is the signal output of neuron i, $\sigma_k[x^i]$ is the square root of the signal variance of neuron i's output across all patterns during iteration k, r_k^{ij} is a first-order calculation of the sensitivity of E to a change in weight w_k^{ij}, and s_k^{ij} is a second-order approximation of the same sensitivity. The reasoning behind the choice for I_k^{ij} has to do with the intrinsic nature of the network's learning process. As training progresses, a broader range of each neuron's non-linear region is used. This means that in general a neuron's internal activity level and output signal have higher variances at the end of the training process than at the beginning. It is reasonable to assume then that connection weights with higher signal variances are in general more important to a node than connections with lower signal variances. Also, each neuron contains a bias constant that is assigned the job of controlling a zero-variance signal. During the error backpropagation phase of training, a gradient algorithm has difficulty distinguishing between a low-variance signal through a synaptic weight and the zero-variance signal controlled by the bias. The choice for I_k^{ij} is meant to slightly modify the credit-assignment task of simple backpropagation so that more credit for the error is given to the bias and less is given to synaptic weights with low-variance signals, thereby forcing the ANN towards less noisy points in weight space. The r_k^{ij} and s_k^{ij} terms allow weights to have a chance of being important in the future. Otherwise, we'd be dealing with a static picture of I_k^{ij} throughout training.

Now that a quantitative measure of importance for each weight at iteration k has been established, it's time to transform the relative importance of a weight into the Γ parameter with range [0,1] so that $p_{B,k}^{ij}$ can finally be calculated. A reasonable choice is the following

set of equations, which results in a calculation for Γ_k^{ij}, where iteration k has been dropped for simplicity:

$$H_l = \frac{n}{\sum_{i=1}^{n} \frac{1}{I^{ij}}} \tag{15}$$

$$R^{ij} = \left(\frac{4I^{ij}}{H_l} \right)^2 \tag{16}$$

$$\gamma^{ij} = \frac{1}{1 + R^{ij}} \tag{17}$$

$$\Gamma^{ij} = A\gamma^{ij} \tag{18}$$

where H_l is the hyper-geometric mean of the I^{ij}'s across all non-zero weights fanning into node j during iteration k, R^{ij} is the relative importance of w^{ij} to node j, γ^{ij} is a suggested value for Γ^{ij}, and A scales the values for all Γ's in the network. Notice that as the importance of a weight approaches zero the calculation for γ^{ij} approaches unity, and as the importance of a weight approaches infinity the calculation for γ^{ij} approaches zero. The value for A is chosen so that both E and E_{fb} approximately meet at some point in between (E and E_{fb}) in one iteration. Solving for the following equation yields a reasonable choice for A:

$$E_{fb} = E + 2\sum_{ij}^{Nij's}\left(r^{ij}\alpha A\gamma^{ij}(p_A^{ij} - p^{ij})\right)$$

(19)

where $Nij's$ is the number of synaptic weights in the network. Now that all the elements in $\tilde{p}_B$ can be calculated for each synaptic weight, the credit-assignment task must be completed. The bias at each neuron j must be additionally assigned the sum of the signal averages taken from the synaptic weights fanning into neuron j. This can be calculated as:

$$p_{B,k}^{0j} = p_k^{0j} + \sum_{i=1}^{n}(p_{B,k}^{ij} - p_k^{ij})\bar{x}^i$$

(20)

where n is the number of weights fanning into neuron j and $\bar{x}^i$ is the average output signal from neuron i in the last layer across all patterns. Using this biasing method, the weight vector $\tilde{w}_k$ is driven to a nearly-equivalent point in weight space containing less noise than that achieved by traveling strictly along $\tilde{p}_k$. With this final equation, the vector $\tilde{p}_B^k$ is complete. So, the following equality completes the biasing process:

$$\tilde{p}_k = \tilde{p}_{B,k}$$

(21)

Test Problems

The following example problems were chosen to test the learning speed and generalization potential of the methods in this paper. The first example is a difficult problem in memorization. Back-propagation methods typically require a large amount of training time just to memorize it. Some variations don't even converge for most architectures. The remaining three examples were chosen to test for generalization as well as speed and memorization. Problems 2 and 3 are described as "impact" problems here, but in the following chapters the underlying mathematical system generates benchmark problems that are redefined as "spooky particle data sets." The fourth is a real world time series problem.

Example 1: Spiral problem

In this example, the well-known spiral problem is used to compare the performance of the scaled conjugate gradient algorithm with various forms of backpropagation. The goal is to show why SCG is chosen instead of BP as the underlying basis for the experiments described in this chapter. This data set is shown in Figure 1, which contains 194 training patterns, 194 cross-validation patterns, and 194 test patterns. Each pattern is comprised of inputs $\{X,Y\}$ and output C, where C classifies which spiral the pattern belongs to by a binary number.

Example 2: Two-dimensional rectangular impact problem

A robot is placed in the middle of a dark, empty room. Its task is to gather information about the room and learn how to avoid running into the padded walls. First, a position measurement is sampled and stored as input variables X and Y. Then, an arbitrary

direction and initial velocity are chosen and stored as input variables V_{x0} and V_{y0}. The robot then travels along that direction at that speed and then coasts until it impacts the wall. At that instant, the robot calculates the elapsed time from its initial position to its point of impact. This result is then stored as output variable T_i. The robot's objective is to accurately model T_i given the fewest measurements possible.

This impact problem is really a thought experiment in generalization. Since there are only a small number of inputs and the system is completely deterministic, sampling a huge number of measurements would require only memorization to generalize. However, the objective is to generalize with a relatively small number of measurements. To learn this problem, an ANN has to indirectly extract the physics of the room from the data. How big is the room and what wall configuration surrounds it? Is the floor slick and flat or carpeted and sloped? Although these questions won't be answered directly by the ANN, the output of importance depends on them.

In this first impact problem, a rectangular room is chosen with dimensions 10' by 20'. For each pattern, the inputs X and Y are assigned a uniformly random position within the bounds of the room. The sampling of this 2D initial positioning is shown in Figure 2. An initial velocity V_0 is randomly chosen within the range [3,5] feet per second and an arbitrary direction is assigned to break V_0 into a vector having X and Y components. An acceleration constant of -0.2 ft/s^2 is selected for each trajectory. Also, a random variable R is introduced into the data as another input.

Each pattern is described by inputs $\{X, Y, V_{x0}, V_{y0}, R\}$ and output T_i. One hundred patterns are generated here for the training set and fifty for the test set.

Example 3: Three dimensional ellipsoidal impact problem

In this next example, the impact problem is extended into three-dimensions. Here, the system is bounded by a hollow ellipsoid defined by axis dimensions $A_x = 1$, $A_y = 2$, and $A_z = 3$. For each pattern, the inputs *X*, *Y*, and *Z* are assigned in the same way as above. An initial velocity of $V_0 = 1$ is set and broken into a vector having *X*, *Y*, and *Z* components by an arbitrarily chosen direction. No acceleration takes place.

Each pattern here is described by inputs $\{X, Y, Z, V_{x0}, V_{y0}, V_{z0}\}$ and output T_i. The sampling of this 3D initial positioning is shown in Figure 3. Two hundred patterns are generated in this case for the training set and fifty for the test set.

Example 4: Electric futures problem

An electricity futures contract is a legal agreement that binds two parties in the future sale of electricity. The agreement is for an agreed upon delivery price, time, and location. Contracts are traded at the New York Mercantile Exchange (NYMEX). Currently, two electricity futures contracts exist, one for Palo Verde, Arizona and the other for the California/Oregon border (COB). In this example, the objective was to predict the next month's futures price for one day ahead given the date, futures price statistics, etc. The ANN models here attempted to predict the change in price from today to tomorrow. Only 120 patterns were available at the time, so this problem was expected to be extremely difficult for an ANN to learn. Figure 4 shows this time series.

Results

A hyperbolic tangent activation function was chosen for the Scaled Conjugate Gradient algorithm. Also, SCG was slightly modified so that it checked the cross-validation set RMS each iteration and saved the network corresponding to the best value seen. All references in this paper to SCG refer to SCG with TANH activation functions and the save-best criterion.

The results in this section demonstrate both the training speed of SCG and the generalization performance of the SCG-based complexity-regularization methods in this chapter. The results for example 1 briefly demonstrate why SCG is chosen as the basis for DNAF and NFD. Then, results for examples 2 and 3 show why there is a need to supplement SCG with DNAF and/or NFD.

Results for example 1

A 2x21x14x7x1 network was arbitrarily chosen as the trainable architecture for the two-spiral problem. Various forms of backpropagation and SCG were compared in terms of learning speed using this neural framework. Carnegie Mellon University's Quickprop freeware and nearly all of the backpropagation methods in the NeuroSolutions package didn't even converge using this architecture on this problem. However, backpropagation with momentum (from NeuroSolutions) did converge, requiring 56 minutes of training time. On the other hand, SCG accomplished the same task in 18.6 seconds. Table I shows the normalized performance (the output's range is [-1,1] due to the TANH activation function) of SCG on the two-spiral problem. The column labeled R^2 refers to the square of the linear correlation coefficient between the actual and desired output across all patterns.

Please refer to Figure 5. This graph shows the model error vs. the pattern number for the learning set. The first 194 patterns represent the memorization set and the next 194 patterns represent the cross-validation set. Notice the difference between the two. Even though the classification percentage is 100% for both sets, the errors across the memorization set are substantially smaller than those across the cross-validation set.

Please refer to Figure 6. This graph shows the model error vs. the pattern number for the test set The errors across the test set are similar in magnitude to those across the cross-validation set, with the exception that one pattern has been classified incorrectly. The focus here was to demonstrate the learning speed of SCG on this problem. The focus on the remaining two example problems is generalization.

Results for examples 2 and 3

In tables II and III, the abbreviation DI refers to dynamic inputs, where the DNAF heuristic allows the addition and subtraction of nodes to occur in the input layer. Also, DFS refers to a dynamic feedback set, where the cross-validation or feedback set is shuffled into the learning set and reselected after each feature modification. This does not instantly memorize the feedback set because each feature alteration is accompanied by a randomization process. The symbol LRN refers to the learning set and TEST refers to the unseen test set.

Cases 1 and 2 represent using SCG to train six different networks having the specified architecture, and then selecting the network that performs the best on the cross-validation set. Cases 7 and 8 represent the same thing with the addition of NFD. Cases 3 through 6 implement the DNAF heuristic, which begins with one single-hidden-layer network having a

number of hidden nodes equal to the number of inputs. Each case was tested 10 times and the results are tabulated in the form of mean plus or minus one standard deviation.

The best generalization performance on average, measured by the results in the unseen test set, was case 7 (SCG+NFD 5x8x8x1). In comparison to SCG, SCG with NFD not only generalized much better on the 5x8x8x1 network, it actually required less than half of the training time of SCG. This is possible because of the difference in stopping criterion between the two. SCG continues until it reaches a minimum in the memorization set RMS. SCG with NFD continues until no progress has been seen in the feedback set for 1000 iterations.

The DNAF heuristic also performed well in terms of generalization. The primary reason that case 3 outperformed cases 4 through 6 was because none of its 10 runs resulted in extremely poor cross-validation performance. The other cases weren't quite as lucky, as you can see by comparing the standard deviations for these DNAF cases in the R^2_{TEST} column. Filtering out the poor cross-validation results would have resulted in higher means for cases 4 through 6. But overall, DNAF proved quite effective.

Cases 2 and 8 were implemented on a 5x6x1 architecture, known after using DNAF as approximately the optimal architecture. Because of this, there is very little difference between SCG and SCG with NFD when the bounded architecture is close to optimal. This coincides with the expectation that NFD shouldn't do much to an already efficient architecture. Once again, NFD doesn't reduce the speed of SCG, which is contrary to traditional structural learning approaches. It only increases its potential to generalize. The results in Table III show similar performance.

Figures 7 and 8 graphically display the generalization performance for each case. While NFD doesn't seem to drastically affect the performance on a nearly optimal bounded architecture, it does significantly improve generalization on a substantially larger bounded architecture. Both DNAF and NFD allow the user the freedom of selecting an initial architecture that is not optimal.

Results for example 4

Tables IV and V show the results for the futures problem. The three ANN learning methods in this thesis were applied with various cross validation partitions. Despite promising results in the memorization (training) sets and even in the cross-validation sets, performance in most of the true test sets was poor. For Palo-Verde futures, a test set RMS of 3.31 corresponds to guessing zero for the change in price every day. For COB futures, a test set RMS of 1.03 was the number to beat. Notice that NFD achieved the best performance for both Palo-Verde and COB futures.

Conclusions

The Scaled Conjugate Gradient algorithm is an extremely fast and robust ANN method for supervised learning. With the addition of saving the network that performed the best on the cross-validation set, the algorithm can also generalize well. However, a technique for optimizing the network architecture is still necessary for any training algorithm to reach its full potential in generalization. Other complexity-regularization methods exist for improving generalization in this way, but most are relatively slow.

Two new complexity-regularization methods were compared to SCG with cross-validation optimization. Both methods relied on checking cross-validation performance from iteration to iteration. This provided feedback for the algorithms to decide how to prevent over fitting of the data. The first method, a dynamic node architecture heuristic, added or cut a feature (or node) based on this feedback. The second, called the Noise Feedback Descent algorithm, biased the conjugate gradient direction to prevent over fitting. Also, both methods relied on a new importance calculation to eliminate unimportant features and weights.

The Scaled Conjugate Gradient algorithm was compared to various forms of backpropagation on the two-spiral problem, chosen mainly because of how difficult it is for an ANN to learn. Since both algorithms attempt to minimize the error in the memorization set, only the speed and convergence were compared between SCG and backpropagation on a 2x21x14x7x1 architecture. A modified backpropagation algorithm, called Quickprop, was downloaded from Carnegie Mellon University's anonymous FTP site. Also, backpropagation solutions from the commercial package NeuroSolutions were tested. The only backpropagation method that converged at all was backpropagation with momentum from NeuroSolutions. It required 56 minutes of training time to reach a respectable RMS in the training set. On the other hand, SCG obtained the same results in 18.6 seconds. Although the speed difference may not be as significant on easier problems, SCG will almost always be much faster than backpropagation.

Choosing SCG as the starting point, attempts were made to improve SCG in terms of generalization. A new benchmark for ANNs, called the impact problem, was created to test the generalization potential of the SCG-based methods in this chapter. On average, NFD

performed better than the dynamic node architecture heuristic, DNAF, which performed better than SCG on an arbitrary static architecture.

References

Bartlett, E. B. (1994). "Dynamic Node Architecture Learning: An Information Theoretic Approach", *Neural Networks 7*, 129-140.

Bartlett, E.B. (1994). A Dynamic Node Architecture Scheme for Layered Neural Networks. *Journal of Artificial Neural Networks*, *1*, 229-245.

Blum, E. K. and L. K. Li, (1991). "Approximation Theory and Feedforward Networks," *Neural Networks*, 4, 511.

C. G. Carmichael (1997). "Experiments in advancing supervised-learning ANN techniques," Thesis(M.S), Iowa State University.

Cherkassky, S. and F. Mulier, (1998). *Learning From Data: Concepts, Theory, and Methods (Adaptive and Learning Systems for Signal Processing, Communications and Control Series).* John Wiley & Sons, New York, New York.

Cun, Y. Le, J. S. Denker, and S. A. Solla (1990). "Optimal brain damage," Advances in Neural Information Processing Systems 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 598-605.

Haykin, S., (1999). *Neural Networks: A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, New Jersey.

Ishikawa, M. (1995). "Structural Learning with Forgetting", *Neural Networks 9*, 509-521.

Mackey, M. C. and L. Glass (1977). "Oscillations and Chaos in Physiological Control Systems," *Science*, 197, 287-289.

Mandilwar, D. K. and H. Qammar (1993). "Prediction of Chaotic Time Series: Neural Versus Fuzzy", *Proceedings of the 1993 International Fuzzy Systems and Intelligent Control Conference,* 189-199.

Miller, W. T., R. S. Sutton, and P. Werbos (Eds.), (1990). *Neural Networks for Control.* Press. Cambridge, Mass.

Mitchell, T., (1997). *Machine Learning.* McGraw-Hill, New York, New York.

Moller, M.F. (1993). "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", *Neural Networks 6,* 525-533.

M. C. Mozer and P. Smolensky (1989). "Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in Neural Information Processing Systems, 1, D. S. Touretzky , Ed. (Denver 1988), pp. 107-115.

Russell, S. and P. Norvig, (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey.

Schmitz, G. P. J. and C. Aldrich (1999). "Combinatorial Evolution of Regression nodes in Feedforward Neural Networks," *Neural Networks,* 12, 175-189.

Wichmann, N. L. and E. B. Bartlett (1997), "Ranking Input Variables using General Regression Neural Networks", *Intelligent Engineering Systems Through Artificial Neural Networks: Volume 7,* pp. 959 – 964.

Wichmann, N. L., (1997). "Variable Importance Ordering for Evapotranspiration using General Regression Neural Networks", *M.S. Thesis, Iowa State University, Ames Iowa.*

Wolpert, D. H., (1990). "A Mathematical Theory of Generalization: Part I and Part II," *Complex Systems,* 4, 151.

Tables and figures

Table I. SCG overall performance on the spiral problem

Set	RMS	R^2	Classification %
Learning	0.008964	0.999929	100
Test	0.084062	0.992972	99.485

Table II. Method performance on rectangular impact problem

Case	Description	RMS_{LRN}	R^2_{LRN}	RMS_{TEST}	R^2_{TEST}	TIME
1	SCG 5x8x8x1	0.3354 ± 0.0632	0.9321 ± 0.0276	0.9257 ± 0.2430	0.5946 ± 0.1029	277.6 ± 28.2
2	SCG 5x6x1	0.3456 ± 0.0335	0.9295 ± 0.0129	0.6244 ± 0.0433	0.7711 ± 0.0359	30.3 ± 4.4
3	SCG+DNAF	0.4214 ± 0.0826	0.8916 ± 0.0491	0.5594 ± 0.1250	0.8016 ± 0.0924	71.2 ± 36.7
4	SCG+DNAF+DI	0.4117 ± 0.1522	0.8870 ± 0.1102	0.6043 ± 0.1381	0.7682 ± 0.1282	115.3 ± 55.0
5	SCG+DNAF+DFS	0.4635 ± 0.1390	0.8622 ± 0.0999	0.6399 ± 0.1927	0.7416 ± 0.1561	88.2 ± 71.9
6	SCG+DNAF+DI+DFS	0.4807 ± 0.1304	0.8539 ± 0.0990	0.5988 ± 0.1361	0.7698 ± 0.1186	98.3 ± 46.3
7	SCG+NFD 5x8x8x1	0.3988 ± 0.0719	0.9074 ± 0.0309	0.5288 ± 0.0338	0.8350 ± 0.0162	114.4 ± 15.8
8	SCG+NFD 5x6x1	0.4729 ± 0.0436	0.8686 ± 0.0227	0.5726 ± 0.0670	0.7954 ± 0.0505	30.8 ± 7.0

Table III. Method performance on ellipsoidal impact problem

Case	Description	RMS_{LRN}	R^2_{LRN}	RMS_{TEST}	R^2_{TEST}	TIME
1	SCG 6x8x8x1	0.2046 ± 0.0453	0.9344 ± 0.0278	0.4825 ± 0.1667	0.6843 ± 0.1470	414.2 ± 31.4
2	SCG 6x6x1	0.2233 ± 0.0119	0.9254 ± 0.0080	0.3221 ± 0.0405	0.8444 ± 0.0300	55.4 ± 10.0
3	SCG+DNAF	0.2654 ± 0.0734	0.8865 ± 0.0814	0.3436 ± 0.1096	0.8033 ± 0.1556	139.0 ± 78.4
4	SCG+DNAF+DI	0.2507 ± 0.0575	0.9012 ± 0.0459	0.3612 ± 0.0834	0.7997 ± 0.0807	240.1 ± 128.7
5	SCG+DNAF+DFS	0.2497 ± 0.0313	0.9052 ± 0.0257	0.3150 ± 0.0561	0.8460 ± 0.0495	133.8 ± 72.1
6	SCG+DNAF+DI+DFS	0.2258 ± 0.0271	0.9224 ± 0.0202	0.2949 ± 0.0621	0.8582 ± 0.0516	185.8 ± 91.3
7	SCG+NFD 6x8x8x1	0.2346 ± 0.0243	0.9198 ± 0.0154	0.2444 ± 0.0326	0.8937 ± 0.0335	184.3 ± 33.6
8	SCG+NFD 6x6x1	0.3015 ± 0.0369	0.8669 ± 0.0323	0.3474 ± 0.0935	0.7844 ± 0.1032	56.3 ± 14.7

Table IV. Method performance on Palo-Verde electric futures

CV Type	CV %	Training Method	MEM RMS	MEM R^2	CV RMS	CV R^2	Test RMS	Test R^2
Random	25	SCG	1.428	0.143	1.177	0.029	3.269	0.045
Random	25	DNAF	0.627	0.835	0.738	0.562	4.165	0.001
Random	25	NFD	1.199	0.276	1.752	0.118	3.504	0.039
Random	35	SCG	1.227	0.116	1.733	0.005	3.313	0.001
Random	35	DNAF	1.055	0.346	1.278	0.43	3.839	0.074
Random	35	NFD	1.119	0.763	1.291	0.043	3.504	0.004
Last N	25	SCG	1.411	0.163	1.436	0.014	3.327	0.007
Last N	25	DNAF	1.317	0.271	0.848	0.524	3.531	0.03
Last N	25	NFD	1.551	0.052	1.115	0.067	3.513	0.027
Last N	35	SCG	1.545	0.001	1.746	0.014	3.369	0.053
Last N	35	DNAF	1.057	0.344	1.397	0.312	3.209	0.081
Last N	35	NFD	1.148	0.227	1.51	0.225	3.045	0.195

Table V. Method performance on COB electric futures

CV Type	CV %	Training Method	MEM RMS	MEM R^2	CV RMS	CV R^2	Test RMS	Test R^2
Random	25	SCG	1.036	0.107	0.832	0.076	1.101	0.006
Random	25	DNAF	0.822	0.438	0.653	0.449	1.203	0.004
Random	25	NFD	0.982	0.203	0.992	0.03	1.267	0.079
Random	35	SCG	0.925	0.002	1.251	0.081	1.031	0.012
Random	35	DNAF	0.525	0.65	1.016	0.387	1.312	0.008
Random	35	NFD	0.979	0.182	1.004	0.028	1.246	0.089
Last N	25	SCG	1.039	0.103	0.952	0.002	1.171	0.068
Last N	25	DNAF	1.142	0.013	0.765	0.211	1.159	0.001
Last N	25	NFD	0.971	0.216	0.976	0.001	0.966	0.103
Last N	35	SCG	0.839	0.104	1.257	0.055	1.133	0.049
Last N	35	DNAF	0.911	0.025	1.163	0.255	1.075	0.021
Last N	35	NFD	0.801	0.235	1.255	0.038	0.979	0.095

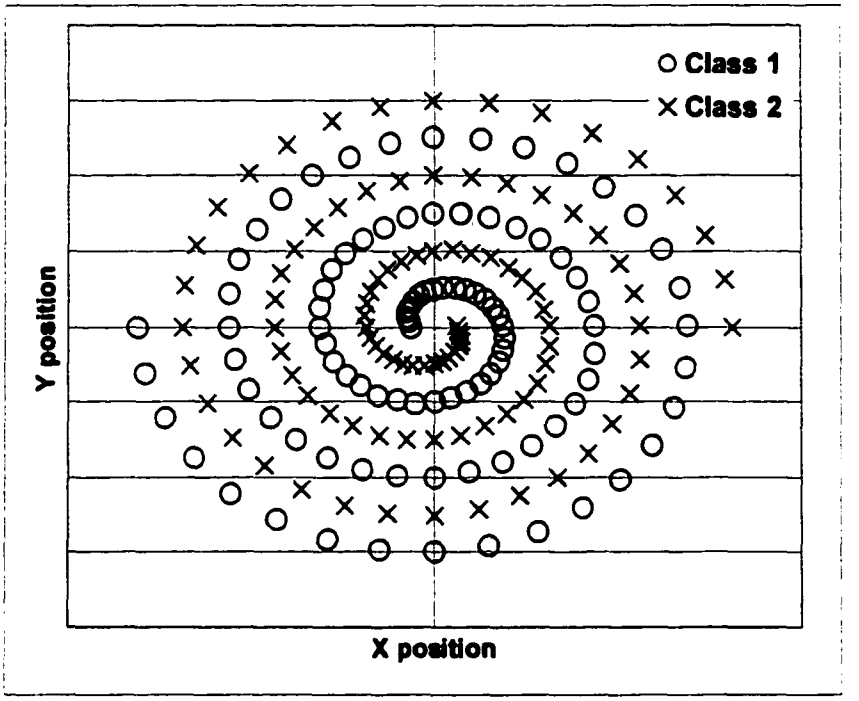


Figure 1. Spiral classification problem

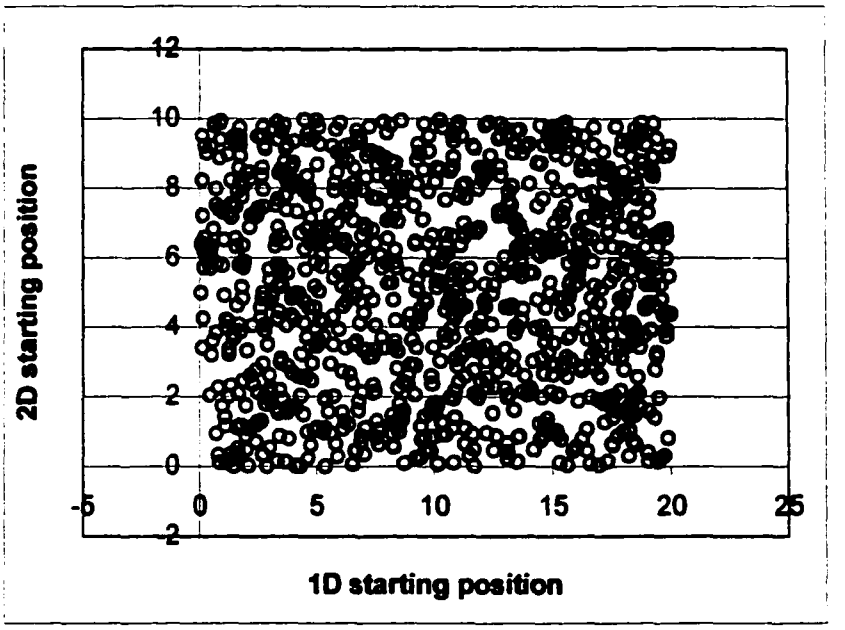


Figure 2. 2D rectangular impact problem (starting positions)

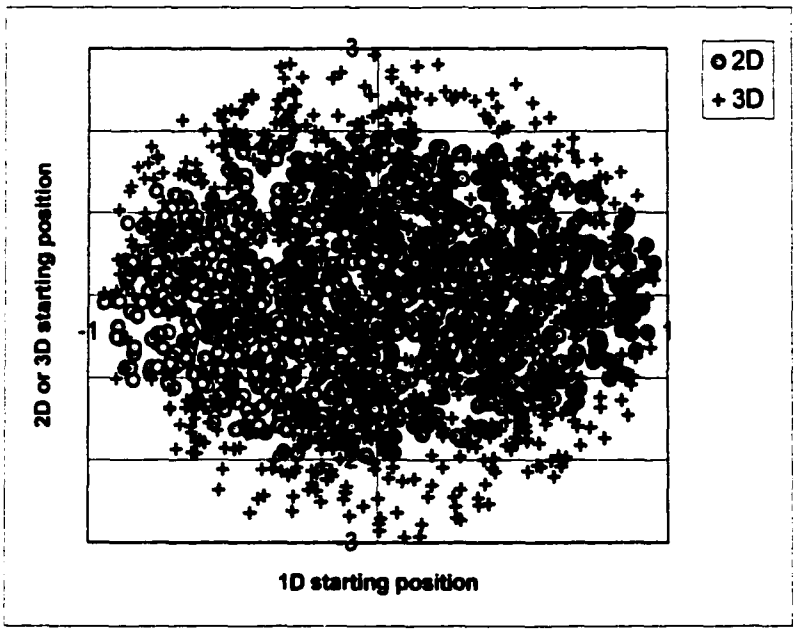


Figure 3. 3D ellipsoidal impact problem (starting positions)

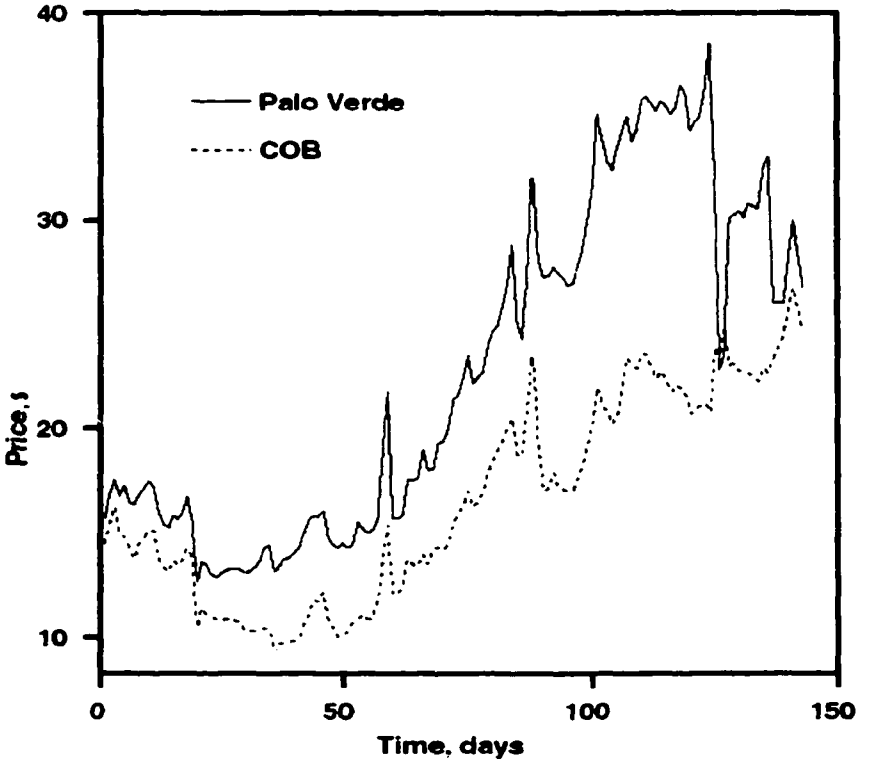


Figure 4. Electric futures time series problem

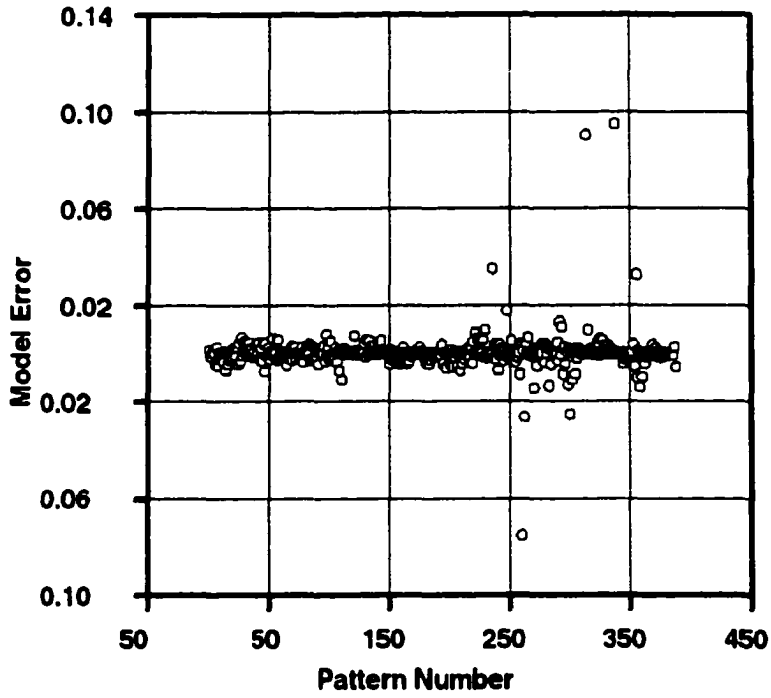


Figure 5. SCG performance on learning set for the spiral problem

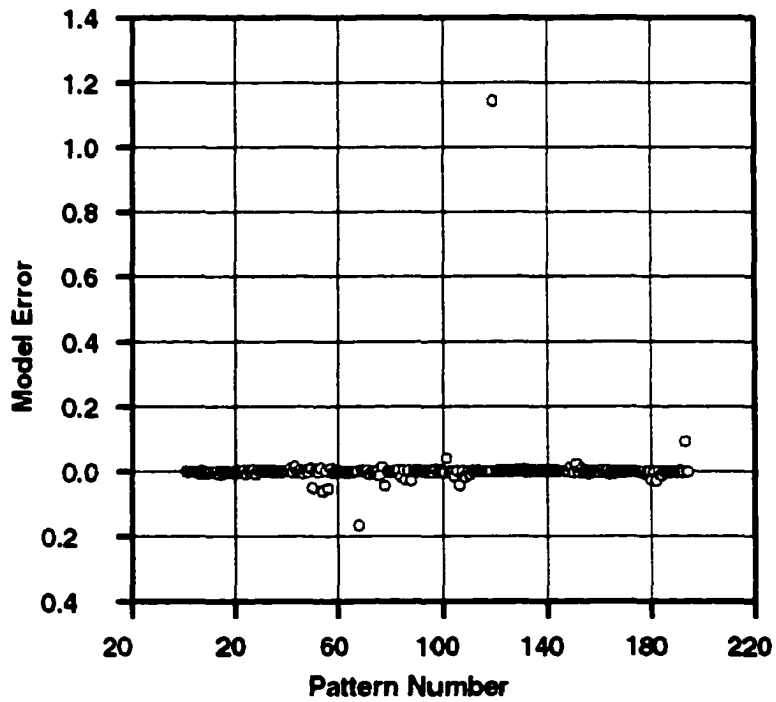


Figure 6. SCG performance on test set for the spiral problem

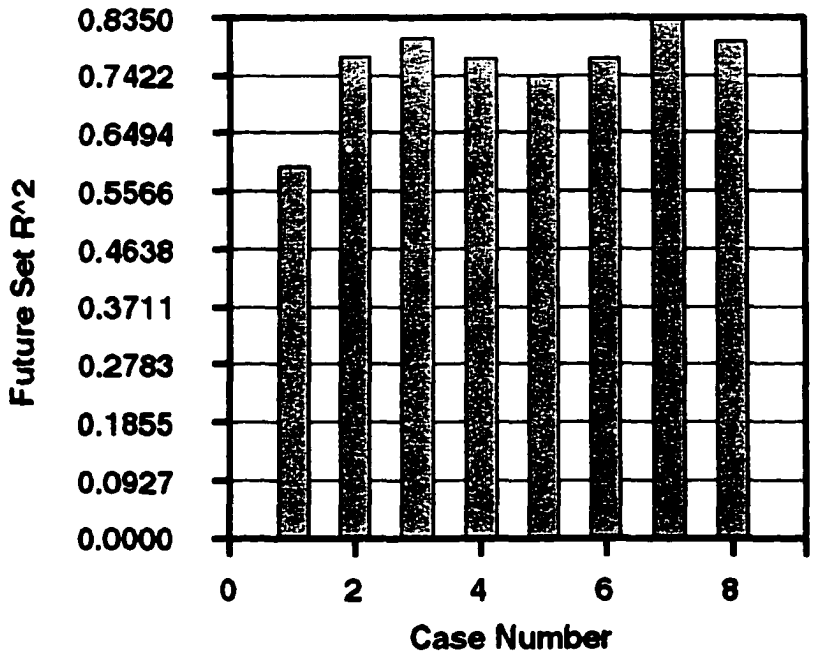


Figure 7. Case generalization on the rectangular impact problem

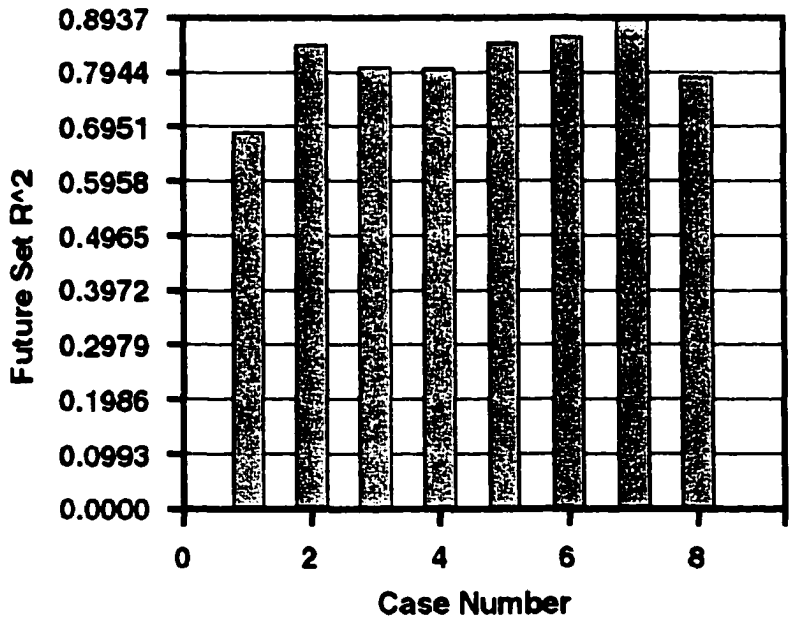


Figure 8. Case generalization on the ellipsoidal impact problem

CHAPTER 3. INSIDE THE BLACK BOX: RECONSTRUCTING A NEURAL NETWORK'S DECISIONS

A paper submitted to the Journal of Machine Learning

Craig G. Carmichael and Eric B. Bartlett

Department of Electrical and Computer Engineering,

Iowa State University, Ames, IA, 50011

Abstract

Artificial Neural Networks (ANNs) have been an intense topic of research in the last decade. In the past, they were viewed as black boxes, where the inputs were known and the outputs were computed, but the underlying statistics and thus reliability of the networks were not fully understood. Because of this, there has been hesitation in utilizing ANNs in automated systems such as intelligent flight control. Now, advances are being made that diminish this hesitation. Individual elements of a neural network can be probed and their decision-making power assessed. Here, a neural network is trained and then various ranking methods are used to assess the importance (saliency or decision-making power, DMP) of each input node. Then, the input data is renormalized according to the DMP input vector and fed to a general regression neural network (GRNN) for training. The accuracy of the DMP ranking methods are then compared against each other from the resulting modified GRNNs. Five ranking methods are tested and compared on four separate data sets, one of which is introduced as a generic data set for future comparison between inductive learning machines.

The resulting analysis demonstrates that saliency estimates based on a sensitivity analysis is the most accurate DMP ranking method for the problems tested here.

Keywords

Artificial Neural Network, General Regression Neural Network, Saliency, Feedforward Networks, Data Modeling

Introduction

Artificial Neural Networks (ANNs) have in the past been described as black boxes. This was often an adequate description, since they were far less understood than they are today. A neural network would be trained on the known data, and then its skills in generalization would be tested on some test set. Depending on the sample size and density of the training data, the reliability of the resulting model would be questioned. Every so often it would generate very inaccurate predictions, and no one knew when to trust it. Therefore, until neural networks could accurately estimate their own confidence in their own predictions, they would always be considered as black boxes to statisticians.

Inductive learners such as feedforward neural networks do not directly provide probability distributions on the output values. With decision trees and other logical representations, the output can be explained as a logical derivation and by appeal to a specific set of cases that support the decision. This has been an elusive task for neural networks [Russell and Norvig, 1995]. Without a way to probe the networks for these supportive cases, a confidence interval on the network cannot be accurately constructed.

However, multiple techniques exist for estimating the saliency, importance, relevance, or decision-making power (DMP), of individual elements within the structure of a trained neural network, including the input nodes connected to the normalized data. Various complexity regularization techniques have been set forth in the last decade for this kind of estimate. With a DMP estimate of each input, the supportive cases for a network's decision can be estimated, a confidence interval constructed, and the reliability and confidence in a network's decision increased, resulting in a box that is no longer completely black and mysterious.

Here, the DMP input vector is calculated using five separate methods: weight magnitude, signal variation (new method), input elimination, sensitivity analysis, and second order sensitivity. The input space is then renormalized based on the DMP estimates to scale up important inputs and scale down unimportant ones. Then, a general regression neural network, GRNN, is trained on the modified data to mimic the original neural network model. In this way, the more accurate DMP estimates should result in more accurate GRNN results. This, in turn, directly reveals which saliency estimates are more precise in the calculation of input importance. The five DMP ranking methods are tested against each other using a GRNN on four separate data sets, renormalized according to the DMP calculation of the trained neural network's inputs. If a DMP method can be shown to be universally superior to the other four techniques, then it will most accurately renormalize the sample space to pick the supportive cases of a network's decision, through the local weighting function approximation of a general regression neural network.

ANN variable importance estimation methods

This section describes various methods for estimating the saliency or importance of each input in a neural network, after the network is first trained on normalized data. The raw decision making power, DMP, of each input i is calculated according to the chosen method and stored as J_i. Then, the DMP input vector, or importance vector $\vec{I}$, is calculated so that the values of the elements of the vector sum to unity:

$$I_i = \frac{J_i}{\sum_{k=1}^{N} J_k} \tag{1}$$

Where N is the number of input nodes in the network. The sub-sections that follow describe the various methods for calculating J_i for each input i.

Method 1: Weight magnitude

The magnitudes of the weights connecting nodes between two layers of a neural network have been shown to regularize the complexity of a neural network during training. This has been shown in a training technique called squared weight decay, where the cost function contains an additional term $\lambda \sum_{ij} W_{ij}^2$ for each ij weight in the network, where the scalar λ controls the strength of the penalty term. In this system, the larger weights are penalized to a greater extent than the smaller ones. This essentially reduces the memorization power of the network, since the signals through the hidden and output nodes are pushed into the linear regions of the transfer functions. The greater the magnitude of a

weight, the more it contributes to the (nonlinear) potential decision making power of the network. Therefore, in this way, larger weights are more important than smaller ones.

Refer to Figure 1. This depicts a trained neural network, where the magnitudes of the connection weights are shown by the thickness of the lines connecting the input nodes (squares), hidden nodes (circles) and output node (triangle). The values of the weights W_{11} and W_{23} are also shown in the figure. One of the simplest ways of estimating each raw element of the DMP input vector is by summing the absolute value of the weights fanning out of each input node as follows:

$$J_i = \sum_{j=1}^{Nfanout} | W_{ij} | \qquad (2)$$

Where *Nfanout* weights are fanning out of input i to hidden node j in the first hidden layer.

Method 2: Signal variation

This subsection introduces a new way of probing the internal structure of an artificial neural network. That is, it provides a new estimate for how important each weight and node is in the network. With this estimate, two input nodes or weights can be directly compared to each other in terms of their importance. In the same way, unimportant nodes and weights can be targeted for structural learning purposes. The calculation is an extension of simple traditional methods.

Complexity-regularization procedures typically assign the importance of connection weight w_{ij} according to some monotonically increasing function of the magnitude of the weight. Here, an observation is made that a more accurate estimate involves the addition of another term: the variance of the signal exiting neuron i and entering w_{ij}, which connects neuron i to neuron j. This variance is determined by using the distribution of the signal exiting neuron i across all patterns.

Let the static local importance of weight w_{ij} to node j, L_{Wij}, be directly proportional to the variance of the signal entering node j. This assumption in fact must be valid in the limit as the variance approaches zero. In this case, the signal can be replaced by a constant, which means that the weight can be completely eliminated without any loss as long as that constant becomes part of the threshold. So, let the local importance of weight w_{ij} to node j, be calculated by the following relation:

$$L_{Wij} = \left| w_{ij} \sigma[x_i] \right| \tag{3}$$

Where x_i represents the signal exiting node i and $\sigma[x_i]$ is the standard deviation of x_i across all training patterns. For reference, see Figure 2. To obtain the normalized local importance of the weight w_{ij} to node j, $\bar{L}_{Wij}$, it must be compared to the importance of other weights fanning into node j. Let the normalized local importance of weight w_{ij} to node j be normalized according to the following equation:

$$\bar{L}_{Wij} = \frac{L_{Wij}}{\sum_{k=1}^{Nfanin} L_{Wkj}} \tag{4}$$

Where *Nfanin* is the number of weights fanning into node j. Now that each weight's local importance has been calculated, the global importance of each node and weight in the network can be estimated. The reason that local and global importance estimates differ is because the information transmitted through a weight must at least reach an output node for that weight to have any chance of being important.

A simple example can demonstrate this effect. Suppose that the normalized local importance of connection weight w_{ij} is large, due to a relatively large weight magnitude $|w_{ij}|$, a large signal variance $\sigma^2[x_i]$, or both. This means absolutely nothing if, say every weight fanning out of node j is zero. No information transmitted across w_{ij} can possibly reach the output node, so the value of $\bar{L}_{Wij}$ is meaningless.

For the importance estimate to be complete, the level of information successfully transmitted across the network from each weight w_{ij} must be accounted for. The calculations above dealt with how important a weight was to the node above it, and they were made possible by a forward recall on the data at the current point in weight space. The calculations below estimate how important each weight and node is to the network. They are made possible by a backward propagation of local importance calculations through the network using the previously calculated values for $\bar{L}_W$.

To begin the global importance estimates, distribute the global importance G_j^l across the output layer l for each output j. Assuming each output is equally important, the following relation holds at the output layer:

$$G_j^l = \frac{1}{O} \tag{5}$$

Where O is the number of outputs. Relatively speaking, each weight connected to an output node is only as important as the node it's connected to. Because of this dependency, let the global importance of each weight G_{Wij} be determined by the following:

$$G_{Wij} = G_j^l \overline{L}_{Wij} \tag{6}$$

Then, working backwards from the output layer, calculate the global importance for each node i in the previous layer, $l-1$, the same way. Once again the importance of the nodes below can only be as important as the weights connected to them from above. So, calculate the global importance of node i in layer $l-1$, G_i^{l-1}, according to the following relation:

$$G_i^{l-1} = \sum_{j=1}^{Nfanout} G_{Wij} \tag{7}$$

Where *Nfanout* is the number of connections above node i. Proceed backwards from the output layer to the input layer until all global calculations have been completed. At this

point, the global importance of each weight and node in the network has been estimated. Due to the normalization process mentioned above, the global importance of all of the inputs sum to unity. Therefore, the calculation for J_i is bypassed and the elements of the input importance vector is simply the following:

$$I_i = G_i^1 \tag{8}$$

Method 3: Input elimination

Refer to Figure 3. The network on the left represents the usual trained neural network being fed by the normalized data with a resulting error computed as the normalized RMS at the output layer. The central network represents the same trained neural network fed by the same data, except the first input is substituted with the average value across all patterns in the training set, and the resulting normalized error is computed as RMS_1'. Similarly, the network on the right is fed the same data, except the second input is substituted with the average value across all patterns in the training set, and the resulting normalized error is computed as RMS_2'. In this way, the neural network is probed in an attempt to find the inputs that are most important to it. The method is to essentially delete each input, substituting the input with its average value, calculating the increase in error, and storing the result. After the resulting increase in normalized error has been computed for each input, the raw DMP of each input i is computed as:

$$J_i = \Delta RMS_i = RMS_i' - RMS \tag{9}$$

This method is similar to the saliency estimates used in a common network pruning method of complexity regularization [Mozer & Smolensky, 1989].

Method 4: Sensitivity analysis

Refer to Figure 4. This method is derived from the cost function minimized during the training process. The ANN batch error for a trained neural network is typically given by the following:

$$E = \frac{1}{2}\sum_{p=1}^{n} e_p^2 \tag{10}$$

Where e_p is the difference between the target and modeled output for pattern p and n is the total number of training patterns. For each pattern, a small perturbation to each i input x_i results in a small perturbation in e_p, with the sensitivity of the error given by:

$$\frac{\delta e}{\delta x_i}(p) = \lim_{\Delta x_i \to 0} \frac{\Delta e}{\Delta x_i}(p) \tag{11}$$

This calculation can be carried out by backpropagating the derivative of the error for each pattern back to the input layer. Using these sensitivities, the raw DMP of each input i is computed as:

$$J_i = \sum_{p=1}^{n} \left| \frac{\delta e}{\delta x_i}(p) \right|$$

(12)

However, a very functionally similar result can be obtained by studying the sensitivity of the output with respect to each input, also using the chain rule in backpropagation without the error term. Since there is no requirement for an error term, a target value is not needed for each pattern, allowing for the DMP input vector calculation over a set without known outputs. Because of this advantage and the extreme similarity in the end result, the following formula will be used here:

$$J_i = \sum_{p=1}^{n} \left| \frac{\delta y}{\delta x_i}(p) \right|$$

(13)

Method 5: Second order sensitivity

Refer to Figure 5. In Optional Brain Damage [Le Cun et al, 1990] the importance or saliency of a weight is measured by estimating the second derivative of the error function with respect to the weight. The algorithm assumes that all the off-diagonal terms of the Hessian matrix are zero. The diagonal terms can be calculated by a modified backpropagation rule. Applying the same logic back to the input nodes, a second order (error) sensitivity analysis can be applied through the following equation:

$$\frac{\delta^2 e}{\delta x_i^2}(p) = \lim_{\Delta x_i \to 0} \frac{\Delta\left(\delta e / \delta x_i\right)}{\Delta x_i}(p)$$

(14)

This calculation can be carried out by backpropagating the second derivative of the error for each pattern back to the input layer, or by perturbing the results from the first order sensitivity analysis by a small Δx_i for each input. Using this second order analysis, the raw DMP of each input i is computed as:

$$J_i = \sum_{p=1}^{n} \left| \frac{\delta^2 e}{\delta x_i^2}(p) \right|$$

(15)

As with the first order sensitivity analysis, the end result can be approximated very accurately without the need for target values on the outputs. Therefore, the DMP input vector calculation using a second order sensitivity analysis will be determined here as follows:

$$J_i = \sum_{p=1}^{n} \left| \frac{\delta^2 y}{\delta x_i^2}(p) \right|$$

(16)

Weighted distance metric

A neural network has been trained. Once a method is chosen for computing the importance of each variable to the trained neural network, the input space can be

renormalized to scale up important variables and scale down unimportant variables. In this way, individual patterns from the data set can be compared by using a weighted distance metric, WDM, so that:

$$D_p^2 = \sum_{i=1}^{Ninputs} I_i \left(x_i - x_{i,p}\right)^2 \tag{17}$$

The effect and usefulness of this will become apparent after the discussion of general regression neural networks and its use of the Euclidean distance metric.

General regression neural networks (GRNN)

A general regression neural network (GRNN) [Specht, 1991], is a feedforward neural network based on Nadaraya-Watson kernel regression, also reinvented in the neural network literature by Schioler and Hartmann. (Kernels are also called "Parzen windows".) They can be thought of as normalized RBF networks with a hidden unit centered at every training pattern. These RBF units are called "kernels" and are usually Gaussian pdf's. The output is just a weighted average of the target values of the training patterns (cases) close to the given input pattern, the closeness being determined by the Euclidean distance between the two patterns. For a GRNN, the expected (modeled) value(s) of the output(s) given the input(s) are determined by:

$$E[y \mid \vec{x}] = \frac{\int_{-\infty}^{\infty} y \cdot f(\vec{x}, y)\, dy}{\int_{-\infty}^{\infty} f(\vec{x}, y)\, dy}$$

(18)

Where y is the output and $\vec{x}$ is the input vector. The above representation for a GRNN that models a finite set of points can be reduced to the following discrete formula for an estimation of $E[y \mid \vec{x}]$.

$$E[y \mid \vec{x}] \approx y(\vec{x}) = \frac{\sum_{p=1}^{n} v_p y_p}{\sum_{p=1}^{n} v_p}$$

(19)

Where v_p is defined as

$$v_p = \exp\left(\frac{-D_p^2}{2\sigma_G^2}\right)$$

(20)

Also, D_p^2 is the Euclidean distance metric between the current $\vec{x}$ and that of pattern p in the training set, defined as

$$D_p^2 = (\vec{x} - \vec{x}_p)^T (\vec{x} - \vec{x}_p)$$

(21)

The single free parameter σ_G is determined by minimizing the predicted error sum of squares (PRESS) in an N-folding process.

Where an artificial neural network doesn't determine the supportive cases of its decision explicitly, a GRNN does this directly through the weighted averaging process, so that there is v_p support for case p in relation to $\vec{x}$. The main drawback of a GRNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. A GRNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. However, if an ANN is first trained on the normalized data and a DMP estimate for each input is assigned, the data can be renormalized accordingly and seamlessly fed to a standard GRNN to mimic the output of the neural network. The irrelevant inputs are renormalized away before being presented to the GRNN.

Instead of renormalizing the data before presenting it to the GRNN, an equivalent result is achieved by replacing the Euclidean distance metric used in a standard GRNN with the weighted distance metric (see above section) based on the normalized DMP input importance vector $\vec{I}$, which in turn is computed by a saliency method such as one of the five techniques described above. This modified GRNN will mimic the trained artificial neural network, sometimes with higher accuracy than the ANN itself.

Testing methodology

Various DMP ranking methods have been described for estimating the importance of each input node of a trained neural network. Once a weighted distance metric has been computed from the DMP estimates, a modified GRNN will mimic the original ANN. The

DMP methods themselves can then be directly compared to each other. Those that produce a modified GRNN that more accurately reconstructs the outputs of the original ANN show directly that they generate better saliency estimates for that particular problem. Four problems are chosen for use by this testing methodology. The last one, called the spooky particle data set, is introduced as a candidate benchmark for comparison between inductive learning methods.

Test problems

Four data sets were chosen for the experimental comparison between DMP ranking methods: two chaotic time series of varying dimensionality, a letter recognition problem, and the new benchmark spooky particle data set. In each case, the data was normalized and partitioned according to the corresponding descriptions below.

Mackey-Glass equation

The control of a chaotic process is a natural application for artificial intelligence techniques because the properties of a chaotic system typically result in an imprecise process model. Neural networks have shown some success at predicting chaotic time series [Mandilwar and Qammar, 1993], and so a chaotic series was chosen here. A problem that has received a lot of attention lately is the Mackey – Glass chaotic time series [Mackey and Glass, 1977]. The equation is as follows:

$$x(t+1) = (1-b)x(t) + \frac{ax(t-\tau)}{1+\{x(t-\tau)\}^{10}}$$

(22)

Where a, b, and τ are set to 0.2, 0.2, and 17 respectively. The initial condition is taken to be $x(0)$=0.5. The objective is to predict $x(t+85)$ given $x(0)$, $x(t-6)$, $x(t-12)$, and $x(t-18)$ [Schmitz and Aldrich, 1999]. Iterations t = 4001 to 4500 are used for the training set and patterns 4501 to 5000 are used for the test set.

Chaos-13 data set

The above specification generates only 4 variables in a time series, where $x(t+85)$ is predicted given $x(0)$, $x(t-6)$, $x(t-12)$, and $x(t-18)$. Here, a benchmark called the "Chaos-13" data set is generated using a chaotic time series based on Verhulst dynamics [Peitgen and Richter, 1986]. The data here (population growth model) has 13 input variables $\{x(0), x(t-1), x(t-2), \ldots, x(t-12)\}$ and a single output $x(t+1)$. Also, 1858 patterns were reserved for the training set and 619 for the test set. In this way, the data contains a similar number of training and testing cases but a significantly larger number of dimensions. This should highlight the difference between good generalizers and those that are cursed by high dimensionality.

Letter recognition data set

The objective for this set of data is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted

into 16 primitive numerical attributes (statistical moments and edge counts) that were then scaled to fit into a range of integer values from 0 through 15. The first 16000 items comprised the training set and the remaining 4000 items were used to predict the letter category. This is a fairly standard benchmark problem for inductive learning methods. For simplicity, the classification of the letter A was only considered here, not all 26 capital letters.

Spooky particle data set

This sub-section briefly introduces a new data set to the field of data modeling as a benchmark for inductive learning machines. Its intended purpose is to provide a benchmark for model comparison and a resource for testing nonlinear modeling methods. Although the set is artificially generated, it can be thought of in terms of real-world physics. The "spooky" description of a particle's behavior originated when Albert Einstein became intrigued with experiments involving the quantum entanglement of twin-photons that originated from a common parent photon. The results apparently suggested that the fundamental particles behave like psychic geniuses, each knowing the other's state instantaneously without regard to the distance separating them. He sneered at the very possibility of such a thing, calling the process "spooky action at a distance."

The spooky particle data set introduced here includes the set of known events that a single particle has experienced in its universe throughout all time. If the particle is in fact a psychic genius, then it should instantly know some aspect of its final state given any initial state. The goal here is to teach the particle how to become psychic in some way by allowing

it to learn from the data set generated by its surrounding universe. A simple example is the best way to demonstrate this.

Let the universe be 3 dimensional (not counting time as a dimension), void of gravity and other such forces, and consist of an empty rectangular box enclosing the volume spanned by $(0,0,0)$ to (l,w,h). Let the particle's initial state in its universe be given by its initial position and velocity in the box $[X_0, V_{x0}, Y_0, V_{y0}, Z_0, V_{z0}]$. Let its final state be given by when it collides with one of the inner walls of the box, and mark this state with T_t, or time until impact from its initial state. If the particle is in fact psychic it should instantly know T_t given any initial state in that universe. Thus many examples can be generated using the specified laws of physics to obtain a data set having inputs $[X_0, V_{x0}, Y_0, V_{y0}, Z_0, V_{z0}]$ and desired output T_t. For this discussion, let the particle behave according to simple Newtonian rules of motion and let the rest of the universe be completely static.

The universe could contain internal (nonmoving) objects, have irregular boundaries, exist in a high number of dimensions, utilize gravity, etc. There are a wide variety of universes and corresponding data sets that can be generated which exhibit very interesting nonlinear properties. The dynamic particle only learns the laws of physics in its surrounding toy universe through example (by colliding with the walls or internal objects) and nothing else. It only becomes "spooky" after it learns how to predict T_t accurately based on previous known events.

The results presented in this paper are based on the simplified universe in a box described above with dimensions $2x5x1$, $V_{min} = 3$, and $V_{max} = 5$, with (X_0, Y_0, Z_0) uniformly distributed in the box and (V_{x0}, V_{y0}, V_{z0}) uniformly distributed in a spherical shell with an

inner and outer radius of V_{min} and V_{max} respectively. Also, 1000 points were generated for the training set and 1000 for the test set. This is a challenging number to use for this problem because T_i is somewhat difficult for the base model to learn but can still be generalized in the test set past the 0.8 r^2 range.

Results

Every neural network in this analysis was trained with 1000 iterations using the scaled conjugate gradient algorithm. Tables I-IV show the RMS error and R^2 results of the trained networks for both the training and test sets. Various network architectures were selected for each of the four data sets, such as "4x10x1" (4 inputs, 10 hidden nodes, 1 output) and "6x15x3x1" (6 inputs, 15 nodes in the first hidden layer, 3 nodes in the second hidden layer, 1 output). Table V consolidates the results of tables I-IV so that the best network architecture for each problem is displayed along with the corresponding test set RMS and R^2.

After the appropriate (best) networks were trained and filtered, their internal structures were probed using the variable importance (saliency) methods described above. As a means for comparison, a standard GRNN is introduced alongside the saliency estimation methods described. In this case, each input variable is equally important. See figures 6-9 for a graphical representation of all of these estimates.

Modified GRNN results, generated from the corresponding saliency estimates, are shown in tables VI-IX. From the test set RMS values in these tables, each saliency estimation method is ranked for each data set in terms of modified GRNN accuracy and

shown in Table X. The average overall rank is highlighted, as well as the rank based on the overall average test set R^2. The sensitivity analysis method had the highest overall rating out of the four chosen data sets, followed in close proximity by the input elimination method. The signal variation method was rated third, followed by the weight magnitude and then the 2nd order sensitivity method. In last place was the standard GRNNs approach. This was expected, since an ordinary GRNN equally weights each input dimension (using a Euclidean distance metric) regardless of the relevance of each input.

Conclusions

The sensitivity analysis method for computing saliency estimates on the input vector was shown here to be the most accurate of the standard and 5 modified GRNN approaches. Since the curse of dimensionality decreases with the increase of accuracy of the neural network saliency estimates, a modified GRNN can essentially reconstruct the intentions of the original trained ANN with greater success than a standard GRNN (with success measured by the rating system in Table X). After probing the networks for an estimate of the decision-making power of each input node, the input space was renormalized accordingly so that the modified GRNN would find more precise supportive cases of the neural network's decisions. From this, a confidence interval on the network can be more accurately constructed. In this way, some of the blackness from the black box of an artificial neural network was diminished. Various methods for highlighting the internal structure of an ANN were compared in a completely quantitative way using the modified GRNN technique.

References

Bartlett, E. B. and A. Whitney (1999). "On The Use Of Various Input Subsets For Stacked Generalization," Intelligent Engineering Systems Through Artificial Neural Networks, Vol 9, 117-124.

Bartlett, E. B. (1994). "Dynamic node architecture learning: an information theoretic approach", *Neural Networks 7,* 129-140.

Bartlett, E. B. and K. Kim, (1993). "Error Bounds on the Output of Artificial Neural Networks," *ANS Trans.*, 69,197-199.

Blum, E. K. and L. K. Li, (1991). "Approximation Theory and Feedforward Networks," *Neural Networks*, 4, 511.

C. G. Carmichael (1997). "Experiments in advancing supervised-learning ANN techniques," Thesis(M.S), Iowa State University.

Cherkassky, S. and F. Mulier, (1998). *Learning From Data: Concepts, Theory, and Methods (Adaptive and Learning Systems for Signal Processing, Communications and Control Series).* John Wiley & Sons, New York, New York.

Cun, Y. Le, J. S. Denker, and S. A. Solla (1990). "Optimal brain damage," Advances in Neural Information Processing Systems 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 598-605.

Haykin, S., (1999). *Neural Networks: A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, New Jersey.

Peitgen, H.-O. and P. H. Richter (1986). *The Beauty of Fractals: Images of Complex Dynamical Systems*. Springer-Verlag, New York, New York.

Hecht-Nielsen, R. (1990). *Neurocomputing.* Addison-Wesley, Reading, Mass.

Kurkova, V. (1992). "Kolmogorov's theorem and multilayer neural networks," *Neural Networks.* 5,501-506

Lippmann, R. P., (1987). "An Introduction to Computing with Neural Nets," *IEEE Acoustics Speech and Signal Processing Magazine*, 4, 4.

Mackey, M. C. and L. Glass (1977). "Oscillations and Chaos in Physiological Control Systems," *Science*, 197, 287-289.

Mandilwar, D. K. and H. Qammar (1993). "Prediction of Chaotic Time Series: Neural Versus Fuzzy", *Proceedings of the 1993 International Fuzzy Systems and Intelligent Control Conference,* 189-199.

Miller, W. T., R. S. Sutton, and P. Werbos (Eds.), (1990). *Neural Networks for Control.* Press. Cambridge, Mass.

Mitchell, T., (1997). *Machine Learning.* McGraw-Hill, New York, New York.

M. C. Mozer and P. Smolensky (1989). "Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in Neural Information Processing Systems, 1, D. S. Touretzky , Ed. (Denver 1988), pp. 107-115.

Nadaraya, E.A. (1964). "On estimating regression", Theory Probab. Applic.10, 186-90.

Narendra, K. S. and K. Parthasarathy, (1990). "Identification and Control of Dynamic Systems Using Neural Networks," *IEEE Trans. Neural Networks,* 1, no. 1, 4.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, (1986). *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol. 1 & 2, MIT Press, Cambridge, Massachusetts.

Russell, S. and P. Norvig, (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey.

Schioler, H. and U. Hartmann (1992). "Mapping Neural Network Derived from the Parzen Window Estimator," *Neural Networks*, 5, 903 - 909.

Schmitz, G. P. J. and C. Aldrich (1999). "Combinatorial Evolution of Regression nodes in Feedforward Neural Networks," *Neural Networks*, 12, 175-189.

Specht, D. F., (1991). "A General Regression Neural Network," *I&E Transactions on Neural Networks*, 2(6), 568-576.

Upadhyaya, B. R., and E. Eryurek, (1992). "Application of neural networks for sensor validation and plant monitoring," *Nuclear Technology* 97, 170-176.

Venkatasubramanian, V., and K. Chan, (1989). "A neural network methodology for process methods," *Journal of Applied Meteorology* 31, 405-420.

Wichmann, N. L. and E. B. Bartlett (1997), "Ranking Input Variables using General Regression Neural Networks", *Intelligent Engineering Systems Through Artificial Neural Networks: Volume 7*, pp. 959 – 964.

Wichmann, N. L., (1997). "Variable Importance Ordering for Evapotranspiration using General Regression Neural Networks", *M.S. Thesis, Iowa State University, Ames Iowa.*

Wolpert, D. H., (1990). "A Mathematical Theory of Generalization: Part I and Part II," *Complex Systems*, 4, 151.

Tables and figures

Table I. Trained ANN results for the Mackey Glass data set over various architectures

ANN architecture	Train RMS	Train R^2	Test RMS	Test R^2
4x5x1	0.10439	0.815366	0.130573	0.674885
4x10x1	0.076649	0.900479	0.109252	0.775292
4x15x1	0.058155	0.942708	0.0937792	0.834942
4x20x1	0.0520089	0.954187	0.089492	0.848283

Table II. Trained ANN results for the Chaos-13 data set over various architectures

ANN architecture	Train RMS	Train R^2	Test RMS	Test R^2
13x5x1	0.0216319	0.963422	0.0211102	0.965011
13x10x1	0.0129933	0.986805	0.0137185	0.98515
13x15x1	0.011414	0.989825	0.0127839	0.987144
13x20x1	0.0108448	0.990811	0.0129216	0.986878

Table III. Trained ANN results for the spooky particle data set over various architectures

ANN architecture	Train RMS	Train R^2	Test RMS	Test R^2
6x5x1	0.100314	0.714426	0.0921829	0.696405
6x10x1	0.0668997	0.872959	0.0703212	0.821239
6x15x1	0.05937	0.899962	0.067733	0.837427
6x15x3x1	0.0431663	0.947017	0.0564924	0.883948

Table IV. Trained ANN results for the letter recognition data over various architectures

ANN architecture	Train RMS	Train R^2	Test RMS	Test R^2
16x10x1	0.0475129	0.942623	0.0640565	0.893993
16x20x1	0.0401445	0.958504	0.0523863	0.928103
16x30x1	0.0365695	0.96549	0.0482026	0.938887
16x20x10x1	0.0041231	0.99956	0.0330517	0.970926

Table V. Best trained ANN results for the various data sets implemented here

Data	Best architecture	Test RMS	Test R^2
Mackey Glass	4x20x1	0.089492	0.848283
Chaos-13	13x15x1	0.0127839	0.987144
Spooky particle	6x15x3x1	0.0564924	0.883948
Letter recognition	16x20x10x1	0.0330517	0.970926

Table VI. Modified GRNN variable importance results for the Mackey Glass data set

Method	Train RMS	Train R^2	Test RMS	Test R^2
Standard GRNN	0.0452069	0.967993	0.0956164	0.837585
Weight Magnitude	0.0449199	0.968583	0.0964478	0.832647
Signal Variation	0.0473613	0.965365	0.0994391	0.820735
Input Elimination	0.0456695	0.967134	0.094233	0.844098
Sensitivity Analysis	0.0448006	0.968621	0.0946378	0.839878
2nd Order Sensitivity	0.0469964	0.965417	0.0963018	0.83134

Table VII. Modified GRNN variable importance results for the Chaos-13 data set

Method	Train RMS	Train R^2	Test RMS	Test R^2
Standard GRNN	0.0154028	0.983029	0.0477103	0.82115
Weight Magnitude	0.011328	0.990552	0.032454	0.918012
Signal Variation	0.00817994	0.994973	0.0230918	0.958336
Input Elimination	0.00684307	0.996449	0.016336	0.979046
Sensitivity Analysis	0.00646319	0.996826	0.0157299	0.980575
2nd Order Sensitivity	0.00664286	0.996645	0.0184852	0.973153

Table VIII. Modified GRNN variable importance results for the spooky particle data set

Method	Train RMS	Train R^2	Test RMS	Test R^2
Standard GRNN	0.0452492	0.958812	0.0968344	0.659622
Weight Magnitude	0.0256915	0.985253	0.086895	0.721119
Signal Variation	0.0538246	0.946441	0.0861475	0.748119
Input Elimination	0.0278866	0.981912	0.0856684	0.72922
Sensitivity Analysis	0.0457485	0.957699	0.0793102	0.777233
2nd Order Sensitivity	0.0445267	0.956258	0.105176	0.591411

Table IX. Modified GRNN variable importance results for the letter recognition data

Method	Train RMS	Train R^2	Test RMS	Test R^2
Standard GRNN	0.000516735	0.999993	0.022928	0.986014
Weight Magnitude	0.000486447	0.999994	0.020101	0.989243
Signal Variation	0.000599869	0.999991	0.0215581	0.98767
Input Elimination	0.000116334	1	0.0196598	0.989725
Sensitivity Analysis	0.000471386	0.999994	0.0234176	0.985431
2nd Order Sensitivity	0.000704893	0.999987	0.0272928	0.980196

Table X. Modified GRNN variable importance rank comparison for various data sets

GRNN Method	Mackey Glass	Chaos-13	Spooky particle	Letter recognition	**Average rank**	**Test set R^2 rank**
Standard GRNN	3	6	5	4	**4.5**	**6**
Weight Magnitude	4	5	4	2	**3.75**	**4**
Signal Variation	6	4	2	3	**3.75**	**3**
Input Elimination	1	2	3	1	**1.75**	**2**
Sensitivity Analysis	2	1	1	5	**2.25**	**1**
2nd Order Sensitivity	5	3	6	6	**5**	**5**

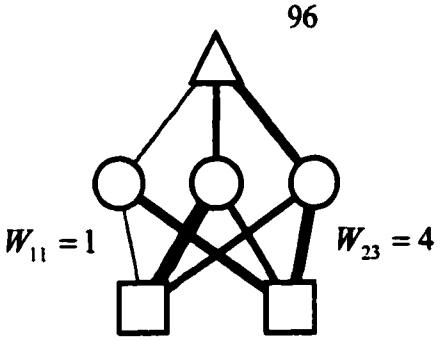


Figure 1. Weight magnitude

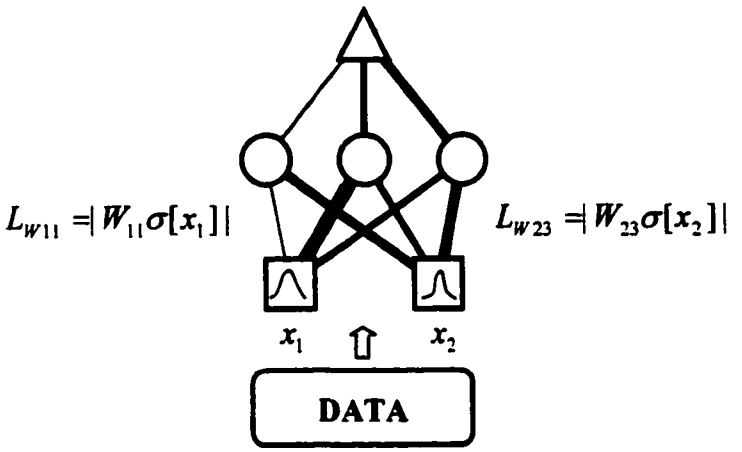


Figure 2. Signal variation

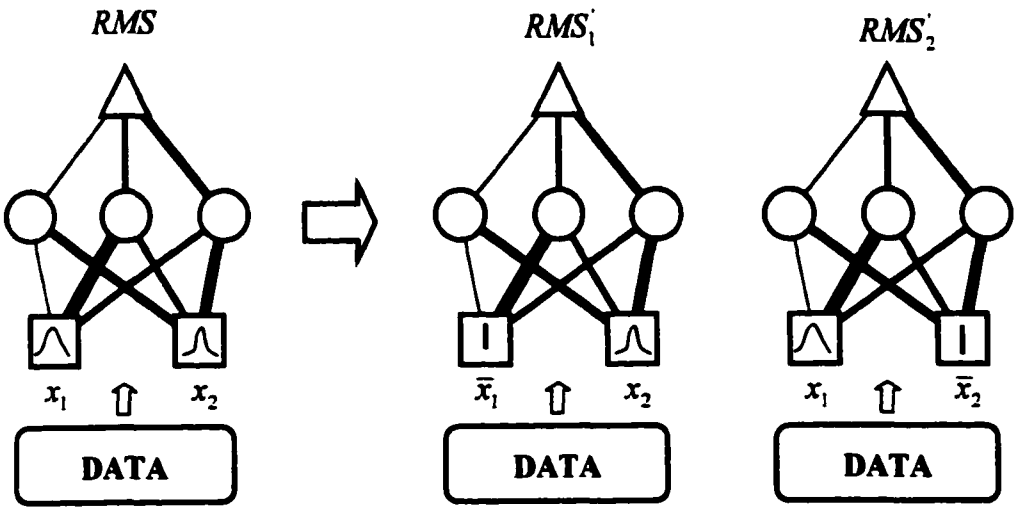
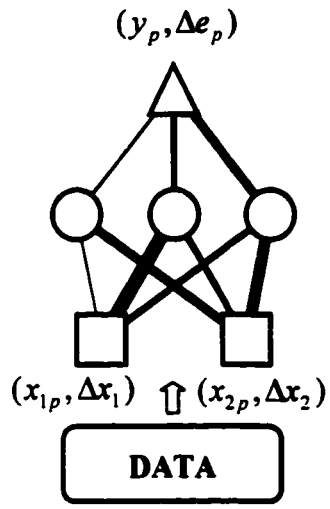
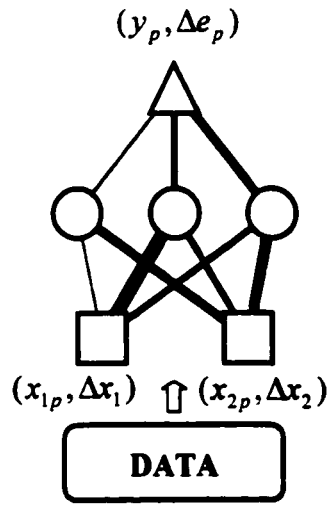


Figure 3. Input elimination



$$\frac{\delta e}{\delta x_i}(p) = \lim_{\Delta x_i \to 0} \frac{\Delta e}{\Delta x_i}(p)$$

Figure 4. Sensitivity analysis



$$\frac{\delta e}{\delta x_i}(p) = \lim_{\Delta x_i \to 0} \frac{\Delta e}{\Delta x_i}(p)$$

$$\frac{\delta^2 e}{\delta x_i^2}(p) = \lim_{\Delta x_i \to 0} \frac{\Delta\left(\delta e / \delta x_i\right)}{\Delta x_i}(p)$$

Figure 5. Second order sensitivity

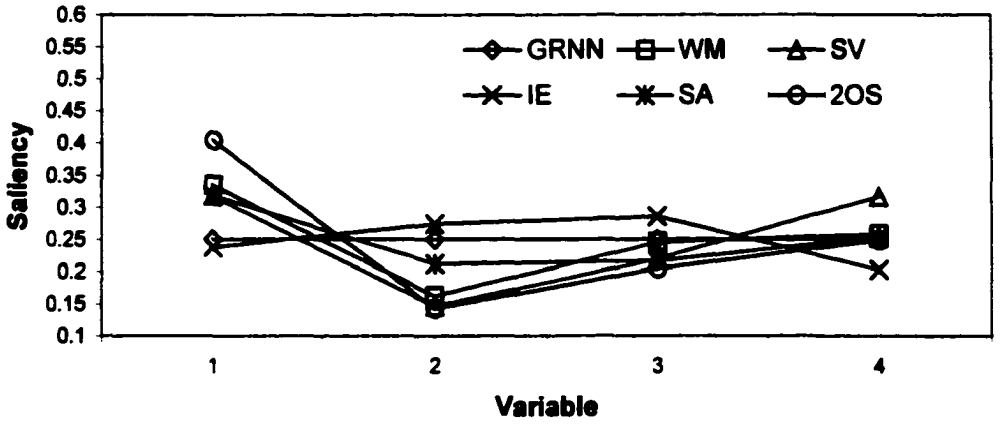


Figure 6. Variable importance estimates on the Mackey Glass data set

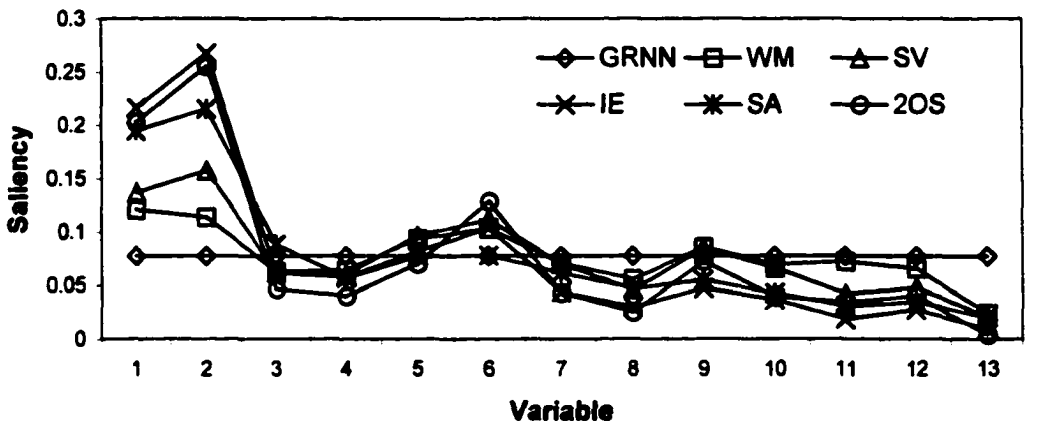


Figure 7. Variable importance estimates on the chaos-13 data set

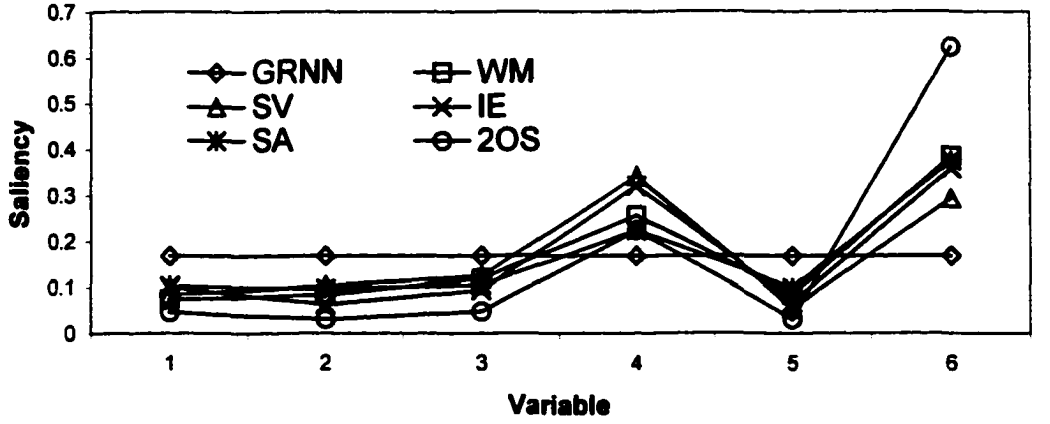


Figure 8. Variable importance estimates on the spooky particle data set

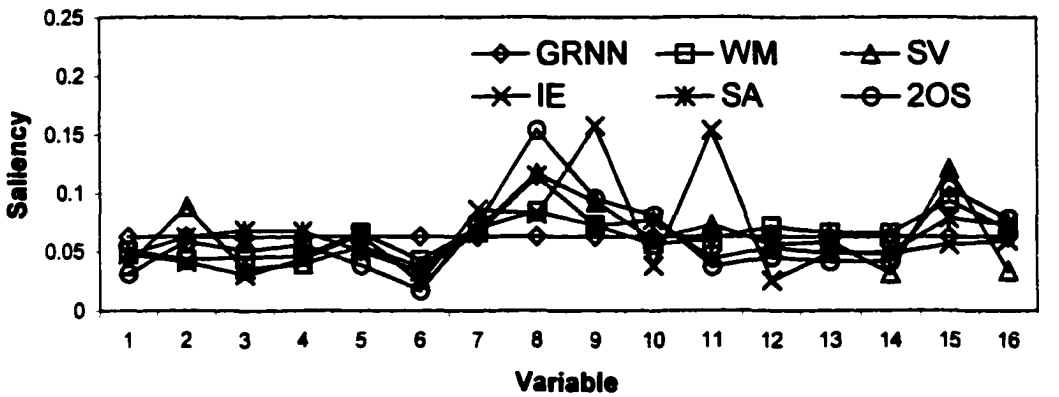


Figure 9. Variable importance estimates on the letter recognition data set

Nomenclature

ΔRMS_i	Resulting increase in RMS after eliminating ANN input i
λ	Weight decay penalty term
σ_G	Free parameter optimized by a GRNN
$\sigma^2[x_i]$	Signal variation exiting input x_i across all training patterns
ANN	Artificial neural network
D	Distance metric used by a GRNN
DMP	Decision making power of an element of a neural network
e_p	Error component of the batch training error for pattern p
E	Typical ANN batch training error
G_i^l	Global importance of node i in layer l using the signal variation method
G_{Wij}	Global importance of weight W_{ij} using the signal variation method
GRNN	General regression neural network
i	Input subscript
$\bar{I}$	DMP input importance vector (after normalization)
j	ANN hidden node subscript
$\bar{J}$	Raw DMP input importance vector (before normalization)
l	ANN hidden layer subscript
L_{Wij}	Local importance of weight W_{ij} using the signal variation method
$\bar{L}_{Wij}$	Normalized local importance of W_{ij} using the signal variation method
p	Training data pattern subscript

n	Number of training patterns
N	Number of inputs
Nfanin	Number of weights fanning into a node in an ANN
Nfanout	Number of weights fanning out of a node in an ANN
O	Number of outputs
PRESS	GRNN predicted error sum of squares
RMS	Root mean squared error of a trained ANN
RMS_i'	Resulting *RMS* after eliminating ANN input i
v_p	GRNN local weighting factor for pattern p
WDM	Weighted distance metric, for use by a GRNN
W_{ij}	Local weighting factor used by DCM
$\vec{x}$	Input space
y	Output, generic description
$y(\vec{x})$	GRNN estimate for $E[y \mid \vec{x}]$
$y \mid \vec{x}$	Theoretical behavior of y in the area of $\vec{x}$

CHAPTER 4. COMBINING NEURAL NETWORKS WITH NEAREST NEIGHBORS

A paper submitted to the Journal of Neural Networks

Craig G. Carmichael and Eric B. Bartlett

Department of Electrical and Computer Engineering,

Iowa State University, Ames, IA, 50011

Abstract

A series of methods are introduced that combine the global nonlinear regression capability of feed-forward, supervised-learning artificial neural networks with the local averaging capability of nearest neighbor approaches. The goal is to combine both methods in a way that outperforms a simple averaging approach. Three standard approaches will be discussed, ranging from a simple nearest neighbors method to artificial neural networks. The first method, k-nearest neighbors, KNN, is provided as a basis for comparison. General regression neural networks, GRNN, will then be introduced as a locally weighted variant of KNN. Next, artificial neural networks, ANN, will be briefly. For each method KNN, GRNN, and ANN, the respective model will be modified and enhanced during a secondary process. During the primary process, the neural network model will determine an estimate for the importance of each of the inputs in the data set, resulting in an importance vector with dimension N (number of inputs). Then the input space will be re-normalized based on this importance vector. The new normalized data set will be fed through the KNN and GRNN,

resulting in increased performance due to this feature weighting process (models FWKNN and FWGRNN). Then two more advanced methods, LMLR and NNA, will be introduced as hybrid neural network, nearest neighbor approaches. These seven methods are tested and compared on four separate data sets, one of which is introduced as a generic data set for future comparison between inductive learning machines. The resulting analysis demonstrates that an ANN is a specific case of NNA, and so this allows for NNA to outperform standard neural networks in terms of generalization.

Keywords

Artificial Neural Network, General Regression Neural Network, Local Modeling, Feedforward Networks, Data Modeling

Introduction

If model accuracy is important and the functionality underlying the corresponding data is unknown, then training an artificial neural network to learn the data is a common first step. After this global modeling approach, it should be a natural step to try various local modeling techniques in search of greater accuracy. Karl Steinbuch proposed neural network designs that explicitly remember the training experiences and used a local representation to do nearest neighbor lookup. They used a layer of hidden units to compute an inner product of each stored vector with the input vector. A winner-take-all circuit then selected the hidden unit with the highest activation. This type of network can find nearest neighbors or best matches using a Euclidean distance metric (Atkeson and Schaal, 1995). Here, instead a

straightforward scaling of the inputs based on the trained neural network will allow for an improved weighted distance metric (WDM) to be fed to local weighting methods.

This paper is intended to tie together some concepts that are scattered throughout the neural network literature. Inductive learners such as feedforward neural networks do not directly provide probability distributions on the output values. With decision trees and other logical representations, the output can be explained as a logical derivation and by appeal to a specific set of cases that support the decision. This has been an elusive task for neural networks [Russell and Norvig, 1995]. Without a way to probe the networks for these supportive cases, a confidence interval on the network cannot be accurately constructed. Substitute the words "nearest neighbor" for "supportive" and it can be seen that the supportiveness of a training pattern to the current case is directly related to their proximity to each other as viewed by the trained neural network. This is one very good reason why local modeling techniques should supplement artificial neural networks.

Also, according to the literature, the main drawback of a GRNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. A GRNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. However, if a trained ANN is already present, then a very simple approach is available to squash the irrelevant inputs and stretch the relevant ones so that the renormalized data can be fed to local modeling techniques, eliminating the curse of dimensionality. Locally weighted learning is critically dependent on the distance function, and probing the ANN provides a simple solution for this.

Seven methods are described which have varying degrees of local and global properties, three of which are standard (KNN, GRNN, and ANN). After the ANN is probed

and the features (inputs) are weighted (according to the viewpoint of the neural network), the ANN-based WDM is used by KNN and GRNN to produce the methods FWKNN and FWGRNN. The last two methods (LMLR and NNA) are more advanced hybrid networks. Localized multivariate linear regression (LMLR) uses the local pattern-weighting scheme of the FGRNN to allow for multivariate linear regression in the local region specified by the WDM. The Neural Neighbors Algorithm (NNA) is introduced as a way to combine the strengths of FWGRNN, ANN, and LMLR in a stacking approach that is more general, and more computationally intensive, than an ANN. The goal is improved accuracy.

Four test problems are used to compare the performance of each new modeling process. The first problem is derived from the Macky-Glass equations. The second is a chaotic time series consisting of 13 inputs. The third problem is called the spooky particle data set, which is introduced as a candidate benchmark for comparison between inductive learning methods. The fourth is a classification problem.

Local and global modeling methods

This section describes various techniques for local and global modeling. The k-nearest neighbors (KNN) method is introduced as a simple local modeling approach. General regression neural networks (GRNN) are then introduced as a variant of KNN with a local weighting factor. Artificial neural networks (ANN) are then briefly described as a global modeling approach. All of these techniques are standard methods used for inductive learning, or learning from data.

k-Nearest Neighbors (KNN)

The k-Nearest Neighbor algorithm is a simple method that stores all available examples and classifies new instances based on a similarity measure. A variant of this algorithm addresses the task of function approximation. The distance between two example vectors is regarded as a measure for their similarity. To classify a new instance e from the set of stored examples, the k examples most similar to e are determined. The new instance is assigned the class most of the k examples belong to. This approach is also suited for function approximation. Instead of assigning the most frequent classification among the k examples most similar to an instance e, an average of the function values of the k examples is calculated as the prediction for the function value e. A variant of this approach calculates a weighted average of the nearest neighbors. Given a specific instance e that shall be classified, the weight of an example increases with increasing similarity to e. For simplicity, the KNN described here uses a Euclidean distance metric with k=1.

General regression neural network (GRNN)

A general regression neural network [Specht, 1991] can be thought of as the weighted average variant of KNN described above. It is a feed-forward neural network based on Nadaraya-Watson kernel regression, also reinvented in the neural network literature by Schioler and Hartmann. (Kernels are also called "Parzen windows".) They can be thought of as normalized RBF networks with a hidden unit centered at every training pattern. These RBF units are called "kernels" and are usually Gaussian pdf's. The output is just a weighted average of the target values of the training patterns (cases) close to the given input pattern, the closeness being determined by the Euclidean distance between the two patterns. For a

GRNN, the expected (modeled) value(s) of the output(s) given the input(s) are determined by:

$$E[y \mid \vec{x}] = \frac{\int_{-\infty}^{\infty} y \cdot f(\vec{x}, y)\, dy}{\int_{-\infty}^{\infty} f(\vec{x}, y)\, dy} \tag{1}$$

Where y is the output and $\vec{x}$ is the input vector. The above representation for a GRNN that models a finite set of points can be reduced to the following discrete formula for an estimation of $E[y \mid \vec{x}]$.

$$E[y \mid \vec{x}] \approx y(\vec{x}) = \frac{\sum_{p=1}^{n} v_p y_p}{\sum_{p=1}^{n} v_p} \tag{2}$$

Where v_p is defined as

$$v_p = \exp\left(\frac{-D_p^2}{2\sigma_G^2}\right) \tag{3}$$

Also, D_p^2 is the Euclidean distance metric between the current $\vec{x}$ and that of pattern p in the training set, defined as

$$D_p^2 = (\vec{x} - \vec{x}_p)^T (\vec{x} - \vec{x}_p)$$

(4)

The single free parameter σ_G is determined by minimizing the predicted error sum of squares (PRESS) in an N-folding process.

Where an artificial neural network doesn't determine the supportive cases of its decision explicitly, a GRNN does this directly through the weighted averaging process, so that there is v_p support for case p in relation to $\vec{x}$. Another advantage of this method for local modeling is the direct extension to an error estimation procedure. For example:

$$\sigma[y \mid \vec{x}] = \left(E[y^2 \mid \vec{x}] - E^2[y \mid \vec{x}]\right)^{1/2}$$

(5)

The expected value of $y^2 \mid \vec{x}$ follows in exactly the same fashion as the calculated expected value of $y \mid \vec{x}$:

$$E[y^2 \mid \vec{x}] \approx y_2(\vec{x}) = \frac{\sum_{p=1}^{n} v_p y_p^2}{\sum_{p=1}^{n} v_p}$$

(6)

Where the same σ_G is used for both $E[y \mid \vec{x}]$ and $E[y^2 \mid \vec{x}]$. The following equation represents the predicted (modeled) standard deviation about the mean for the pattern at input space $\vec{x}$.

$$\sigma[y \mid \vec{x}] \approx \sigma_m(\vec{x}) = \left(y_2(\vec{x}) - y^2(\vec{x})\right)^{1/2} \tag{7}$$

Artificial neural networks (ANN)

Artificial neural networks are simplified models of the biological counterpart. They typically consist of processing units, weighted interconnections between the processing units, an activation rule to propagate signals through the network, and a learning rule to specify how the weighted interconnections are adjusted during the training phase. The most common ANN models are feed-forward networks structured in layers (see Figure 1), with an input layer where data is presented as a set of numeral patterns, one or more hidden layers to store the intermediate results of each pattern, and an output layer that contains the resulting nonlinear mapping of the network. Usually, the training phase consists of a gradient descent method, such as the scaled conjugate gradients algorithm (SCG), in weight space. The weight vector $\vec{w}$ is optimized in this way to minimize the error between the desired output(s) and model output(s) over the entire training set. Although these neural networks provide a global nonlinear mapping from the input space to the desired output target space, they can be queried in a way that produce a local nonlinear mapping. In this way, they can be directly related to KNN and GRNN methods.

Input scaling

Locally weighted learning is critically dependent on the distance function. There are many different approaches to defining a distance function, and this section briefly discusses

how to scale the inputs (for the distance function) from the viewpoint of a trained neural network. The relative importance of the input dimensions in generating the distance measurement depends on how the inputs are scaled (Atkeson, Moore, and Schaal, 1997). Various techniques will be described below which utilize this altered distance metric. First, a method will be described for input scaling that will then be fed to a weighted distance metric for local modeling.

A trained artificial neural network can be used to directly assess how much the inputs should be stretched or squashed for locally weighted learning. There are various techniques that can be used to assess the decision-making power (DMP), or relevance, of each input j in the trained ANN. The raw DMP of each input R_j can be calculated by summing the weights fanning out of input j, eliminating input j and measuring the increase in training RMS, etc. There are many ways to do this, some of which are derived from complexity regularization techniques. What follows is a reliable method for calculating R_j for all j by using a sensitivity analysis.

Refer to Figure 1. This method is derived from the cost function minimized during the training process of an ANN. The batch error for a trained neural network is typically given by the following:

$$E = \frac{1}{2} \sum_{p=1}^{Ntrain} e_p^2 \tag{8}$$

Where e_p is the difference between the target and modeled output for pattern p and *Ntrain* is the total number of training patterns. For each pattern, a small perturbation to each j input x_j results in a small perturbation in e_p, with the sensitivity of the error given by:

$$\frac{\delta e}{\delta x_j}(p) = \lim_{\Delta x_j \to 0} \frac{\Delta e}{\Delta x_j}(p) \tag{9}$$

This calculation can be carried out by backpropagating the derivative of the error for each pattern back to the input layer in normalized space. Using these sensitivities, the relevance or raw DMP of each input j is computed as:

$$R_j = \sum_{p=1}^{Ntrain} \left| \frac{\delta e}{\delta x_j}(p) \right| \tag{10}$$

However, a very functionally similar result can be obtained by studying the sensitivity of the output with respect to each input, also using the chain rule in backpropagation without the error term. Since there is no requirement for an error term, a target value is not needed for each pattern, allowing for the DMP input vector calculation over a set without known outputs. Because of this advantage and the extreme similarity in the end result, the following formula will be used here:

$$R_j = \sum_{p=1}^{Ntrain} \left| \frac{\delta y}{\delta x_j}(p) \right|$$

(11)

From the raw DMP estimates in the input space, the normalized importance of each input can be determined by:

$$I_j = \frac{R_j}{\sum_{k=1}^{N} R_k}$$

(12)

So that $\sum_{j=1}^{N} I_j = 1$. The vector $\vec{I}$ provides the necessary information for scaling inputs.

Feature weighted k-nearest neighbors (FWKNN)

A major problem of the simple approach of KNN is that the vector distance will not necessarily be suited for finding intuitively similar examples, especially if irrelevant attributes are present. Therefore KNN is sensitive to the distance function because of the inherent sensitivity to the irrelevant features. Therefore, a feature weighted KNN approach is presented here (FWKNN) which utilizes the following distance metric based on $\vec{I}$ presented above:

$$D_p^2 = \sum_{j=1}^{N} I_j \left(x_j - x_{j,p}\right)^2$$

(13)

All else is the same as in KNN. The only difference is this weighted distance metric based on the DMP estimates of the neural network's inputs, which requires a trained ANN.

Feature weighted general regression neural network (FWGRNN)

The main drawback of a GRNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. A GRNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. However, if an ANN is first trained on the normalized data and a DMP estimate for each input is assigned, the data can be renormalized and squashed accordingly and seamlessly fed to a standard GRNN to mimic the output of the neural network. The irrelevant inputs are renormalized away before being presented to the GRNN. Call this modified GRNN a feature weighted GRNN (FWGRNN) because the input (feature) space is scaled in a non-uniform (weighted) way, based on the viewpoint of the trained ANN. This difference is conceptually very similar to that between KNN and FWKNN.

Instead of renormalizing the data before presenting it to the GRNN, an equivalent result is achieved by replacing the Euclidean distance metric used in a standard GRNN with the weighted distance metric used by the FWKNN method. This modified GRNN will mimic the trained artificial neural network, sometimes with higher accuracy than the ANN itself. This is similar to the weight adjusted k-nearest neighbor (WAKNN) approach (Han, Karypis, and Kumar, 1991), except here the features are scaled according to the viewpoint of the trained neural network.

Neural Neighbors Algorithm (NNA)

The Neural Neighbors Algorithm (NNA) fits a local regression surface to the nearest neighbors of each new $\bar{x}$ as determined by the viewpoint of the trained neural network. It is

a combination of multivariate linear regression in the local space of $\bar{x}$ and the global regression surface determined by the trained ANN. The locality of the space is determined once again by the scaling of more relevant inputs and the squashing of more irrelevant inputs, in the way discussed above. Again, this requires the original trained ANN. In describing this technique, first revisit multivariate linear regression (MLR), where the output $y_{MLR,p}$ for pattern p is determined by:

$$y_{MLR,p} = A_0 + A_1 x_{1p} + A_2 x_{2p} \ldots + A_N x_{Np} \tag{14}$$

Where $\{A_0, A_1, A_2, \ldots, A_N\}$ are the free parameters and the error E_{MLR} is determined by:

$$E_{MLR} = \sum_{p=1}^{Ntrain} (T_p - y_{MLR,p})^2 \tag{15}$$

Where T_p is the target for pattern p and *Ntrain* is the number of training examples. By taking the partial derivatives of the error with respect to each of the free parameters and setting all of the equations equal to zero, the following system of linear equations (where each of the summations is over *Ntrain*) becomes straightforward:

$$\begin{bmatrix} \sum_p 1 & \sum_p x_{1p} & \cdots & \sum_p x_{Np} \\ \sum_p x_{1p} & \sum_p x_{1p}^2 & & \vdots \\ \vdots & & \ddots & \vdots \\ \sum_p x_{Np} & \sum_p x_{1p}x_{Np} & \cdots & \sum_p x_{Np}^2 \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_N \end{bmatrix} = \begin{bmatrix} \sum_p T_p \\ \sum_p T_p x_{1p} \\ \vdots \\ \sum_p T_p x_{Np} \end{bmatrix}$$

(16)

In this case, using MLR requires solving the set of free parameters $\{A_0, A_1, A_2, \ldots, A_N\}$ only once.

Now, consider a first order polynomial fit in multidimensional space where the space surrounding each new $\vec{x}$ is fit in a similar fashion. Call this localized multivariate linear regression (LMLR), where the output $y_{LMLR,p}$ becomes:

$$y_{LMLR}(\vec{x}) = A_0(\vec{x}) + A_1(\vec{x})x_1 + A_2(\vec{x})x_2 \ldots + A_N(\vec{x})x_N$$

(17)

Where $\{A_0(\vec{x}), A_1(\vec{x}), A_2(\vec{x}), \ldots, A_N(\vec{x})\}$ are the free parameters, which are all functions of the current point in input space $\vec{x}$. In this case, the error $E_{LMLR}(\vec{x})$ is determined by:

$$E_{LMLR}(\vec{x}) = \sum_{p=1}^{Ntrain} w_p \left(T_p - y_{LMLR}(\vec{x}_p)\right)^2$$

(18)

Where each pattern p is weighted in the same manner as in FWGRNN, so that:

$$w_p = \exp\left(\frac{-D_p^2}{2\sigma_G^2}\right)$$

(19)

And the distance metric D_p^2 is non-Euclidean, with the same input scaling as that used in both FWKNN and FWGRNN. The resulting system of linear equations is very similar to that of ordinary MLR:

$$\begin{bmatrix} \sum_p w_p & \sum_p w_p x_{1p} & \cdots & \sum_p w_p x_{Np} \\ \sum_p w_p x_{1p} & \sum_p w_p x_{1p}^2 & & \vdots \\ \vdots & & \ddots & \vdots \\ \sum_p w_p x_{Ni} & \sum_p w_p x_{1p} x_{Np} & \cdots & \sum_p w_p x_{Np}^2 \end{bmatrix} \begin{bmatrix} A_0(\vec{x}) \\ A_1(\vec{x}) \\ \vdots \\ A_N(\vec{x}) \end{bmatrix} = \begin{bmatrix} \sum_p w_p T_p \\ \sum_p w_p T_p x_{1p} \\ \vdots \\ \sum_p w_p T_p x_{Np} \end{bmatrix}$$

(20)

This is a form of multivariate linear regression in the local region of $\vec{x}$ where the locality is determined by the underlying neural network. At this point, the local modeling formulation for LMLR is becoming very nonlinear, even though the term "linear" is used. However, this approach encounters problems as the summation terms involving w_p approach zero. This will happen if the current point in input space $\vec{x}$ is far away from all training patterns.

To solve this problem, the Neural Neighbors Algorithm (NNA) is introduced. If a global solution is best, the output $y_{MNR}(\vec{x})$ becomes the neural solution $y_{ANN}(\vec{x})$. If a local solution is best, $y_{MNR}(\vec{x})$ becomes more $y_{LMLR}(\vec{x})$ or $y_{GRNN}(\vec{x})$. Due to the local modeling uncertainty in regions of input space having few examples represented by the training set (where a GRNN and LMLR will become inaccurate), certainty will be provided by the global modeling solution of the ANN. Let the local representation $L(\vec{x})$ of the space surrounding $\vec{x}$ be determined by:

$$L(\vec{x}) = \sum_{p=1}^{Ntrain} w_p$$

(21)

On certain data sets, the accuracy of LMLR can be outstanding in regions of high $L(\vec{x})$, with the exception of a few points having a very high error, typically in regions of low $L(\vec{x})$. This problematic behavior can be overcome by stacking the results of the GRNN, LMLR, and ANN in the following way:

$$y_{MNR}(\vec{x}) = f\left(y_{GRNN}(\vec{x}), y_{LMLR}(\vec{x}), y_{ANN}(\vec{x}), L(\vec{x}), d_{GRNN-LMLR}(\vec{x}), d_{GRNN-ANN}(\vec{x}), d_{LMLR-ANN}(\vec{x})\right)$$

(22)

Where $f(\bullet)$ is a single hidden layer ANN. After training this neural network, $y_{MNR}(\vec{x})$ may often become $y_{LMLR}(\vec{x})$ at high values of $L(\vec{x})$, since this is where LMLR is most often highly accurate. The d terms represent the absolute differences between each of the three models. This will allow the top ANN to more easily decide whether a certain model should be considered to represent the output. Also, a 2nd order localized multivariate quadratic regression LMQR model can be included alongside the 1^{st} order LMLR and 0^{th} order FWGRNN models. As with LMLR, care should be taken when inverting matrices driven by low values of $L(\vec{x})$.

Test problems

Four data sets were chosen for the experimental comparison between various local and global modeling methods: two chaotic time series of varying dimensionality, a letter recognition problem, and the new benchmark spooky particle data set. In each case, the data was normalized and partitioned according to the corresponding descriptions below.

Mackey-Glass Equation

The control of a chaotic process is a natural application for artificial intelligence techniques because the properties of a chaotic system typically result in an imprecise process model. Neural networks have shown some success at predicting chaotic time series [Mandilwar and Qammar, 1993], and so a chaotic series was chosen here. A problem that has received a lot of attention lately is the Mackey – Glass chaotic time series [Mackey and Glass, 1977]. The equation is as follows:

$$x(t+1) = (1-b)x(t) + \frac{ax(t-\tau)}{1+\{x(t-\tau)\}^{10}} \tag{23}$$

Where a, b, and τ are set to 0.2, 0.2, and 17 respectively. The initial condition is taken to be $x(0)$=0.5. The objective is to predict $x(t+85)$ given $x(0)$, $x(t-6)$, $x(t-12)$, and $x(t-18)$ [Schmitz and Aldrich, 1999]. Iterations t = 4001 to 4500 are used for the training set and patterns 4501 to 5000 are used for the test set.

Chaos-13 data set

The above specification generates only 4 variables in a time series, where $x(t+85)$ is predicted given $x(0)$, $x(t-6)$, $x(t-12)$, and $x(t-18)$. Here, a benchmark called the "Chaos-13" data set is generated using a chaotic time series based on Verhulst dynamics [Peitgen and Richter, 1986]. The data here (population growth model) has 13 input variables $\{x(0), x(t-1), x(t-2), \ldots, x(t-12)\}$ and a single output $x(t+1)$. Also, 1858 patterns were reserved for the training set and 619 for the test set. In this way, the data contains a similar number of training and testing cases but a significantly larger number of dimensions. This should highlight the difference between good generalizers and those that are cursed by high dimensionality.

Letter recognition data set

The objective for this set of data is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) that were then scaled to fit into a range of integer values from 0 through 15. The first 16000 items comprised the training set and the remaining 4000 items were used to predict the letter category. This is a fairly standard benchmark problem for inductive learning methods. For simplicity, the classification of the letter A was only considered here, not all 26 capital letters.

Spooky particle data set

This sub-section briefly introduces a new data set to the field of data modeling as a benchmark for inductive learning machines. Its intended purpose is to provide a benchmark for model comparison and a resource for testing nonlinear modeling methods. Although the set is artificially generated, it can be thought of in terms of real-world physics. The "spooky" description of a particle's behavior originated when Albert Einstein became intrigued with experiments involving the quantum entanglement of twin-photons that originated from a common parent photon. The results apparently suggested that the fundamental particles behave like psychic geniuses, each knowing the other's state instantaneously without regard to the distance separating them. He sneered at the very possibility of such a thing, calling the process "spooky action at a distance."

The spooky particle data set introduced here includes the set of known events that a single particle has experienced in its universe throughout all time. If the particle is in fact a psychic genius, then it should instantly know some aspect of its final state given any initial state. The goal here is to teach the particle how to become psychic in some way by allowing it to learn from the data set generated by its surrounding universe. A simple example is the best way to demonstrate this.

Let the universe be 3 dimensional (not counting time as a dimension), void of gravity and other such forces, and consist of an empty rectangular box enclosing the volume spanned by $(0,0,0)$ to (l,w,h). Let the particle's initial state in its universe be given by its initial position and velocity in the box $[X_0, V_{x0}, Y_0, V_{y0}, Z_0, V_{z0}]$. Let its final state be given by when it collides with one of the inner walls of the box, and mark this state with T_i, or time until

impact from its initial state. If the particle is in fact psychic it should instantly know T_t given any initial state in that universe. Thus many examples can be generated using the specified laws of physics to obtain a data set having inputs $[X_0, V_{x0}, Y_0, V_{y0}, Z_0, V_{z0}]$ and desired output T_t. For this discussion, let the particle behave according to simple Newtonian rules of motion and let the rest of the universe be completely static.

The universe could contain internal (nonmoving) objects, have irregular boundaries, exist in a high number of dimensions, utilize gravity, etc. There are a wide variety of universes and corresponding data sets that can be generated which exhibit very interesting nonlinear properties. The dynamic particle only learns the laws of physics in its surrounding toy universe through example (by colliding with the walls or internal objects) and nothing else. It only becomes "spooky" after it learns how to predict T_t accurately based on previous known events.

The results presented in this paper are based on the simplified universe in a box described above with dimensions $2x5x1$, $V_{min} = 3$, and $V_{max} = 5$, with (X_0, Y_0, Z_0) uniformly distributed in the box and (V_{x0}, V_{y0}, V_{z0}) uniformly distributed in a spherical shell with an inner and outer radius of V_{min} and V_{max} respectively. Also, 1000 points were generated for the training set and 1000 for the test set. This is a challenging number to use for this problem because T_t is somewhat difficult for the base model to learn but can still be generalized in the test set past the 0.8 r^2 range.

Results

Every neural network in this analysis was trained with 1000 iterations using the scaled conjugate gradient algorithm. Tables I-IV show the RMS error and R^2 results of the models described above over various ANN architectures for the Mackey Glass data set. The KNN, GRNN, and ANN models were independently trained. The remainder of the methods, FWKNN, FWGRNN, LMLR, and NNA, were all dependent on the corresponding ANN through the calculation of the distance metric using scaled inputs based on $\vec{R}$ and $\vec{I}$. LMLR used the FWGRNN estimate for σ_G to locally weight each pattern. NNA stacked a second ANN on top of the FWGRNN, LMLR, and ANN models using $L(\vec{x})$ and d as extra inputs to help decide which local or global modeling approach to use in a particular case. Tables V-VIII, IX-XII, and XIII-XVI show the RMS error and R^2 results of the models over various ANN architectures for the chaos-13, spooky particle, and letter recognition data sets respectively. Table XVII shows a modeling comparison for all of the data sets used here. The results of Table XVII have been averaged (by test set R^2) over all of the ANN architectures used for each problem. The numbers represent a ranking (from 1 to 7) so that all of the information in the previous tables can be seen at once.

Refer to Figure 2. This shows the test set R^2 average performance (over all ANN architectures) of each model on the Mackey Glass data set. Figures 3-4 represent the same model comparison over the chaos-13 and spooky particle data sets. Figure 5 depicts the overall performance comparison between all of the methods on all of the problems over all of the ANN architectures used. The methods not using a weighted distance metric based on the trained ANN (KNN, FWKNN, and GRNN) are trailing the other methods by a large margin.

Hybrid approaches that combine the neural network with nearest neighbors (FWGRNN, LMLR, and NNA) tended to outperform ANN as a whole. However, only NNA tended to outperform the corresponding ANN at an optimal architecture.

Conclusions

Seven general data-driven methods were described for mapping nonlinear data, three of which are standard approaches (KNN, GRNN, and ANN). Four other approaches (FWKNN, FWGRNN, LMLR, and NNA) are described which utilize a weighted distance metric from the output scaling based on the DMP demonstrated by the corresponding trained ANN. They combine local weighting methods with the global regression capability of neural networks to allow for potentially improved accuracy at the expense of increased computation (compared to recalling a neural network). In general, over all trained architectures tested on the four problems described above, the resulting overall rank suggests the following:

$$\text{NNA} > \text{FWGRNN} > \text{ANN} > \text{LMLR} > \text{GRNN} > \text{FWKNN}(k=1) > \text{KNN}(k=1) \tag{24}$$

Combining the outputs of LMLR, GRNN, and ANN in a way similar to the stacking approach of NNA with an ANN-based weighted distance metric can improve generalization. Although in some cases an optimal architecture ANN can result in greater accuracy than a FWGRNN or LMLR, the NNA approach is general enough to reduce to the ANN global solution if a particular problem demonstrates this is the best thing to do.

References

Aha, D. (1997). *Lazy Learning.* Kluwer Academic Publishers.

Aha, D., Kibler, D., Albert, M. (1991). "Instance-based learning algorithms," *Machine Learning 6*, 37-66.

Atkeson, C. G. and S. Schaal (1995). "Memory-based neural networks for robot learning," *Neurocomputing*, Vol 9, 1-27.

Atkeson, C. G., A.W. Moore, and S. Schaal (1997). " Locally Weighted Learning," *Artificial Intelligence Review*, Vol 11, 11-73.

Bartlett, E. B. and A. Whitney (1999). "On The Use Of Various Input Subsets For Stacked Generalization," *Intelligent Engineering Systems Through Artificial Neural Networks*, Vol 9, 117-124.

Blum, E. K. and L. K. Li, (1991). "Approximation Theory and Feedforward Networks," *Neural Networks*, 4, 511.

Carmichael, C. (1997). "Experiments in advancing supervised-learning ANN techniques," Thesis(M.S), Iowa State University.

Cleveland, W. and C. Loader (1995). "Smoothing by Local Regression: Principles and Methods (with discussion)," Computational Statistics.

Cherkassky, S. and F. Mulier, (1998). *Learning From Data: Concepts, Theory, and Methods (Adaptive and Learning Systems for Signal Processing, Communications and Control Series).* John Wiley & Sons, New York, New York.

Cun, Y. Le, J. S. Denker, and S. A. Solla (1990). "Optimal brain damage," Advances in Neural Information Processing Systems 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 598-605.

Fan, J. (1995). "Local Modeling," *Encyclopedia of Statistical Science.*

Han, E., Karypis, G., Kumar, V. (1991). "Text Categorization Using Weight Adjusted k-Nearest Neighbor Classication," *Proceedings of The Twelfth International Joint Conference on Artificial Intelligence.*

Hecht-Nielsen, R. (1990). *Neurocomputing.* Addison-Wesley, Reading, Mass.

Lippmann, R. P., (1987). "An Introduction to Computing with Neural Nets," *IEEE Acoustics Speech and Signal Processing Magazine*, 4, 4.

Mackey, M. C. and L. Glass (1977). "Oscillations and Chaos in Physiological Control Systems," *Science*, 197, 287-289.

Mandilwar, D. K. and H. Qammar (1993). "Prediction of Chaotic Time Series: Neural Versus Fuzzy", *Proceedings of the 1993 International Fuzzy Systems and Intelligent Control Conference,* 189-199.

Miller, W. T., R. S. Sutton, and P. Werbos (Eds.), (1990). *Neural Networks for Control.* Press. Cambridge, Mass.

Mitchell, T., (1997). *Machine Learning.* McGraw-Hill, New York, New York.

M. C. Mozer and P. Smolensky (1989). "Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in Neural Information Processing Systems, 1, D. S. Touretzky , Ed. (Denver 1988), pp. 107-115.

Nadaraya, E. A. (1964). "On estimating regression," *Theory of Probability and its Applications 9*, 141-142.

Narendra, K. S. and K. Parthasarathy, (1990). "Identification and Control of Dynamic Systems Using Neural Networks," *IEEE Trans. Neural Networks,* 1, no. 1, 4.

Parzen, E. (1962). "On estimation of a probability density function and mode," *Annals Mathematical Statistics 33*, 1065-1076.

Peitgen, H.-O. and P. H. Richter (1986). *The Beauty of Fractals: Images of Complex Dynamical Systems*. Springer-Verlag, New York, New York.

Rosenblatt, M. (1956). "Remarks on some nonparametric estimates of a density function," *Annals Mathematical Statistics 27*, 832-837.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, (1986). *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol. 1 & 2, MIT Press, Cambridge, Massachusetts.

Russell, S. and P. Norvig, (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey.

Schioler, H. and U. Hartmann (1992). "Mapping Neural Network Derived from the Parzen Window Estimator," *Neural Networks*, 5, 903 - 909.

Schmitz, G. P. J. and C. Aldrich (1999). "Combinatorial Evolution of Regression nodes in Feedforward Neural Networks," *Neural Networks,* 12, 175-189.

Specht, D. F., (1991). "A General Regression Neural Network," *I&E Transactions on Neural Networks,* 2(6), 568-576.

Upadhyaya, B. R., and E. Eryurek, (1992). "Application of neural networks for sensor validation and plant monitoring," *Nuclear Technology* 97, 170-176.

Venkatasubramanian, V., and K. Chan, (1989). "A neural network methodology for process methods," *Journal of Applied Meteorology* 31, 405-420.

Watson, G. S. (1964). "Smooth Regression Analysis," *Sankhya – The Indian Journal of Statistics 26,* 359-372.

Wichmann, N. L. and E. B. Bartlett (1997), "Ranking Input Variables using General Regression Neural Networks", *Intelligent Engineering Systems Through Artificial Neural Networks: Volume 7*, pp. 959 – 964.

Wichmann, N. L., (1997). "Variable Importance Ordering for Evapotranspiration using General Regression Neural Networks", *M.S. Thesis,* Iowa State University, Ames Iowa.

Tables and figures

Table I. Model results for the Mackey Glass data set using a 4x5x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.12226	0.733664
GRNN	0.0452069	0.967993	0.0956164	0.837585
ANN (4x5x1)	0.10439	0.815366	0.130573	0.674885
FWKNN (k=1)	0	1	0.125643	0.715279
FWGRNN	0.0424963	0.971438	0.0934421	0.844206
LMLR	0.0236535	0.992266	0.0885214	0.856794
NNA	0.163588	0.908858	0.105131	0.865892

Table II. Model results for the Mackey Glass data set using a 4x10x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.12226	0.733664
GRNN	0.0452069	0.967993	0.0956164	0.837585
ANN (4x10x1)	0.076649	0.900479	0.109252	0.775292
FWKNN (k=1)	0	1	0.122467	0.73312
FWGRNN	0.044903	0.96832	0.0939729	0.842581
LMLR	0.02571	0.990886	0.0865376	0.86268
NNA	0.0563754	0.946646	0.0807134	0.877557

Table III. Model results for the Mackey Glass data set using a 4x15x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.12226	0.733664
GRNN	0.0452069	0.967993	0.0956164	0.837585
ANN (4x15x1)	0.058155	0.942708	0.0937792	0.834942
FWKNN (k=1)	0	1	0.124937	0.720967
FWGRNN	0.0446563	0.96867	0.0933022	0.84394
LMLR	0.0254065	0.991141	0.0853495	0.866755
NNA	0.0746109	0.906533	0.0895922	0.856173

Table IV. Model results for the Mackey Glass data set using a 4x20x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.12226	0.733664
GRNN	0.0452069	0.967993	0.0956164	0.837585
ANN (4x20x1)	0.0520089	0.954187	0.089492	0.848283
FWKNN (k=1)	0	1	0.122575	0.731166
FWGRNN	0.0448006	0.968621	0.0946378	0.839878
LMLR	0.0259024	0.990789	0.0864533	0.862796
NNA	0.117634	0.938383	0.11589	0.83948

Table V. Model results for the chaos-13 data set using a 13x5x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.0583179	0.74981
GRNN	0.0154028	0.983029	0.0477103	0.82115
ANN (13x5x1)	0.0216319	0.963422	0.0211102	0.965011
FWKNN (k=1)	0	1	0.0232921	0.958182
FWGRNN	0.00696676	0.996321	0.0175863	0.975756
LMLR	0.000176662	0.999998	0.0171029	0.977068
NNA	0.00595919	0.997375	0.0101756	0.9919

Table VI. Model results for the chaos-13 data set using a 13x10x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.0583179	0.74981
GRNN	0.0154028	0.983029	0.0477103	0.82115
ANN (13x10x1)	0.0129933	0.986805	0.0137185	0.98515
FWKNN (k=1)	0	1	0.0222636	0.961701
FWGRNN	0.0067046	0.996592	0.0168126	0.977826
LMLR	0.000158176	0.999998	0.0203487	0.96812
NNA	0.00693586	0.996299	0.00992877	0.99225

Table VII. Model results for the chaos-13 data set using a 13x15x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.0583179	0.74981
GRNN	0.0154028	0.983029	0.0477103	0.82115
ANN (13x15x1)	0.011414	0.989825	0.0127839	0.987144
FWKNN (k=1)	0	1	0.0212754	0.965122
FWGRNN	0.00646319	0.996826	0.0157299	0.980575
LMLR	0.000177657	0.999998	0.0275247	0.941528
NNA	0.00588981	0.997429	0.00898582	0.993624

Table VIII. Model results for the chaos-13 data set using a 13x20x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.0583179	0.74981
GRNN	0.0154028	0.983029	0.0477103	0.82115
ANN (13x20x1)	0.0108448	0.990811	0.0129216	0.986878
FWKNN (k=1)	0	1	0.0214441	0.964614
FWGRNN	0.00626806	0.997008	0.0157134	0.980662
LMLR	0.000166661	0.999998	0.0173245	0.976444
NNA	0.00663106	0.997384	0.00897883	0.993669

Table IX. Model results for the spooky particle data set using a 6x5x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.136231	0.505311
GRNN	0.0452492	0.958812	0.0968344	0.659622
ANN (6x5x1)	0.100314	0.714426	0.0921829	0.696405
FWKNN (k=1)	0	1	0.110914	0.592064
FWGRNN	0.0237302	0.986105	0.0814946	0.754214
LMLR	0.00679051	0.998937	0.0792638	0.774792
NNA	0.0993977	0.906781	0.0863518	0.733372

Table X. Model results for the spooky particle data set using a 6x10x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.136231	0.505311
GRNN	0.0452492	0.958812	0.0968344	0.659622
ANN (6x10x1)	0.0668997	0.872959	0.0703212	0.821239
FWKNN (k=1)	0	1	0.111225	0.605889
FWGRNN	0.0448778	0.959946	0.0803335	0.772508
LMLR	0.0220942	0.990919	0.0723662	0.821905
NNA	0.0550367	0.946498	0.063235	0.856851

Table XI. Model results for the spooky particle data set using a 6x15x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.136231	0.505311
GRNN	0.0452492	0.958812	0.0968344	0.659622
ANN (6x15x1)	0.05937	0.899962	0.067733	0.837427
FWKNN (k=1)	0	1	0.109513	0.60647
FWGRNN	0.0460369	0.957309	0.0803549	0.771664
LMLR	0.0224759	0.990537	0.072104	0.822881
NNA	0.0595949	0.916784	0.0678738	0.831598

Table XII. Model results for the spooky particle data set using a 6x15x3x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.136231	0.505311
GRNN	0.0452492	0.958812	0.0968344	0.659622
ANN (6x15x3x1)	0.0431663	0.947017	0.0564924	0.883948
FWKNN (k=1)	0	1	0.104523	0.635512
FWGRNN	0.0457485	0.957699	0.0793102	0.777233
LMLR	0.0221615	0.990772	0.0732146	0.815017
NNA	0.0356954	0.964445	0.063961	0.860801

Table XIII. Model results for the letter recognition data set using a 16x10x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.027386	0.980039
GRNN	0.000516735	0.999993	0.022928	0.986014
ANN (16x10x1)	0.0475129	0.942623	0.0640565	0.893993
FWKNN (k=1)	0	1	0.027386	0.980039
FWGRNN	0.0000271	1	0.024669	0.983764
LMLR	0.000002	1	0.063703	0.896385
NNA	0.000583	1	0.030739	0.974819

Table XIV. Model results for the letter recognition data set using a 16x20x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.027386	0.980039
GRNN	0.000516735	0.999993	0.022928	0.986014
ANN (16x20x1)	0.0401445	0.958504	0.0523863	0.928103
FWKNN (k=1)	0	1	0.027386	0.980039
FWGRNN	0.000546597	0.999992	0.0221014	0.986995
LMLR	0.000052	1	0.050114	0.933714
NNA	0.000094	1	0.026118	0.981815

Table XV. Model results for the letter recognition data set using a 16x30x1 ANN

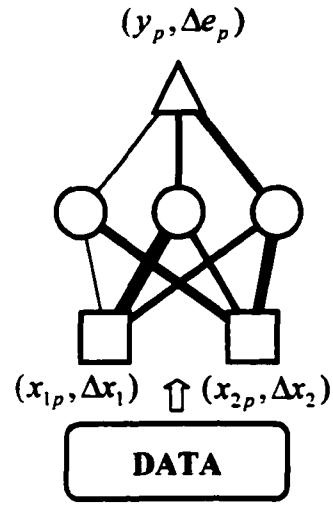
Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.0273861	0.980039
GRNN	0.000516735	0.999993	0.022928	0.986014
ANN (16x30x1)	0.0365695	0.96549	0.0482026	0.938887
FWKNN (k=1)	0	1	0.0316228	0.9737
FWGRNN	0.000528594	0.999993	0.0201109	0.989236
LMLR	0.000051	1	0.049195	0.936274
NNA	0.000696	0.999988	0.022089	0.986991

Table XVI. Model results for the letter recognition data set using a 16x20x10x1 ANN

Model	Train RMS	Train R^2	Test RMS	Test R^2
KNN (k=1)	0	1	0.0273861	0.980039
GRNN	0.000695283	0.999988	0.022928	0.986014
ANN (16x20x10x1)	0.0041231	0.99956	0.0330517	0.970926
FWKNN (k=1)	0	1	0.0273861	0.980004
FWGRNN	0.000471386	0.999994	0.0234176	0.985431
LMLR	0.000045	1	0.043097	0.950575
NNA	0.000129	1	0.022234	0.986929

Table XVII. Modeling method comparison for various data sets

Modeling Method	Mackey Glass	Chaos-13	Spooky particle	Letter recognition	**Average rank**
KNN (k=1)	6	7	7	4	**6**
GRNN	4	6	5	2	**4.25**
ANN	5	2	2	6	**3.75**
FWKNN (k=1)	7	5	6	5	**5.75**
FWGRNN	3	3	4	1	**2.75**
LMLR	1	4	3	7	**3.75**
NNA	2	1	1	3	**1.75**



$$\frac{\delta e}{\delta x_i}(p) = \lim_{\Delta x_i \to 0} \frac{\Delta e}{\Delta x_i}(p)$$

Figure 1. Input scaling by sensitivity analysis

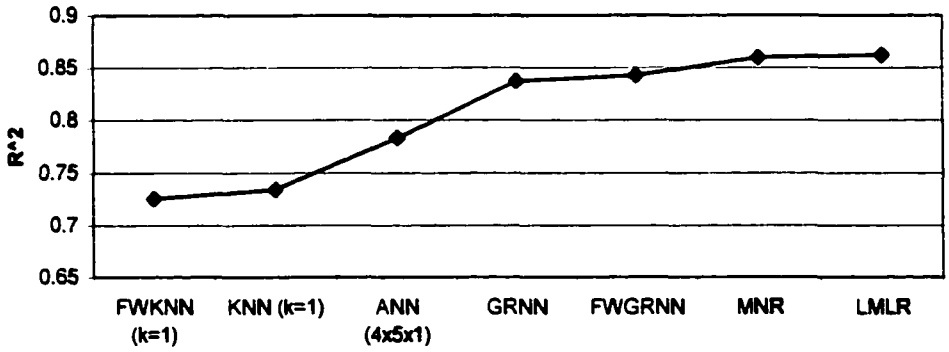


Figure 2. Model comparison for the Mackey Glass data set

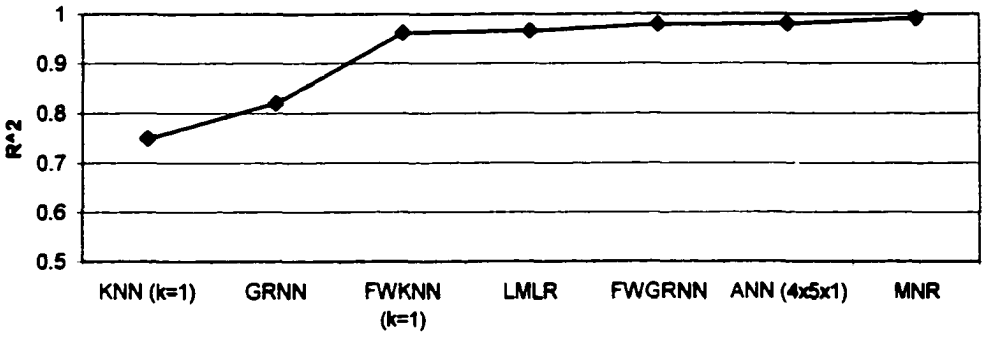


Figure 3. Model comparison for the Chaos-13 data set

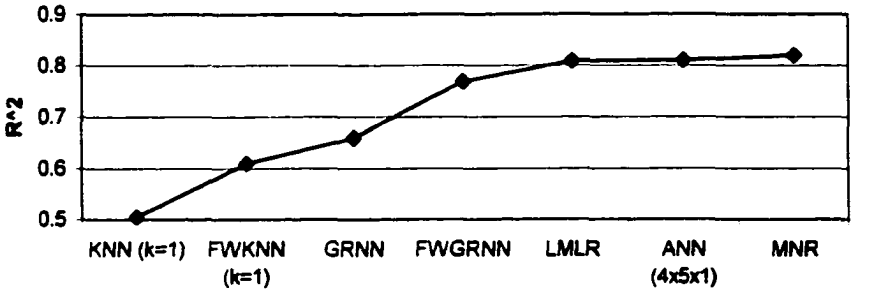


Figure 4. Model comparison for the spooky particle data set

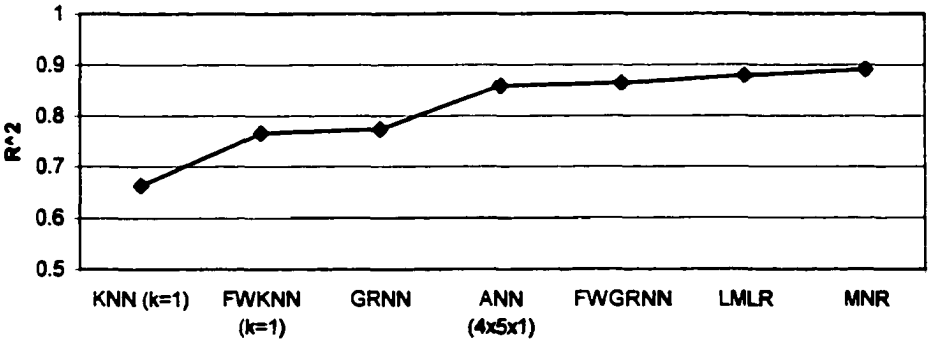


Figure 5. Averaged model comparison for all data

Nomenclature

σ_G	Free parameter optimized by a GRNN
$\sigma_m(\vec{x})$	Modeled standard deviation in the area of $\vec{x}$
$\vec{A}$	Free parameters for multivariate linear regression and local MLR
ANN	Artificial neural network
d	Absolute difference (or discrepancy) of the output between two models
D	Distance metric
DMP	Decision making power of an element of a neural network
e_p	Error component of the batch training error for pattern p
E	Batch training error
FWGRNN	Feature weighted GRNN
FWKNN	Feature weighted KNN
GRNN	General regression neural network
$\vec{I}$	DMP input importance vector (after normalization)
j	Input subscript
KNN	k-Nearest Neighbors algorithm
$L(\vec{x})$	Local training data density of the space surrounding $\vec{x}$
LMLR	Localized multivariate linear regression
MLR	Multivariate linear regression
NNA	Neural Neighbors Algorithm
n	Number of training patterns (same as *Ntrain*)
N	Number of inputs

Ntrain	Number of training patterns (same as n)
p	Training data pattern subscript
PRESS	GRNN predicted error sum of squares
$\vec{R}$	Relevance or raw DMP input importance vector (before normalization)
T_p	Target output for pattern p
v_p	GRNN local weighting factor for pattern p (same as w_p)
w_p	GRNN local weighting factor for pattern p (same as v_p)
WDM	Weighted distance metric
$\vec{x}$	Input space
y	Output, generic description
$y \mid \vec{x}$	Theoretical behavior of y in the area of $\vec{x}$

CHAPTER 5. STACKING DIVERSE MODELS TO ACHIEVE RELIABLE ERROR RESPONSE DISTRIBUTIONS

A paper submitted to the International Journal of Smart Engineering System Design

Craig G. Carmichael and Eric B. Bartlett

Department of Electrical and Computer Engineering,

Iowa State University, Ames, IA, 50011

Abstract

Artificial Neural Networks (ANNs) can be useful for modeling real - world processes such as time series weather, financial, or chaotic data. The generalization and robustness of these models can be improved and estimates of the modeling error distributions can be made using a technique called Stacked Generalization (SG). SG uses a number of diverse models, each of which is trained and queried on independent cross validation subsets of the process data. The models are then combined in the stacking process to provide error estimates and improved accuracy. These improvements depend on the individual model response diversity between networks. Modified Series Association (MSA), an extension to SG, presents the various models with different input subspaces from the raw data as a catalyst for increased diversity. Model diversity is formulated, and an alternative model combination approach is derived from it, called Diversified Committee Machines (DCM). A framework for quantifying error estimation reliability is presented and discussed. Using this framework, the predictive accuracy of SG and DCM are compared in terms of both the modeled target

function and the model's confidence interval about it. This is achieved through a new measure called the confidence coefficient. A benchmark problem is also introduced as a generic data set for future comparison between inductive learning machines.

Keywords

Artificial Neural Network, Diversity, Feedforward Networks, Hierarchical Neural Networks, Data Modeling, Prediction Error

Introduction

The Inductive Learning Hypothesis (ILH) provides a backbone for all inductive learning machines: Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples. Yet, there is ongoing debate regarding the best way to learn and compare hypotheses from limited data [Mitchell, 1997]. In practice, sufficiently large data sets to one observer can seem limited to another. Therefore, regardless of the size of the data at which ILH breaks down and the debate begins, all inductive learning machines should attempt to not only approximate the target function but also assess the uncertainty in the approximation with some probability distribution or confidence interval.

Inductive learners such as feedforward neural networks do not directly provide probability distributions on the output values. With decision trees and other logical representations, the output can be explained as a logical derivation and by appeal to a specific set of cases that support the decision. This has been an elusive task for neural networks [Russell and Norvig, 1995]. Without a way to probe the networks for these supportive cases,

a confidence interval on the network cannot be accurately constructed. However, a neural network's output can in fact be explained in terms of a specific set of cases that support the network's decision, as described in the next section. With this information, a confidence interval can be more accurately constructed to estimate the uncertainty in the output in relation to the supportive cases so that:

$$P(\text{decision error} < \text{bound} \mid \text{supportive cases of decision}) \neq P(\text{decision error} < \text{bound} \ \forall \ \text{cases}) \tag{1}$$

Supportive cases of a network's decision

A neural network's output is mapped nonlinearly from the input space represented by the training data set. The supportive cases of a network's decision can be estimated from the input space of the most relevant cases supplied by the training set. Inductive techniques such as local basis function networks and nearest neighbor approaches determine this relevance through some sort of a kernel function [Cherkassky and Mulier, 1998]. Although neural networks use a different form of basis function than local basis function networks, the most relevant cases can still be approximated in roughly the same local manner revealed by a kernel function. After training a neural network over some representative data set, the network can be probed for its assessment of each input. A sensitivity analysis of the output with respect to each of the inputs over all cases in the training set can then estimate the information content provided by each input in the data set to the trained neural network. An

input with higher overall sensitivity across all training cases should generally have higher relevance in supporting a network's decision. This relevance should be apparent through a distance metric that recognizes an imbalance of input importance. Call this metric a weighted distance metric, WDM.

The input space corresponding to a network's output (decision) can be compared to the input space of each example (case) in the training set through some WDM variant. The WDM can then be fed to a standard kernel function that is nonnegative, radially symmetric, monotonically increasing with locality, and approaching zero at large distances. The WDM kernel function can then directly reveal an approximation for how supportive each training example is in reaching the decision of the neural network. Therefore, with some appropriate distance metric, a confidence interval can be constructed on the network's decision in a fashion that utilizes the supportive cases from the data set that generated the decision. The model combination and error estimation techniques presented here rely on some WDM variant.

Model combination and error estimation

ANNs can be regarded as generalizers because they infer parent functions from sets of data [Cybenko, 1989; Wolpert, 1990; Kurkova, 1992]. Most other modeling methods can also be considered to be generalizers as well. For example, statistical and even first principle methods can be considered as generalizers. Therefore, the following discussion can be applied to a large class of modeling methods. It is, however, difficult to find methods for error estimation and consolidation of general data driven nonlinear modeling techniques such as ANNs, and this is where the MSA technique, an extension to SG, can be used to full effect

[Narendra and Parthasarathy, 1990; Blum & Li, 1991; Bartlett, 1992 & 1994; Bartlett & Kim, 1993].

Stacked Generalization was proposed as a method of using multiple models to provide improved accuracy or confidence intervals [Wolpert, 1992]. Wolpert suggests using a number of models, in this case ANNs, on different subsets of the data, in order to obtain models that are slightly different. These models are then recalled over the remainder of the data and this information is used in the stacking process. Modified Series Association goes an additional step further to obtain model diversity. MSA allows the models to have different input sub-vectors of the input space. For the remainder of the discussion, the specific form of SG implemented here is MSA [Bartlett and Whitney, 1999].

The information generated by querying these various networks on their respective cross validation sets constitutes new information, since they are not expected to produce exactly identical results. This information is then used to develop the error predictor and consolidation models. SG is a method of developing a stacked model that utilizes the results of two levels of multiple networks or generalizers. The models in the first level, called the Level 0 networks, are trained not only on partitions of the given data set, but also subspaces input vector. The Level 0 models are then queried, or recalled, on the unused, or cross validation, data. The results of the recall of the Level 0 models on these "novel" cross validation partitions are then stored. The second level of models, called the Level 1 models, are then developed using the results of this Level 0 cross validation recall. These Level 1 models provide the consolidated, or stacked, model output and the corresponding errors for each prediction. These Level 1 results are based on the diverse behavior of the Level 0 models. Once the Level 1 models are created with the Level 0 cross validation data sets, then

these Level 0 models are discarded. New Level 0 models are then developed on the complete training data set without partition using their appropriate input subspace and some WDM variant. These new networks are then used in the stacked recall process to generate consolidated predictions and their associated uncertainties. This approach has been shown to be effective in conducting error analyses on ANN models for pattern recognition [Kim and Bartlett, 1996] and to provide improved functional models of the desired outputs [Sridhar, 1996]. When properly combined, these various models yield reliable error estimates for ANN function approximation, as will be shown in this paper.

Reliable error estimates

What does it mean to say the word "reliable" when talking about modeled error estimates? The simplest approach for estimating errors is to report twice the standard deviation of the $E[y \mid \vec{x}]$ model's residuals over some cross validation set as the expected 95^{th} percent confidence interval for every new instance of $\vec{x}$. Is this "reliable?" Yes, it is reliable. But is it really a model? Not really. This is equivalent to the unnecessarily simplified confidence interval described above.

So what is a model? A convincing error estimation model should exhibit a positive correlation coefficient between the absolute value of the actual errors and the modeled errors on some arbitrary test set provided that the modeled problem isn't impossible. So, the question now is "What is the upper limit of performance for the error estimation model, and how can it be measured?" Without an answer to this question, robust claims about the error model's performance cannot be made. A comparison to the best possible model is required to clearly define the reliability of the model.

Best possible error estimation model

Assume that the actual residual on the base model, e_i, is Gaussian for each new pattern i where the standard deviation $\sigma_{m,i}$, is the output for each pattern of the Gaussian error estimation model. Farther, assume that the output of the error prediction model yields error estimates in the range from σ_L to σ_H where the subscripts L and H stand for low and high respectively. The residual e_i will fall within $-\sigma_{m,i} < e_i < +\sigma_{m,i}$ value 67% of the time for the best possible model. Because $\sigma_{m,i}$ is bounded in this case to σ_L and σ_H, the point $(\sigma_{m,i}, e_i)$ will fall, 67% of the time, within the heavy dotted line of Figure 1 and be bounded by the region:

$$\{(\sigma_L,\sigma_L),(\sigma_H,\sigma_H),(\sigma_H,-\sigma_H),(\sigma_L,-\sigma_L)\} \tag{2}$$

Also, 95% of the time the point will fall within the light dotted line bounded by

$$\{(\sigma_L,2\sigma_L),(\sigma_H,2\sigma_H),(\sigma_H,-2\sigma_H),(\sigma_L,-2\sigma_L)\} \tag{3}$$

Now consider the plot of the error $|e|$ vs. σ_m in Figure 3. Here, 95% of the time the point $(\sigma_{m,i}, |e_i|)$ will fall within the light dotted line bounded by

$$\{(\sigma_L,2\sigma_L),(\sigma_H,2\sigma_H),(\sigma_H,0),(\sigma_L,0)\} \tag{4}$$

The important difference between Figure 1 and Figure 2 is that the correlation coefficient r based on the generated points in Figure 1 will be zero and in Figure 2 it will be greater than or equal to zero. With enough i points in the calculation of the coefficient based on a plot like Figure 2, the only way that this r, referred to as r_e below, will be equal to zero is if σ_m is held constant for each point by the constraint $\sigma_L = \sigma_H$. So what can be expected about the model's calculated value for r_e? To answer this question, introduce the stochastic ratio, s, defined by

$$s = \frac{\sigma_L}{\sigma_H - \sigma_L} \tag{5}$$

Referring to Figure 2, imagine what happens as s goes from infinity towards zero. By holding σ_H (the high limit of σ_m) constant and varying σ_L (the low limit of σ_m) from σ_H to 0, this can be demonstrated. The correlation coefficient r_e will increase monotonically from zero to some upper limit $r_{e,\max}$. Therefore, it would be desirable for σ_L to approach (but not necessarily reach) 0 so that a high value for the correlation coefficient, r_H, can be attained that will allow r_e to approach $r_{e,\max}$.

The ratio s sheds light on the stochastic nature of the modeled data set. It is problem dependent. Therefore, different problems will have different values for s and hence different values for $r_H = r_H(s)$. A small but significant measurement error in the data set's output, for example, could result in something like $\sigma_L = 0.1\sigma_H$ which would render $0 < r_H < r_{e,\max}$ on an otherwise completely deterministic problem. Now there is a reference point to compare

the error estimation models, dictated by the specific problem that designates the value for s. After the real world r_e is computed from $(\sigma_{m,i}, |e_i|)\ \forall i$, the following equation for the confidence coefficient, r_σ, describes the error model's performance.

$$r_\sigma = \frac{r_e}{r_H} \tag{6}$$

In addition to this measure of performance, the confidence interval of the model is scaled properly so that approximately 95% of the time the errors $|e_i|$ are bounded by 2 times the modeled sigma, $\sigma_{m,i}$, over some cross-validation set. This can be achieved, if necessary, by multiplying each future σ_m by an amount that produces the desired confidence level in the validation set. The difficulty tends to be creating a composite model that attains the highest r_e compared to r_H which brings r_σ close to 1.

The value of r_H can be calculated from the estimate of s which in turn yields an estimate to the error model's performance given by r_σ. First, however, the distribution of σ_m throughout the range of σ_L to σ_H should be described more precisely. A uniform distribution $U(\sigma_L, \sigma_H)$ is straightforward, but this kind of distribution is not generally expected. Therefore the span between the high and low "limits" of σ_m can be replaced with a 95% confidence interval having mean $\bar{\sigma}_m = \frac{1}{2}(\sigma_L + \sigma_H)$ so that this Gaussian distribution has a mean and standard deviation defined by $G(\bar{\sigma}_m, \frac{1}{4}(\sigma_H - \sigma_L))$, making the appropriate adjustments so that σ_m is never negative. Now the effect of s can be demonstrated on the

high limit of r_e. Figure 3 shows $r_H(s)$ for both the uniform and Gaussian case. Although a gamma distribution for σ_m would be more versatile than a uniform or Gaussian distribution, it is not included in the analysis presented here.

The following equations fit the curves shown in Figure 3 using 100,000 trials for each point plotted for σ_m, describing both the uniform and Gaussian cases:

$$r_{H,G} = \frac{2}{5s+4} - 0.01 \tag{7}$$

$$r_{H,U} = \frac{r_{H,G}}{0.88} \tag{8}$$

$$r_H(s) = \begin{cases} r_{H,U}, & \textit{uniform } \sigma_m \\ r_{H,G}, & \textit{gaussian } \sigma_m \end{cases} \tag{9}$$

Therefore, the following equation holds for purely deterministic problems since $s = 0$ in this case.

$$r_{e,\max} = r_H(s \mid s = 0) = \begin{cases} 0.56, & \textit{uniform } \sigma_m \\ 0.49, & \textit{gaussian } \sigma_m \end{cases} \tag{10}$$

If the problem is known to be completely deterministic with respect to the data set's input space, then $s = \sigma_L = 0$. Ideally, the best possible error estimation model will be achieved

with $r_\sigma = 1$ and $r_H = r_{e,\max}$, provided that the set is large enough for the calculated r on the error estimate (r_e) to be statistically significant. Adding a small amount of noise σ_L to the output of the otherwise deterministic data set will still allow $r_\sigma = 1$ but with $r_H < r_{e,\max}$ since $s > 0$. The best possible error estimation model in this case is limited because $\sigma_L > 0$.

Modeled error low limit determination

It is important to determine the lower limit on σ_m adequately in order to assess the calculated r_e compared to the best possible error estimation model through r_σ. An estimate for σ_H could be obtained by viewing the distribution of $|e|$, but σ_L may be much more challenging to estimate. A problem does not have to be entirely deterministic for s and σ_L to be relatively small compared to σ_H. Only occasionally does the problem have to exhibit deterministic behavior for this to occur. Since the estimate of r_σ is intended to be reliable, so is the estimate for σ_L.

Confidence in the estimate for σ_L is established if it is known from the nature of the problem itself. The "Spooky Particle Data Set", described below, is based on a completely causal imaginary universe. Therefore, $\sigma_L = 0$ for this problem because the data set is based on deterministic rules. Therefore confidence can be established in the assessment of the $E[y \mid \vec{x}]$ model's predicted errors by using r_σ, which is based on a careful estimate of s.

In practice, where the data set is not always artificially generated and σ_L is not always immediately apparent, the estimates of σ_L and the σ_m distribution are critical. The

analysis is simplified by assuming a uniform or Gaussian distribution on σ_m so σ_L can be estimated.

A first estimate of σ_L and $\sigma_{m,i}$ can be determined for all i using a modification to the standard general regression neural network, GRNN, models. For a GRNN, the expected (modeled) value(s) of the output(s) given the input(s) are determined by:

$$E[y \mid \vec{x}] = \frac{\int_{-\infty}^{\infty} y \cdot f(\vec{x}, y)\, dy}{\int_{-\infty}^{\infty} f(\vec{x}, y)\, dy} \tag{11}$$

The above representation for a GRNN that models a finite set of points can be reduced to the following discrete formula for an estimation of $E[y \mid \vec{x}]$.

$$E[y \mid \vec{x}] \approx y(\vec{x}) = \frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i} \tag{12}$$

Where w_i is defined as

$$w_i = \exp\left(\frac{-D_i^2}{2\sigma_p^2}\right) \tag{13}$$

Also, D_i^2 is the Euclidean distance metric between the current $\vec{x}$ and that of pattern i in the training set, defined as

$$D_i^2 = (\vec{x} - \vec{x}_i)^T (\vec{x} - \vec{x}_i) \tag{14}$$

The single free parameter σ_p is determined by minimizing the predicted error sum of squares (PRESS) in an N-folding process. The GRNN approach is modified here to include the $E[y^2 \mid \vec{x}]$ term as part of the structure for simplicity and to allow the use of the same σ_p for both the calculation of $E[y \mid \vec{x}]$ and $E[y^2 \mid \vec{x}]$. Then $\sigma_m = \sigma[y \mid \vec{x}]$ is estimated as follows:

$$\sigma[y \mid \vec{x}] = \left(E[y^2 \mid \vec{x}] - E^2[y \mid \vec{x}]\right)^{1/2} \tag{15}$$

$$E[y^2 \mid \vec{x}] \approx y_2(\vec{x}) = \frac{\sum_{i=1}^{n} w_i y_i^2}{\sum_{i=1}^{n} w_i} \tag{16}$$

$$\sigma[y \mid \vec{x}] \approx \sigma_m(\vec{x}) = \left(y_2(\vec{x}) - y^2(\vec{x})\right)^{1/2} \tag{17}$$

From this modeled distribution for σ_m, an upper bound for σ_L can be estimated as the minimum point of $\sigma_{m,i}$. If σ_m provides good values for r_σ (given σ_L) then it may not be necessary to remodel σ_m with more sophisticated methods, especially if σ_L yields a relatively low value for s.

As the problem of generating the distribution of σ_m becomes more difficult, where, for example, the input space is large or the available data is reduced, the relatively standard GRNN approach mentioned above starts to become increasingly inaccurate. In such cases, a good first step is to improve the GRNN algorithm even more by adjusting the distance metric in the GRNN model. To accomplish this, first train a neural network on the data. Next, probe the neural network in an attempt to find the inputs that are most important to it. Delete each input, substituting the input with its average value, calculating the increase in error, and storing the result. Next, normalize this result for each input so that the sum of the normalized results adds to 1, and let the set of normalized results comprise an "importance" vector $\bar{I}$. Finally, adjust the distance metric in the GRNN model so that it is no longer Euclidean but is based on this importance vector. That is, weigh the important dimensions (to the neural net) more heavily based on the corresponding elements in the importance vector. Substitute this adjusted distance metric for D_i^2 back into the modified GRNN from above and recalculate the estimates for $E[y \mid \vec{x}]$, $E[y \mid \vec{x}]$, and $\sigma[y \mid \vec{x}]$. Experience with this technique has shown that it will generally improve the accuracy of these predictions and provide better estimates for σ_L, σ_H and consequently s.

Now consider a problem that is even more sparsely sampled. What happens to the above error estimation approach? The growing inaccuracies in this case arises from the GRNN's reliance on local estimates for both $E[y \mid \vec{x}]$ and $E[y^2 \mid \vec{x}]$. In truly challenging problems where data is sparse, a neural network can often outperform a GRNN in its estimate for $E[y \mid \vec{x}]$ even with the more accurate distance metric described previously. However, the main goal here is to determine an upper bound on σ_L so that s can be approximated,

followed by an approximation for $r_H(s)$. If this turns out to be a poor estimate, it is worth noting that a poor estimate is more informative than no estimate. When the discussion returns to stacked generalization below, more accurate error estimate approaches will be investigated for more challenging sparse data sets.

A generated faith in the confidence coefficient

The confidence coefficient r_σ pertaining to error modeling should perhaps be developed and investigated further before it can be used as a standard in measuring error estimation performance. A straightforward approach to develop confidence in this coefficient is to write a relatively short program to generate the best possible error estimation model assuming Gaussian residuals on the $E[y \mid \vec{x}]$ base model and to show that r_σ approaches 1 as the number of generated points becomes very large. Figure 3 was generated in this way. Notice the calculated $r(s)$ (on a plot like that of Figure 2) becomes very nearly $r_H(s)$ for each s using 100,000 trial points.

To generate a curve like that of Figure 3, first choose the type of distribution for the modeled errors, σ_m. Select a uniform or Gaussian distribution. Then arbitrarily assign an average for the modeled errors, $\bar{\sigma}_m$. A reasonable choice is to calculate the standard deviation of the $E[y \mid \vec{x}]$ model's residuals over some test set representing the current problem and use this as $\bar{\sigma}_m$. Next, choose the high limit of the correlation coefficient desired so that $0 < r_H < r_{\epsilon,\max}$. A point similar to those shown in Figure 2 can now be generated. After the following algorithm is coded, the calculated r_ϵ should be very close to the desired

r_H since the points are generated based on the best possible error estimation model. Given r_H, the stochastic ratio, s, can be determined by reversing the order of the equations above:

$$s = \begin{cases} \dfrac{0.4}{0.88 r_H + 0.01} - 0.8, & \textit{uniform } \sigma_m \\ \dfrac{0.4}{r_H + 0.01} - 0.8, & \textit{gaussian } \sigma_m \end{cases} \tag{18}$$

Given the original equation for $s = s(\sigma_L, \sigma_H)$ and the fact that $\bar{\sigma}_m = \frac{1}{2}(\sigma_L + \sigma_H)$, determine:

$$\sigma_L = \frac{2s\bar{\sigma}_m}{2s+1} \tag{19}$$

$$\sigma_H = \frac{\sigma_L(s+1)}{s} \tag{20}$$

Now generate about 100,000 artificial patterns of $(\sigma_{m,i}, |e_i|)$ using the perfect error estimation model:

$$\sigma_{m,i} = \begin{cases} U(\sigma_L, \sigma_H), & \textit{uniform } \sigma_m \\ G(\bar{\sigma}_m, \frac{1}{4}(\sigma_H - \sigma_L)), & \textit{gaussian } \sigma_m \end{cases} \tag{21}$$

If $\sigma_{m,i}$ is less than zero, which can happen about 2.5% of the time in the Gaussian case when σ_L is near zero, then regenerate $\sigma_{m,i}$ until this doesn't happen. The other option is to truncate it at zero in such a case. However, this will slightly change $s(r_H)$ with s close to 0. Finally, the (randomized) residual can be generated in a way that should emit from the perfect model's estimate of $\sigma_{m,i}$:

$$e_i = G(0, \sigma_{m,i}) \tag{22}$$

Numerous points of $(\sigma_{m,i}, |e_i|)$ can be produced and graphed in this way, given r_H and $\bar{\sigma}_m$. The actual correlation coefficient, r_e, can be computed from this generated set. With a sufficient number of points, the condition $r_e \approx r_H$ holds. The best possible model has now been confirmed to achieve the expected result of $r_\sigma = r_e / r_H = 1$, with a r_H value that is limited by s. In reality, r_σ will be less than 1, and this will show up through the calculated r_e in the error estimation model. For an even better estimate for r_H, it might be useful to start with the error estimation model for each $\sigma_{m,i}$ and to assign $e_i = G(0, \sigma_{m,i})$. By repeating this procedure, the high limit for this "true" distribution, $r_{H,T}$, can be obtained. In this way, a uniform or Gaussian assumption on the distribution for σ_m is unnecessary. However, the purpose here is to introduce a standard for performance that is relatively simple to verify through straightforward experimentation. The uniform and Gaussian distributions for σ_m are useful in this respect.

Error splitting

The quantity of $y \mid \vec{x}$ becomes uncertain if it is strictly interpreted as "y given $\vec{x}$", since there may be no samples in the data set at precisely $\vec{x}$. Therefore, to estimate $E[y \mid \vec{x}]$ and $Var[y \mid \vec{x}]$ it may be useful to describe $y \mid \vec{x}$ as

$$y \mid \vec{x} = M_y(\vec{x}) + WN(0, \sigma_U^2(\vec{x})) + WN(0, \sigma_N^2(\vec{x})) \tag{23}$$

Where $M_y(\vec{x})$ is a composite model's estimate for $E[y \mid \vec{x}]$, $\sigma_U^2(\vec{x})$ is the composite model's variation or uncertainty in its assessment of $E[y \mid \vec{x}]$, $\sigma_N^2(\vec{x})$ is the noise level inherent to the problem which prevents memorization of the data, and $WN(0, \sigma^2)$ is a white noise generator with variance σ^2. Therefore, the variance $Var[y \mid \vec{x}]$ can be described as

$$\sigma_m^2(\vec{x}) = \sigma_N^2(\vec{x}) + \sigma_U^2(\vec{x}) \tag{24}$$

The variation $\sigma_N^2(\vec{x})$ decreases as the problem becomes more deterministic, where $\sigma_U^2(\vec{x})$ decreases as the number of explanatory inputs decrease and the number of non-repeating patterns increase. Since $\sigma_N^2(\vec{x})$ is problem dependent and $\sigma_U^2(\vec{x})$ ultimately depends on how well the problem is sampled, the problem dependent parameter σ_L, if unknown, can be estimated through the distribution of $\sigma_N(\vec{x})$ across all $\vec{x}$. It may be useful to split up the error estimation process for $\sigma_m(\vec{x})$ into these two components, so that a divide and conquer

approach can provide an alternate way of modeling the confidence interval σ_m for an arbitrary $\vec{x}$. A good way of estimating $\sigma_N^2(\vec{x})$ is through the following relation:

$$\sigma_N^2(\vec{x}) \approx M_{y2}(\vec{x}) - M_y^2(\vec{x}) \tag{25}$$

Where $M_{y2}(\vec{x})$ is a memorizing model's estimate for $E[y^2 \mid \vec{x}]$, and $M_y(\vec{x})$ here is a memorizing model's estimate for $E[y \mid \vec{x}]$. These models determine how well the problem can be memorized. By combining the outputs of $M_y(\vec{x})$ and $M_{y2}(\vec{x})$ into a new training set for $\sigma_N^2(\vec{x})$ according to the relation above, a model for $\sigma_N^2(\vec{x})$ can be obtained (perhaps using stacked generalization) with the potential to generalize over some future set. A manipulation of network size (perhaps common to both models $M_y(\vec{x})$ and $M_{y2}(\vec{x})$) may be required to find a suitable "memorizing" model. Fortunately, something can be targeted through this manipulation process: the highest possible confidence coefficient r_σ for the modeled errors on the stacked generalization model.

Now that $\sigma_N^2(\vec{x})$ can be estimated, an assessment of $\sigma_U^2(\vec{x})$ will complete the estimate for $\sigma_m(\vec{x})$. This quantity represents the composite model's uncertainty in its estimate of the unknown quantity $E[y \mid \vec{x}]$. It generally is higher in relatively unexplored regions of $\vec{x}$, and lower in regions where many points are condensed. An estimation method for $\sigma_U^2(\vec{x})$ will be introduced below after revisiting stacked generalization and quantifying the concept of diversity.

Stacked generalization and model diversity

In the developments above, a framework was built to test a model's error estimation in terms of the confidence coefficient r_σ. Error splitting was introduced to help explore issues related to modeling confidence intervals. At work here is not only an inherent noise level in the problem for each given point in space, but also an uncertainty in the model's estimation of $E[y \mid \vec{x}]$. The stacked generalization process discussed earlier directly estimates the summation of the two variances in one step. This has the strong advantage of simplicity, since there may be no way to clearly demark the two separate forms of noise if the terms are modeled separately. Still, there are problems where splitting errors may be definitively the best approach to use. Below, this alternative approach for filling in the missing gap of $\sigma_U^2(\vec{x})$ will be discussed. Also, the model stacking process described above relied on utilizing various subspaces of the data set in both the input dimension and the pattern dimension to produce diverse models. This is central to the overall stacking process, and is essential to the error estimation process. Without diversity at Level 0, the Level 1 model will be deprived of useful new information that may assist in improved performance for both the composite model's output and error estimation. In fact, diversity can be directly linked to $\sigma_U^2(\vec{x})$ which will be approximated in an alternative stacking approach below.

Diversity, quantified

After turning an abstract word like "diversity" into a computable number, the result will be fed to the Diversified Committee Machine (DCM) stacking approach described in the next section. It is important to note that the following quantification of model diversity lacks

some degree of rigor. This is debatably acceptable, since a lack of precision is better than a lack of an estimate altogether. In the stacking process described above, various subspaces of both the inputs and patterns were utilized in generating models, so high levels of diversity are expected between the models. How much (approximately) will be discussed next.

Let the definition of diversity Δ_{AB} be defined between each A-B pair of Level 0 models. At first, limit the scope of this definition to just artificial neural networks since they contain nicely global characteristics. Each trained Level 0 network can be probed and an approximation retrieved for how important the neural network evaluates each input in the raw trainable data set, with the results stored in some normalized importance vector $\bar{I}$ (described above). The definition of pair-wise diversity is comprised entirely of how the trained networks differ in how the raw data's input space is utilized nonlinearly to achieve its end result. Let the following be the estimate for pair-wise model diversity:

$$\Delta_{AB} = \sum_{i=1}^{Ninputs} \left[(1 - C_i) \left| I_{A,i} - I_{B,i} \right| + \tfrac{1}{2} C_i \left| \left((I_{A,i} + I_{A,j}) - (I_{B,i} + I_{B,j}) \right) \right| \right] \tag{26}$$

Where j is the input having the highest magnitude of the linear correlation coefficient with input i, $I_{D,k}$ is the importance of input k to model D normalized to $[0,1]$ so that $\sum_{k}^{Ninputs} I_{D,k} = 1$, and C_i is the maximum absolute value of the linear correlation coefficient between input i and any other input k across the training set, i.e. $C_i = MAX(|r_{ik}|, k)$.

The purpose of the C_i term is to help prevent linearly repeated inputs from confusing the estimates of nonlinear differences in $\bar{I}$ between model A and B. If there are no inputs

that are linearly related to other inputs in the raw data set, then C_i will be calculated as zero for each i, and then the following simplified relationship holds:

$$\Delta_{AB} = \sum_{i=1}^{Ninputs} \left| I_{A,i} - I_{B,i} \right| \tag{27}$$

Diversified committee machines (DCM)

DCM stacking is similar to ordinary SG's Level 1 stacking in that the underlying driving force in the stacking process is model diversity. Thus far, stacked generalization provided diversity by training Level 0 models over various partitions of the data set and over various subsets of the input space. The Level 1 model reached out for the diverse strengths in the models below by learning an additional set of free parameters proportional to the number of Level 0 models.

DCM stacking is different than SG's Level 1 stacking in that no new free parameters are introduced above Level 0. This provides the potential advantage of utilizing the diverse strengths from a large number of generalized Level 0 models over a small amount of data since no new free parameters are introduced with each successive Level 0 model. Using SG on 500 Level 0 models and 200 data points would not likely result in preferred generalization for this reason. However, DCM wouldn't over train on this small data set above Level 0 since Level 1 free parameters do not exist.

Provided that estimates exist for the expected performance, p_A, of each Level 0 model A and the pair-wise diversity Δ_{AB} between all pairs of models $A-B$, a weighting function w_A can be introduced for each model A:

$$w_A = p_A \sum_{B,\forall B \neq A} \Delta_{AB} \tag{28}$$

Where p_A can be calculated, for instance, as the r^2 or RMS^{-1} performance of y_A over some new cross validation set relevant to $\vec{x}$. If more sophistication is required, then let $p_A = p_A(\vec{x})$, where the expected performance is determined based on the known local performance y_A on non-Euclidean nearest neighbors (see modified GRNN above) near the space of $\vec{x}$. Similarly, $\Delta_{AB} = \Delta_{AB}(\vec{x})$ is also computable in the same way with a localized calculation for $\vec{I}_A = \vec{I}_A(\vec{x})$. In this way, a distance metric is folded into the DCM stacking process through $w_A = w_A(\vec{x})$ without the need for new free parameters at the upper level of stacking.

As mentioned above, the behavior of $y \mid \vec{x}$ is not strictly known since it may well be undefined in continuous input space. Therefore, one way to estimate $E[y \mid \vec{x}]$ and $Var[y \mid \vec{x}]$ is to artificially generate points for y around $\vec{x}$ based on known model performance and diversity near regions of space defined by $\vec{x}$. As a model's performance and/or diversity decreases, the weighting function w_A should decrease. With this intuitive way of generating $y \mid \vec{x}$ without necessarily having any discrete points of y at $\vec{x}$, the composite DCM model becomes straightforward:

$$E(y \mid \vec{x}) \approx \bar{y}(\vec{x}) = \frac{\sum_A w_A(\vec{x}) y_A(\vec{x})}{\sum_A w_A(\vec{x})}$$

(29)

$$Var(y \mid \vec{x}) - \sigma_N^2(\vec{x}) = \sigma_U^2(\vec{x}) \approx \frac{\sum_A w_A(\vec{x})\left(y_A(\vec{x}) - \bar{y}(\vec{x})\right)^2}{\sum_A w_A(\vec{x})}$$

(30)

This completes the basics of the DCM stacking approach. Diversified Committee Machines are intuitive because the diversity and performance tendencies of models over specific regions of $\vec{x}$ are utilized, similar to what would be expected in a committee-like decision. The main disadvantage of this method is the need for a secondary process to estimate $\sigma_N^2(\vec{x})$. An approach was given earlier for manipulating network sizes in a memorizing process to estimate this quantity. Also, in a scaling process similar to that in Modified Series Association (MSA), the DCM estimate for $\sigma_U^2(\vec{x})$ can be artificially scaled with a constant k_U over a cross-validation set to match the desired hit rate of 95% at $2\sigma_m(\vec{x})$ so that $\sigma_m(\vec{x}) = k_U \sigma_U(\vec{x})$. This way $\sigma_N^2(\vec{x})$ can be temporarily ignored. Either way, the goal is to reach the desired hit rate and to maximize r_σ.

In the following section, a new data set is introduced as a benchmark for the stacking process. The data is artificially generated using deterministic sets of rules so that the elements of the set $\{s, \sigma_L, \sigma_N(\vec{x})\}$ are all set equal to zero. This will simplify a comparison between SG's Level 1 stacking and DCM stacking. The DCM stacking approach can be used to combine neural networks and localized functions like GRNN's, but this analysis requires lumping pair-wise diversity within model types (like ANN and GRNN) and providing the neural networks' estimates for $\vec{I}$ as a means for assessing the global diversity between

localized models. The simplified analysis of combining just ANN's at Level 1 with DCM's will be provided here.

Spooky particle data set

This section introduces a new data set to the field of data modeling as a benchmark for inductive learning machines. Its intended purpose is to provide a benchmark for model comparison and a resource for testing nonlinear modeling methods. Although the set is artificially generated, it can be thought of in terms of real-world physics. The "spooky" description of a particle's behavior originated when Albert Einstein became intrigued with experiments involving the quantum entanglement of twin-photons that originated from a common parent photon. The results apparently suggested that the fundamental particles behave like psychic geniuses, each knowing the other's state instantaneously without regard to the distance separating them. He sneered at the very possibility of such a thing, calling the process "spooky action at a distance."

The spooky particle data set introduced here includes the set of known events that a single particle has experienced in its universe throughout all time. If the particle is in fact a psychic genius, then it should instantly know some aspect of its final state given any initial state. The goal here is to teach the particle how to become psychic in some way by allowing it to learn from the data set generated by its surrounding universe. A simple example is the best way to demonstrate this.

Let the universe be 3 dimensional (not counting time as a dimension), void of gravity and other such forces, and consist of an empty rectangular box enclosing the volume spanned by (0,0,0) to (l, w, h). Let the particle's initial state in its universe be given by its initial

position and velocity in the box $[X_0, V_{x0}, Y_0, V_{y0}, Z_0, V_{z0}]$. Let its final state be given by when it collides with one of the inner walls of the box, and mark this state with T_t, or time until impact from its initial state. If the particle is in fact psychic it should instantly know T_t given any initial state in that universe. Thus many examples can be generated using the specified laws of physics to obtain a data set having inputs $[X_0, V_{x0}, Y_0, V_{y0}, Z_0, V_{z0}]$ and desired output T_t. For this discussion, let the particle behave according to simple Newtonian rules of motion and let the rest of the universe be completely static.

The universe could contain internal (nonmoving) objects, have irregular boundaries, exist in a high number of dimensions, utilize gravity, etc. There are a wide variety of universes and corresponding data sets that can be generated which exhibit very interesting nonlinear properties. The dynamic particle only learns the laws of physics in its surrounding toy universe through example (by colliding with the walls or internal objects) and nothing else. It only becomes "spooky" after it learns how to predict T_t accurately based on previous known events.

A deterministic universe should generate data sets that yield $\sigma_L \approx 0$ and $s(\sigma_L, \sigma_H) \approx 0$. The trained composite error estimation model on this set should therefore be capable of achieving $r_\sigma \approx 1$ with $r_H(s \approx 0) \approx r_{e,\max}$. The results presented in this paper are based on the simplified universe in a box described above with dimensions $2x5x1$, $V_{\min} = 3$, and $V_{\max} = 5$, with (X_0, Y_0, Z_0) uniformly distributed in the box and (V_{x0}, V_{y0}, V_{z0}) uniformly distributed in a spherical shell with an inner and outer radius of $V_{\min}$ and $V_{\max}$ respectively. Also, 1000 points were generated for the training set and 1000 for the test

set. This is a challenging number to use for this problem because T_i is somewhat difficult for the base model to learn but can still be generalized in the test set past the 0.8 r^2 range.

Utilizing the modified GRNN with a non-Euclidean distance metric, a value for s was estimated at 0.069 given this data set ($\sigma_L = 0.018, \sigma_H = 0.279$), which is reasonably close to the known value of $s = 0$ for purely deterministic problems. The stacking methods MSA and DCM were compared using 5 diverse single hidden layer neural networks on this 1000 pattern test set. In both cases, the same Level 0 models were used. The resulting performance of the composite model output of each method is shown in Table I. MSA and DCM produced very similar performance results on this problem with respect to the stacked model output, with DCM slightly better. The performance in error estimation of the models using the framework outlined above is shown in Table II. Here, MSA produced noticeably better results, demonstrating strong confidence coefficients $r_{\sigma,U}$, $r_{\sigma,G}$, $r_{\sigma,T}$ which are fairly close to the limit of 1. The subscripts U, G, and T represent "uniform", "Gaussian", and "true". Since $r_{H,T}(MSA) = 0.61$ and $r_{H,T}(DCM) = 0.52$, it would be theoretically impossible for the DCM stacking process to produce an error estimate correlation coefficient r_e of 0.55 given enough samples and a Gaussian error distribution, but it would be possible for MSA in this case. The bottom-line results regarding the error estimates can be seen perhaps through $r_{\sigma,T}(MSA) = 0.75$ and $r_{\sigma,T}(DCM) = 0.61$. There is room for improvement in both cases, but not a great deal, especially for MSA. The DCM model here used a constant w_A, p_A, and Δ_{AB} based on cross-validation set performance instead of a more relevant non-Euclidean estimate for each new point in space $\vec{x}$ through a localized $w_A = w_A(\vec{x})$. This gave MSA an advantage in that is utilized a distance metric in its

confidence interval estimation, but DCM wasn't provided with that information here. This sheds light on the importance of a distance metric in estimating errors and therefore demonstrates the usefulness in locating the supportive cases of a network's decision.

It's worth noting that this is just one example. Each method has its specific strengths and there will be times when each respective method, when at its best, outperforms the other. This depends on the problem at hand.

Conclusions

In the past, general, non-parametric, data driven methods, such as ANNs, could not be used for modeling with a model error prediction objective. Modified Series Association and Diversified Committee Machines provide the required reliably estimated error bounds as well as improved results though diverse model combination and consolidation. These methods can provide both an accurate model of the desired output and an estimation of the error bounds on the predicted output in an automated and user-friendly manner. The resulting models are both accurate and precise. MSA and DCM tend to generalize well, and they each have their specific strengths. They also take advantage of the results of ranking methods to preprocess and partition the available data. Diversity is provided in this sense. The framework introduced above for quantifying diversity and error estimate reliability allows future comparisons between error estimation techniques. The stacking methods introduced here are provided as a set of tools, each of which having the potential perform well in certain situations, depending on the problem at hand.

References

Bartlett, E. B. and A. Whitney (1999). "On The Use Of Various Input Subsets For Stacked Generalization," Intelligent Engineering Systems Through Artificial Neural Networks, Vol 9, 117-124.

Bartlett, E. B. (1994). "Dynamic node architecture learning: an information theoretic approach", *Neural Networks 7*, 129-140.

Bartlett, E. B. and K. Kim, (1993). "Error Bounds on the Output of Artificial Neural Networks," *ANS Trans.*, 69,197-199.

Bartlett, E. B. and R. E. Uhrig (1992). "Nuclear power plant status diagnostics using an artificial neural network," *Nuclear Technology* 97, 272-281.

Bhat, N., and T. McAvoy, (1990). "Use of Neural Nets for Dynamic Modeling And Control of Chemical Process Systems," *Computers Chem. Engng.* 14, 573-583.

Blum, E. K. and L. K. Li, (1991). "Approximation Theory and Feedforward Networks," *Neural Networks*, 4, 511.

Cherkassky, S. and F. Mulier, (1998). *Learning From Data: Concepts, Theory, and Methods (Adaptive and Learning Systems for Signal Processing, Communications and Control Series).* John Wiley & Sons, New York, New York.

Cybenko, G. (1989). "Approximation by superposition of a sigmoidal function," *Mathematics of Control, Signals, and Systems 2*, 303-314.

Haykin, S., (1999). *Neural Networks: A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, New Jersey.

Hecht-Nielsen, R. (1990). *Neurocomputing.* Addison-Wesley, Reading, Mass.

Kim, K. and E. B. Bartlett, (1996). "Nuclear Power Plant Fault Diagnosis Using Neural Networks with Error Estimation by Series Association," IEEE Transactions on Nuclear Engineering Vol 43, No 4, pp 2373 - 2388

Kurkova, V. (1992). "Kolmogorov's theorem and multilayer neural networks," *Neural Networks*. 5,501-506

Lapades, A., and R. Farber, (1987). *Nonlinear signal processing using neural networks: prediction and system modeling.* Los Alamos National Laboratory Technical Report LA-UR-87-2662.

Lippmann, R. P., (1987). "An Introduction to Computing with Neural Nets," *IEEE Acoustics Speech and Signal Processing Magazine*, 4, 4.

Mackey, M. C. and L. Glass (1977). "Oscillations and Chaos in Physiological Control Systems," *Science*, 197, 287-289.

Miller, W. T., R. S. Sutton, and P. Werbos (Eds.), (1990). *Neural Networks for Control.* Press. Cambridge, Mass.

Mitchell, T., (1997). *Machine Learning*. McGraw-Hill, New York, New York.

Narendra, K. S. and K. Parthasarathy, (1990). "Identification and Control of Dynamic Systems Using Neural Networks," *IEEE Trans. Neural Networks*, 1, no. 1, 4.

Parzen , E., (1962). "On estimation of a probability density function and mode," *Ann. Math.* fist. 33, 065-1076.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, (1986). *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol. 1 & 2, MIT Press, Cambridge, Massachusetts.

Russell, S. and P. Norvig, (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey.

Schmitz, G. P. J. and C. Aldrich (1999). "Combinatorial Evolution of Regression nodes in Feedforward Neural Networks," *Neural Networks,* 12, 175-189.

Specht, D. F., (1990). "A General Regression Neural Network," *I&E Transactions on Neural Networks,* 2, 568.

Sridhar, D. V., (1996). "Process Modeling Using Stacked Neural Networks," *Ph.D. Dissertation, Iowa State University, Ames, Iowa.*

Upadhyaya, B. R., and E. Eryurek, (1992). "Application of neural networks for sensor validation and plant monitoring," *Nuclear Technology* 97, 170-176.

Uhrig, R. E. (1989). "Use of neural networks in nuclear power plant diagnostics," *Proc. Int. Conf on Availability Improvements in Nuclear Power Plant,* 310-315 Madrid, Spain.

Venkatasubramanian, V., and K. Chan, (1989). "A neural network methodology for process methods," *Journal of Applied Meteorology* 31, 405-420.

Wichmann, N. L. and E. B. Bartlett (1997), "Ranking Input Variables using General Regression Neural Networks", *Intelligent Engineering Systems Through Artificial Neural Networks: Volume 7,* pp. 959 – 964.

Wichmann, N. L., (1997). "Variable Importance Ordering for Evapotranspiration using General Regression Neural Networks", *M.S. Thesis, Iowa State University, Ames Iowa.*

Wolpert, D. H., (1990). "A Mathematical Theory of Generalization: Part I and Part II," *Complex Systems,* 4, 151.

Wolpert, D. H., (1992). "Stacked Generalization", *Neural Networks,* 5, 241.

Zhang, X., J. P. Mesirov, and D. L. Waltz, (1992). "Hybrid system for prediction for protein secondary structure prediction," *Journal of Molecular Biology*, 225, 1049-1063.

Tables and figures

Table I. Test set results of the stacked output using the Spooky Particle data set

Stacked Model	MSA	DCM
Correlation Coefficient	0.884	0.909
RMS	0.077	0.075

Table II. Test set results of the stacked error estimates using the Spooky Particle data set

Stacked Model	MSA	DCM
r_e	0.45	0.32
$r_{H,T}$	0.61	0.52
$r_{\sigma,T}$	0.75	0.61
$r_{\sigma,G}$	0.93	0.64
$r_{\sigma,U}$	0.82	0.57

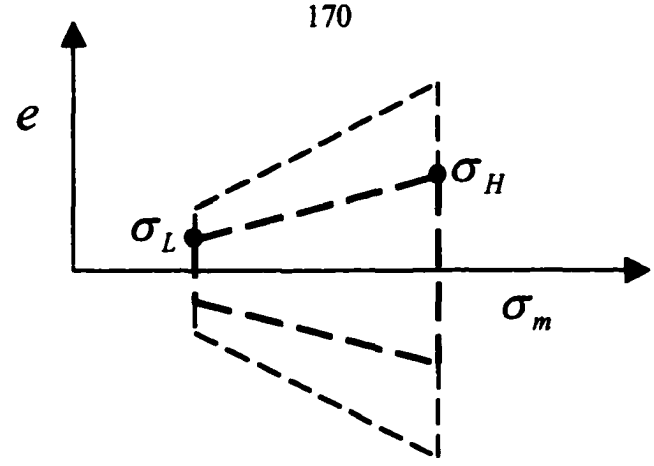


Figure 1. Best possible model for residual distribution

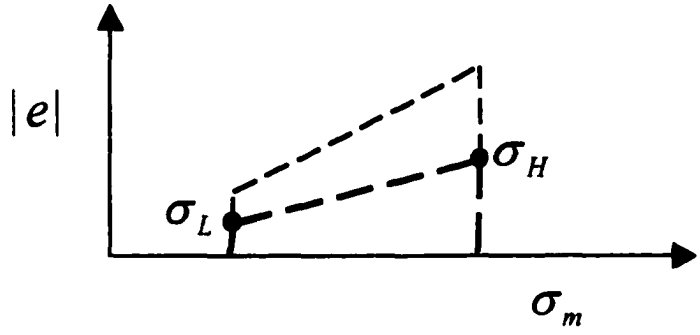


Figure 2. Best possible model for error distribution

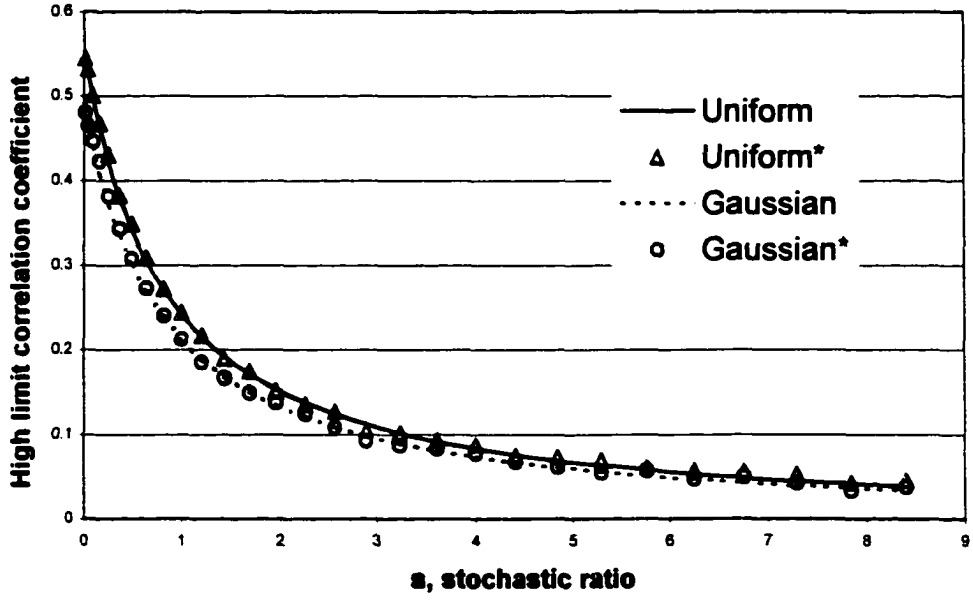


Figure 3. Effect of stochastic ratio s on highest attainable r_H

Nomenclature

Δ_{AB}	Pair-wise diversity between model A and B
$\sigma_m(\vec{x})$	Modeled 67% confidence interval for base model at $\vec{x}$
σ_H	High limit of the distribution for $\sigma_m(\vec{x})$
σ_L	Low limit of the distribution for $\sigma_m(\vec{x})$
$\sigma_N(\vec{x})$	Variation at $\vec{x}$ due to inherent noise in the data
$\sigma_U(\vec{x})$	Variation at $\vec{x}$ due uncertainty in the estimate of $E[y \mid \vec{x}]$
$\bar{\sigma}_m$	Mean of the distribution for $\sigma_m(\vec{x})$
σ_p	Free parameter optimized by a GRNN
τ	Mackey – Glass chaotic time series parameter
a	Mackey – Glass chaotic time series parameter
b	Mackey – Glass chaotic time series parameter
C	Measure of input redundancy
D	Distance metric used by a GRNN
e	Residual on base model
$\vec{I}$	Importance vector, which estimates the importance of each component of $\vec{x}$
k_U	Scaling factor for the DCM estimate of $\sigma_m(\vec{x})$
$M_y(\vec{x})$	Generic model estimate for $E[y \mid \vec{x}]$
$M_{y2}(\vec{x})$	Generic model estimate for $E[y^2 \mid \vec{x}]$
p_A	Expected performance of Level 0 model A

r_e	Measured correlation coefficient between sampled σ_m and $\lvert e \rvert$
$r_{e,\max}$	Highest attainable r_e for a given distribution of σ_m
r_H	Highest attainable r_e for a given s and a given distribution of σ_m
r_σ	Confidence coefficient, which measures error estimation performance
s	Stochastic ratio, which sets limits on r_H
w	Local weighting factor used by a GRNN
w_A	Local weighting factor used by DCM
$\vec{x}$	Input space
y	Output, generic description
$y(\vec{x})$	GRNN estimate for $E[y \mid \vec{x}]$
$y_2(\vec{x})$	GRNN estimate for $E[y^2 \mid \vec{x}]$
$y \mid \vec{x}$	Theoretical behavior of y in the area of $\vec{x}$

CHAPTER 6. GENERAL CONCLUSIONS

The Scaled Conjugate Gradient algorithm is an extremely fast and robust ANN method for supervised learning. This was demonstrated in chapter 2. With the addition of saving the network that performed the best on the cross-validation set, the algorithm can also generalize well. However, a technique for optimizing the network architecture is still necessary for any training algorithm to reach its full potential in generalization. Other complexity-regularization methods exist for improving generalization in this way, but most are relatively slow.

Every neural network in the analysis from chapter 3 was trained with 1000 iterations using the scaled conjugate gradient algorithm. Tables I-IV (chapter 3) show the RMS error and R^2 results of the trained networks for both the training and test sets. Various network architectures were selected for each of the four data sets, such as "4x10x1" (4 inputs, 10 hidden nodes, 1 output) and "6x15x3x1" (6 inputs, 15 nodes in the first hidden layer, 3 nodes in the second hidden layer, 1 output). Table V (chapter 3) consolidates the results of tables I-IV (chapter 3) so that the best network architecture for each problem is displayed along with the corresponding test set RMS and R^2.

After the appropriate (best) networks were trained and filtered, their internal structures were probed using the variable importance (saliency) methods described above. As a means for comparison, a standard GRNN is introduced alongside the saliency estimation methods described. In this case, each input variable is equally important. See figures 6-9 (chapter 3) for a graphical representation of all of these estimates.

Modified GRNN results, generated from the corresponding saliency estimates, are shown in tables VI-IX (chapter 3). From the test set RMS values in these tables, each saliency estimation method is ranked for each data set in terms of modified GRNN accuracy and shown in Table X (chapter 3). The average overall rank is highlighted, as well as the rank based on the overall average test set R^2. The sensitivity analysis method had the highest overall rating out of the four chosen data sets, followed in close proximity by the input elimination method. The signal variation method was rated third, followed by the weight magnitude and then the 2nd order sensitivity method. In last place was the standard GRNNs approach. This was expected, since an ordinary GRNN equally weights each input dimension (using a Euclidean distance metric) regardless of the relevance of each input.

Tables I-IV (chapter 4) show the RMS error and R^2 results of the models described above over various ANN architectures for the Mackey Glass data set. The KNN, GRNN, and ANN models were independently trained. The remainder of the methods, FWKNN, FWGRNN, LMLR, and NNA, were all dependent on the corresponding ANN through the calculation of the distance metric using scaled inputs based on $\vec{R}$ and $\vec{I}$. LMLR used the FWGRNN estimate for σ_G to locally weight each pattern. NNA stacked a second ANN on top of the FWGRNN, LMLR, and ANN models using $L(\vec{x})$ and d as extra inputs to help decide which local or global modeling approach to use in a particular case. Tables V-VIII (chapter 4), IX-XII (chapter 4), and XIII-XVI (chapter 4) show the RMS error and R^2 results of the models over various ANN architectures for the chaos-13, spooky particle, and letter recognition data sets respectively. Table XVII (chapter 4) shows a modeling comparison for all of the data sets used here. The results of Table XVII (chapter 4) have been averaged (by

test set R^2) over all of the ANN architectures used for each problem. The numbers represent a ranking (from 1 to 7) so that all of the information in the previous tables can be seen at once.

Refer to Figure 2 (chapter 4). This shows the test set R^2 average performance (over all ANN architectures) of each model on the Mackey Glass data set. Figures 3-4 (chapter 4) represent the same model comparison over the chaos-13 and spooky particle data sets. Figure 5 (chapter 4) depicts the overall performance comparison between all of the methods on all of the problems over all of the ANN architectures used. The methods not using a weighted distance metric based on the trained ANN (KNN, FWKNN, and GRNN) are trailing the other methods by a large margin. Hybrid approaches that combine the neural network with nearest neighbors (FWGRNN, LMLR, and NNA) tended to outperform ANN as a whole. However, only NNA tended to outperform the corresponding ANN at an optimal architecture.

General Discussion

The sensitivity analysis method for computing saliency estimates on the input vector was shown here to be the most accurate of the standard and 5 modified GRNN approaches demonstrated in chapter 3. Since the curse of dimensionality decreases with the increase of accuracy of the neural network saliency estimates, a modified GRNN can essentially reconstruct the intentions of the original trained ANN with greater success than a standard GRNN. After probing the networks for an estimate of the decision-making power of each input node, the input space was renormalized accordingly so that the modified GRNN would

find more precise supportive cases of the neural network's decisions. From this, a confidence interval on the network can be more accurately constructed. In this way, some of the blackness from the black box of an artificial neural network was diminished. Various methods for highlighting the internal structure of an ANN were compared in a completely quantitative way using the modified GRNN technique.

Seven general data-driven methods were described in chapter 4 for mapping nonlinear data, three of which are standard approaches (KNN, GRNN, and ANN). Four other approaches (FWKNN, FWGRNN, LMLR, and NNA) are described which utilize a weighted distance metric from the output scaling based on the DMP demonstrated by the corresponding trained ANN. They combine local weighting methods with the global regression capability of neural networks to allow for potentially improved accuracy at the expense of increased computation (compared to recalling a neural network). In general, over all trained architectures tested on the four problems described above, the resulting averaged test set R^2 suggests the following:

NNA>LMLR>FWGRNN>ANN>GRNN>FWKNN(k=1)>KNN(k=1)

Combining the outputs of LMLR, GRNN, and ANN in a way similar to the stacking approach of NNA with an ANN-based weighted distance metric can improve generalization. Although in some cases an optimal architecture ANN can result in greater accuracy than a FWGRNN or LMLR, the NNA approach is general enough to reduce to the ANN global solution if a particular problem demonstrates this is the best thing to do.

In the past, general, non-parametric, data driven methods, such as ANNs, could not be used for modeling with a model error prediction objective. In chapter 5, Modified Series Association and Diversified Committee Machines provide the required reliably estimated error bounds as well as improved results though diverse model combination and consolidation. These methods can provide both an accurate model of the desired output and an estimation of the error bounds on the predicted output in an automated and user-friendly manner. The resulting models are both accurate and precise. MSA and DCM tend to generalize well, and they each have their specific strengths. They also take advantage of the results of ranking methods to preprocess and partition the available data. Diversity is provided in this sense. The framework introduced above for quantifying diversity and error estimate reliability allows future comparisons between error estimation techniques. The stacking methods introduced here are provided as a set of tools, each of which having the potential perform well in certain situations, depending on the problem at hand.

APPENDIX A. ELECTRIC FUTURES FORECASTING USING ARTIFICIAL NEURAL NETWORKS

A paper published in the Thirty-Fifth Annual Report of the Electric Power Research Center / Power Affiliate Research Program, April 1998

Craig Carmichael, Eric Bartlett, and Gerald Sheble'

Department of Electrical and Computer Engineering,

Iowa State University, Ames, IA, 50011

Abstract

Deregulation of the electric power industry will increase competitiveness among the various electric utilities. These electric companies will compete in a commodities futures market to buy and sell electricity and other materials needed for the generation of electric power. They can reduce risk and increase profits by trading electric futures in the commodities market. How well they trade determines how much they profit, and this depends on how well they forecast certain future events. The objective here was to improve artificial neural network (ANN) forecasting techniques and apply them to electric futures historical data. These techniques included an importance calculation, a diversity calculation, and two new ANN training methods. All of them were designed to improve the generalization potential of the ANN models. The two training methods were benchmarked against a standard approach on the electric futures data. One of them, called the noise feedback descent method, showed some promise in terms of complexity regularization.

Introduction

An ANN'S job is to adequately map an example set of input-output pairs. Mapping is achieved through an optimization (training) procedure designed to minimize the RMS error over the example set by manipulation of the network's set of synaptic weights. For each pattern in the example set, an ANN under supervised learning is told what the inputs are and also what the corresponding outputs are supposed to be. After the ANN has learned this mapping to some predetermined level of accuracy, the resulting ANN model is tested on a novel set, which has never been seen before by the network The ultimate goal for the ANN is to adequately map this set via generalization.

This goal becomes increasingly difficult to attain as the input space increases in size and the number of patterns in the example set decreases. The problem is compounded when the desired mapping becomes less functional. In the case of forecasting the volatility and price of electric futures for purposes of hedging and risk management, generalization is constrained by all of the above factors. Time series analysis significantly expands the input space, the number of patterns in the example set is currently on the order of 100, and the system under consideration is far from deterministic. In light of all the challenges that extraordinarily difficult problems such as this one offer an ANN, great emphasis was placed on improving the generalization potential of existing ANN training methods.

ANN training

Training involves the manipulation of the network's set of synaptic weights. These weights are free parameters, which more or less determine how complex the ANN's mapping

can be. As the number of hidden layers and nodes in each layer increases, so does the number of connection weights between nodes. Therefore, this complexity also increases with network size. However, no rule of thumb can determine the best network architecture to train on before training. Therefore, some kind of iterative approach is needed to find just the right level of complexity to maximize the generalization potential of the ANN.

Also, stacking multiple trained networks has been shown to increase the robustness of the overall ANN model. Since the overall model can require a great deal of number crunching, it is obviously important to use the fastest possible training algorithm known for each trained architecture and each stacked, optimal-architecture, trained model. 1%at algorithm is the scaled conjugate gradient algorithm (SCG), not the accepted standard of back-propagation (BP). To demonstrate the importance of this choice of training algorithm, a non-stacked static architecture (2x21x14x7xl) was trained on the well-known spiral problem using BP and again using SCG on the same machine. BP required 56 minutes of training time and SCG required only 21 seconds, yet the resulting models were nearly identical. le enormous difference in training time between the two methods will be increased by another factor of 5 through the use of advanced programming methods. Although this vast difference in speed between SCG and BP cannot be achieved with every problem, training speed is very much an issue in the trial-and-error phase of finding the best solution to the problem at hand.

ANN methods development

The reader at this point is encouraged to temporarily step back from the problem of forecasting electric futures and look inside the black-box aspect of artificial neural networks. The reason for this emphasis is that the success or failure of the overall ANN forecaster will

ultimately hinge on the generalization potential of the methods used to train and stack it. This becomes increasingly true as the problem becomes more difficult. Also, any problem that can be quantitatively broken down into an example set of input-output pairs is just data to the ANN, That's all it really cares about.

Feedback set and generalization

In the past, some researchers gave artificial neural networks a bad name. They accomplished this by simply training an ANN on the example set and reporting the results. As mentioned above, the true test for generalization is the trained ANN's performance over the novel test set. The hope is that the ANN will not memorize the idiosyncrasies in the example set, but rather map global trends that are likely to occur in the novel future set. This hope can be realized to a greater extent by breaking the example set into two subsets, a memorization set and a cross-validation set. Standard approaches utilize a training method such as SCG over the memorization set and repeatedly check that model's performance over the cross-validation set, saving the model with the lowest cross-validation RMS. The saved model at the end of training is considered the best model and this is what is tested a single time on the novel future set.

While standard approaches passively utilize the cross-validation set as a simple repeated check, the methods developed here actively pass this information back to the training algorithm, where that algorithm then decides how to regulate the complexity of the model. This type of feedback is of utmost importance because without it, the training algorithm cannot adequately decide when the optimal network architecture has been reached or what to do about it if it hasn't. Because of the active use of the cross-validation set in our

research, it was renamed "feedback set." When the feedback set suggests that the ANN is generalizing well, the training algorithm adds free parameters (synaptic weights and nodal biases) to the network architecture, Otherwise it chops less important pieces away from the network The determination of what to chisel away from the network in this case is based on the importance calculation that follows below.

Importance calculation

The importance calculation is the underlying basis for two of our SCG-based training methods described in the following sections. A dynamic node architecture heuristic with feedback (DNAF) adds and eliminates complete features (nodes) at a time while the noise feedback descent (NFD) method delicately regenerates and eliminates individual synaptic weights. Both methods, when given the feedback set's signal to reduce the size of the network architecture, eliminate the least important parameters first.

The standard approach for determining the importance of a synaptic weight is simple: the importance is directly proportional to the absolute value of the weight. This approach, however, falls apart easily under close inspection. For example, if the magnitude of a synaptic weight fanning into a hidden node is very large but all of the weights fanning out of the node are zero, then there is no possibility that any of the signal passing through the large weight will reach the output layer. Therefore, the assumption that the importance of a weight is independent of all other weights in the network is inappropriate. Also, as the variance (across all patterns) of the signal passing through a synaptic weight approaches zero, the information contained in the signal becomes indistinguishable from the adjacent bias parameter. Here too, the magnitude of the weight becomes irrelevant. The standard approach

for determining the importance of a hidden or input node in the network is closely dependent on the standard importance calculation for a synaptic weight. Therefore, this standard calculation is also faulty. The first set of equations in the attached appendix briefly outlines a much more accurate and robust importance calculation for both synaptic weights and nodes within the network.

As described above, the importance calculation is used to assist the training algorithm in the task of optimizing the network architecture, which increases the generalization potential of the network, which provides a good level zero model to stack with a collection of other level zero models. This, in turn, results in a good overall ANN model to be applied to the problem at hand. That begins to bring the other major contribution of the importance calculation into focus.

The term "stacking," referred to in various places above, can be thought of as simply a way of mediating the outputs from multiple (level zero) ANN models. One very simple stacking approach is to let the stacked model's output be equal to the average output from all level zero models for each pattern. For example, let's say that we have four level zero models and, for a given pattern, three of them tell you that the answer is zero but the other one insists that the answer is exactly one. Also, let's say that each of the four models performed equally well over the feedback set. What should the stacked model's output be?

The first response one might give to this situation is that the answer should be one-fourth. Now what happens if you learn that each of the three models yielding zero as the answer was actually the exact same model in weight space? Because the three models as a set completely lack diversity, probably a more diplomatic solution to this dilemma is to let the stacked model's output be equal to one-half for this pattern. In the same way that a

calculation for diversity can yield a more suitable result for the stacked model's output, it can also yield a more suitable result for confidence in the stacked model's output. This in fact has direct implications for risk management using stacked networks in electric futures forecasting.

We as logical creatures routinely assess whether or not to take an action based on how sure we are that it's the right thing to do. Our certainty is based largely on how much confirmation we receive from diverse internal resources – the greater the diversity and the lesser the disagreement, the greater the certainty. Any student who has checked his or her work by attacking each homework problem from a variety of different angles can verify this.

Now that considerable justification has been introduced to explain why an accurate diversity assessment is important, it is time to discuss how to carry out the calculation. For level zero models A and B, carry out the importance calculation (as described in the appendix) across each of the input nodes and store the result in their respective importance vectors, $\vec{I}_A$ and $\vec{I}_B$. Then calculate the diversity Δ_{AB} between models A and B as follows:

$$\Delta_{AB} = \tfrac{1}{2} \sum_{i=1}^{Ninputs} \left[(1 - C_i) \left| I_{A,i} - I_{B,i} \right| + \tfrac{1}{2} C_i \left| \left(I_{A,i} + I_{A,j} \right) - \left(I_{B,i} + I_{B,j} \right) \right| \right] \tag{1}$$

C_i refers to the maximum absolute value of the linear correlation coefficient between input i and any other input, with j being that other input. With this, a quantitative measure for diversity can be applied as a weighted term to pairs of level zero models, which can improve the stacked model's forecast prediction and certainty of prediction. Therefore, this is an

excellent way to assess the confidence in the results from the stacked model on a pattern-by-pattern basis.

Advanced training algorithms

Because of space limitations, the SCG-based training algorithms currently developed cannot be adequately described. As mentioned above, the DNAF process manipulates the network architecture on a nodal basis while NFD prunes and regenerates individual synaptic weights. These iterative decisions are based on the difference in batch error between the feedback set and the memorization set ($E_{fb} - E_{mem}$). Once again, the objective is to optimize the network architecture in a way that will increase the generalization potential of the network. This should improve the results to the problem at hand. For a brief outline of NFD, refer to the attached appendix.

Equations section

The importance calculation described in the text above is calculated from the following set of equations:

$$L_{Wij} = \mid w_{ij} \mid \sigma[x_i] \tag{2}$$

$$\bar{L}_{Wij} = \frac{L_{Wij}}{\sum_{k=1}^{Nbelow} L_{Wkj}} \tag{3}$$

$$G_j^l = \frac{1}{O} \tag{4}$$

$$G_{Wij} = G_j^l \bar{L}_{Wij} \tag{5}$$

$$G_i^{l-1} = \sum_{j=1}^{Nabove} G_{Wij} \tag{6}$$

where

L_{Wij} = local importance of weight w_{ij}

w_{ij} = synaptic weight connecting node i below to node j above

$\sigma[x_i]$ = standard deviation of the signal from node i below across all patterns

$\bar{L}_{Wij}$ = normalized local importance of weight w_{ij}

$Nbelow$ = number of weights below fanning into node j

G_j^l = global importance of node j in layer l

O = total number of output nodes

G_{Wij} = global importance of weight w_{ij}

$Nabove$ = number of weights fanning out of node i

The noise feedback descent (NFD) method was derived to improve generalization through a combination of conjugate gradient descent, weight decay, feedback set information, and the

above importance calculation. This algorithm is inserted into the scaled conjugate gradient (SCG) algorithm just before it updates weight space a distance α along the conjugate gradient direction $\tilde{p}$. A complete description of SCG is given by Moller, 1993. The following set of equations highlight the NFD process, which utilizes the signal ($E_{fb} - E_{mem}$) to bias $\tilde{p}$ towards noiseless points in weight space:

$$I_{ij} = G_{Wij} \mid r_{ij} s_{ij} \mid \tag{7}$$

$$H_{I,J} = \frac{Nbelow}{\sum_{i=1}^{Nbelow} \frac{1}{I_{ij}}} \tag{8}$$

$$R_{ij} = \left(\frac{4 I_{ij}}{H_{I,J}} \right)^2 \tag{9}$$

$$\gamma_{ij} = \frac{1}{1 + R_{ij}} \tag{10}$$

$$p_{A,ij} = -\frac{w_{ij}}{\alpha} \tag{11}$$

$$V = \frac{E_{fb} - E_{mem}}{2\alpha \sum_{ij} \left[r_{ij} \gamma_{ij} (p_{A,ij} - p_{ij}) \right]}$$

(12)

$$\Gamma_{ij} = V\gamma_{ij}$$

(13)

$$p_{B,ij} = p_{ij} + \Gamma_{ij}(p_{A,ij} - p_{ij})$$

(14)

$$p_{B,0j} = p_{0j} + \sum_{i=1}^{Nbelow}(p_{B,ij} - p_{ij})\bar{x}_i$$

(15)

$$\tilde{p} = \tilde{p}_B$$

(16)

where

I_{ij} = global dynamic importance of weight w_{ij}

$H_{I,j}$ = hyper-geometric mean of all I fanning into node j

R_{ij} = relative dynamic importance of weight w_{ij}

γ_{ij} = virtual pruning parameter for weight w_{ij}

$p_{A,ij}$ = absolute pruning component for weight w_{ij}

V = virtual shrink factor

Γ_{ij} = actual pruning parameter for weight w_{ij}

$\tilde{p}_B$ = biased conjugate gradient direction

$\tilde{p}$ = scaled conjugate gradient direction (which is replaced by $\tilde{p}_B$)

All variables not described above are part of, and have been calculated by, the SCG algorithm. It is important to note that the NFD process is skipped entirely during SCG iterations where $E_{mem} \geq E_{fb}$. Also, the feedback set should be chosen so that it represents the memorization set well.

The data

Now the reader is encouraged to step forward again to the problem at hand. Electric futures historical data was obtained from the New York Mercantile Exchange. The data included two futures contracts, which spanned between January 2 and September 29 of last year. Currently only contracts for Palo Verde, Arizona, and the California/Oregon border are offered. Their symbols are KV and MW. Figure 1 illustrates the price vs. time for both contracts.

Using this data, which included each day's price for open, high, low, and close over several months' contracts, the model was set up so that it would try to forecast the change in opening price a day ahead for the next month's contract. One of the major dilemmas in this experiment was that the data sets consisted of only 143 day's worth of information (patterns). By forecasting a day ahead, using the last 15 days as inputs (time series data preprocessing), and discarding 7 obvious flaws in the data, only 120 patterns remained. From that, 20% was reserved for the novel test (future) set, leaving only 96 patterns on which the ANN could train. That training set was broken into a memorization (MEM) set and a cross-validation (CV) set, which was used as feedback for both DNAF and NFD. A random and "Last N" CV type (to preserve causality) was tested, as well as two different CV partitions (25% and

35%). The generalization performance of three different ANN training methods was tested using this setup-DNAF, NFD, and standard SCG.

Focus was placed on maximizing the potential of the level zero ANN models to forecast futures prices. With an adequate setup and adequate level zero models, the use of stacking and the diversity calculation will aid in the forecasting of electric futures volatility. Once again, this has direct implications towards hedging and risk management.

Results

The results shown in Tables I and II demonstrate the effectiveness of the methods developed to improve generalization on this problem. For each set, two performance measures were given: *RMS* and R^2 (square of the linear correlation coefficient). Both measures describe the difference between the desired and actual output across all patterns in the set.

While DNAF appeared to have almost stellar performance on the CV sets, the performance didn't carry over to the novel test set (the R^2 measures were very near zero). This is so because DNAF trains on a large number of architectures. In the process it can get lucky in saving the model with the best performance on the CV set. With more patterns in the CV set, the probability of luck influencing the generalization potential of the network would diminish. However, this is not an option here.

Compared to both SCG and DNAF, NFD demonstrated the best overall generalization potential. Using a 35% Last N CV set, NFD achieved the highest R^2 and the lowest *RMS* in the novel test set.

Conclusions

Artificial neural network techniques were developed to aid in risk management and hedging for electric power companies. Although these techniques haven't yet been incorporated into a real world trading system, the level zero ANN performance will drive the expected performance of any trading system incorporating these techniques. Therefore, emphasis was placed on electric futures price forecasting using level zero ANN models.

The ANN techniques developed included an importance calculation for each node and synaptic weight in the network, a diversity calculation for improved stacked generalization and certainty assessment, and two training methods (DNAF and NFD) designed to improve the generalization potential of each level zero model. The diversity calculation, DNAF, and NFD were all based on the importance calculation. The results above indicate that NFD shows the most promise in terms of forecast generalization with the present amount of data. DNAF is expected to significantly outperform SCG, driven more data.

References

M. F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks,* vol. 6, no. 3, 1993, pp. 525-533.

E. B. Bartlett, 'Ilynamic node architecture learning: An information theoretic approach," *Neural Networks,* vol. 7, no. 1, 1994, pp. 129-140.

E. B. Bartlett, "A dynamic node architecture scheme for layered neural networks," *Journal of Artificial Neural Networks,* vol. 1, no. 2, 1994, pp. 229-245.

S. Haykin, *Neural Networks: A Comprehensive Foundation.* New York: MacMillan, 1994.

M. Ishikawa, "Structural learning with forgetting," *Neural Networks,* vol. 9, no. 3, 1995, pp. 509-521.

C. G. Carmichael, *Experiments in advancing supervised-learning ANN techniques.* M.S. Thesis. Iowa State University, Ames, Iowa, 1997.

Tables and figures

Table I. Method performance on KV electric futures

CV Type	CV %	Training Method	MEM RMS	MEM R^2	CV RMS	CV R^2	Test RMS	Test R^2
Random	25	SCG	1.428	0.143	1.177	0.029	3.269	0.045
Random	25	DNAF	0.627	0.835	0.738	0.562	4.165	0.001
Random	25	NFD	1.199	0.276	1.752	0.118	3.504	0.039
Random	35	SCG	1.227	0.116	1.733	0.005	3.313	0.001
Random	35	DNAF	1.055	0.346	1.278	0.43	3.839	0.074
Random	35	NFD	1.119	0.763	1.291	0.043	3.504	0.004
Last N	25	SCG	1.411	0.163	1.436	0.014	3.327	0.007
Last N	25	DNAF	1.317	0.271	0.848	0.524	3.531	0.03
Last N	25	NFD	1.551	0.052	1.115	0.067	3.513	0.027
Last N	35	SCG	1.545	0.001	1.746	0.014	3.369	0.053
Last N	35	DNAF	1.057	0.344	1.397	0.312	3.209	0.081
Last N	35	NFD	1.148	0.227	1.51	0.225	3.045	0.195

Table II. Method performance on MW electric futures

CV Type	CV %	Training Method	MEM RMS	MEM R^2	CV RMS	CV R^2	Test RMS	Test R^2
Random	25	SCG	1.036	0.107	0.832	0.076	1.101	0.006
Random	25	DNAF	0.822	0.438	0.653	0.449	1.203	0.004
Random	25	NFD	0.982	0.203	0.992	0.03	1.267	0.079
Random	35	SCG	0.925	0.002	1.251	0.081	1.031	0.012
Random	35	DNAF	0.525	0.65	1.016	0.387	1.312	0.008
Random	35	NFD	0.979	0.182	1.004	0.028	1.246	0.089
Last N	25	SCG	1.039	0.103	0.952	0.002	1.171	0.068
Last N	25	DNAF	1.142	0.013	0.765	0.211	1.159	0.001
Last N	25	NFD	0.971	0.216	0.976	0.001	0.966	0.103
Last N	35	SCG	0.839	0.104	1.257	0.055	1.133	0.049
Last N	35	DNAF	0.911	0.025	1.163	0.255	1.075	0.021
Last N	35	NFD	0.801	0.235	1.255	0.038	0.979	0.095

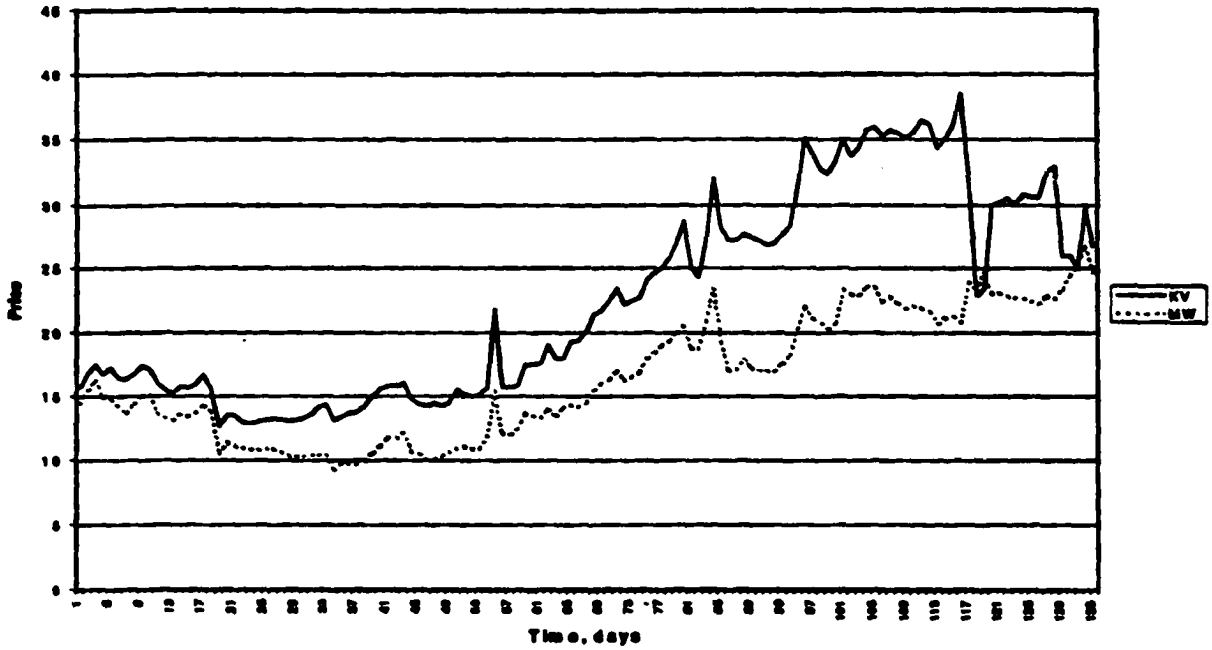


Figure 1. Price vs. time for electric futures contracts

APPENDIX B. RISK FORECASTING WITH NETWORK DIVERSITY OPTIMIZATION

A paper published in the Thirty-Fifth Annual Report of the Electric Power Research Center / Power Affiliate Research Program, April 1999

Craig Carmichael, Gerald Sheble', and Eric Bartlett

Department of Electrical and Computer Engineering,

Iowa State University, Ames, IA, 50011

Abstract

This paper describes a quantitative method of combining artificial neural networks (ANNs) to forecast risk (uncertainty) via network diversity optimization. Committee Machines and Gated Networks [Haykin 1999; Freund and Schapire, 1996] also address the question of how to combine multiple network models, but model diversity is not assessed. The current method, called network diversity optimization (NDOP), introduces an estimate for the diversity between each pair of trained networks. NDOP then uses this estimate to combine the pairs to produce a forecast distribution. Even with limited data, the current method can combine a very large number of ANN models without a loss in generalization, which is partially due to a minimal number of free parameters optimized during the stacking process. NDOP is a common sense, yet general-purpose solution to modeling uncertainty for a large class of risk-related problems.

Introduction

Error forecasting is important whenever risk management is involved. Risk means uncertainty. Uncertainty means error. Managing risk implies managing error. The only way to manage error effectively is to model it, and this usually starts with some random distribution that appears Gaussian. The simplest possible way to model uncertainty may be to bring up a spreadsheet program and compute the standard deviation of the model errors over the whole forecast set. This, however, is so trivial that it provides no useful information for typical risk-management scenarios. In a trading system, for example, no one wants to lose and no one wants to risk more than they have to for a given return. Since just about anyone can calculate the standard deviation over any given set of data, applying the trivial model will likely not result in any advantage. Better error forecasting than this trivial model is surprisingly difficult however.

Starting from common sense, how do humans assess uncertainty in a variety of situations with apparently little effort? At least part of the answer lies in diversity. If a group of students is asked to solve a calculus problem, usually the best way to get the right answer is for each one of them to solve the problem independently and then to compare answers later. If all the answers are the same and if all the students are not clones of one another, then the group of students will most likely be very sure that the group's answer is right. A single student who is motivated to be very certain of his/her answer for the same calculus problem may try to solve it in more than one way using multiple approaches in much the same as the group of students did.

Simply stated, as more diverse models approach the same answer, uncertainty decreases. If a model is diverse from the rest yet consistently yields terrible answers, then its

diversity is irrelevant. Of course it's easy to give different answers than a good model if you're a bad model. Unfortunately, bad models add little useful information, and thus cannot help the stacking (model combining) process. If multiple models yield the same answer, yet they are functionally equivalent, then the fact that they're saying the same thing is irrelevant with regards to uncertainty. Of course they're saying the same thing. They're the same models. Even if the internal parameters of the models appear very different, as long as the models are extracting information from the input space the same v ay and yielding the same output, they are functionally equivalent. Since model performance and diversity are the key issues here, then the only unknown at this point is a calculation for diversity. Restricting the class of models to artificial neural networks, the following sections will provide an estimate for diversity between each pair of ANN models. The network diversity optimization procedure will then be introduced as a means for stacking networks to yield a forecast distribution that includes a forecast and the forecast's uncertainty (risk).

Artificial neural networks: An introduction

Artificial neural networks (ANNs) are modeling techniques that simulate brain-like behavior. Figure 1 shows the typical architecture of an ANN having one hidden layer. Each ANN takes a set of examples in the form (inputs $\vec{x}$, desired output y_D), and for each example provides a nonlinear transform to approximately map the example's input vector to the example's desired output. The weight vector w determines how well the ANN maps this set of examples, and $\vec{w}$ is determined during an iterative training process. A change in the initial starting point $\vec{w}_0$, or a change in the examples that are presented to the network's training algorithm will cause a change in $\vec{w}$. In other words, by varying the starting point or

the set of examples, a variety of networks can be trained. Once trained, each pair of networks can be tested for approximately how different they are from each other, storing the results in a diversity matrix. Then the collection of networks and its diversity matrix can be used to create a stacked model that yields an output and an estimated uncertainty of the output.

Network diversity optimization

As mentioned above, how a model uses information from the input space relates to how the model functions. Estimating diversity relies on measuring divergence in functionality. So the diversity between two models must be a function of the differences in how the input space is being used by the models. Let an importance vector be defined for each model as a vector consisting of an importance measure for each input. Let network diversity be defined as the sum of the nonlinear differences in the importance vector between two neural network models *A* and *B*. The importance of an input can be estimated by simply replacing the value of the input for each example with that input's average over the example set and calculating how much the total error increased. The percentage increase of the total error for each input can be tabulated in this way and then normalized to [0,1] with the additional constraint that the sum of the importance values adds to one. An importance of zero means that the input is not important at all. An importance of unity means that the input is very important. There are other more sophisticated methods to calculate the importance vector for the input space, but the above description is adequate for the purpose of demonstration here. From this, the diversity can be estimated as follows:

$$\Delta_{AB} = \tfrac{1}{2} \sum_{i=1}^{Ninputs} \left[(1 - C_i) \left| I_{A,i} - I_{B,i} \right| + \tfrac{1}{2} C_i \left| (I_{A,i} + I_{A,j}) - (I_{B,i} + I_{B,j}) \right| \right] \tag{1}$$

where:

Δ_{AB}: diversity between neural models A and B

C_i: maximum absolute value of the linear correlation coefficient between input i and any other input k across the training (example) set, $C_i = MAX(|r_{ik}|, k)$

j: input having the highest magnitude of the linear correlation coefficient with input i

$I_{D,k}$: importance of input k to model D normalized to [0,1] so that $\sum_{k=1}^{Ninputs} I_{D,k} = 1$

It turns out that Δ_{AB} must then also fall within in the range [0,1]. A diversity of zero between models A and B means that the networks are functionally equivalent even if their internal parameters are dissimilar.

A diversity of one means that models A and B are very different. This estimate of pair-wise diversity between models A and B can then be used to solve for the stacked (combined) model output and error estimate for each example pattern p.

$$Y_p = \sum_{AB} \frac{\Delta_{AB} (g_A y_{A,p} + g_B y_{B,p})}{g_A + g_B}$$

(2)

$$\sigma_p = \alpha\left(\sum_{AB} \Delta_{AB} g_A g_B \mid y_{A,p} - y_{B,p} \mid\right) + \beta$$

(3)

where:

g_D: goodness of model D over the training set, for example $g_D = r_D^2$, the square of the linear correlation coefficient between the actual output of model D and the desired output across all example training patterns

$y_{D,p}$: output of model D for example pattern p

α, β: free parameters fit during the stacking process, optimized over another example (cross-validation) set

Let the uncertainty component u_{AB} be equal to the term $\Delta_{AB} g_A g_B$. This is simply a factor of how much uncertainty is introduced by the level of disagreement between the outputs of models A and B. If $u_{AB} = 0$, then the level of disagreement $\mid y_{A,p} - y_{B,p} \mid$ of model pair AB contributes nothing to the error forecast σ_p for novel pattern p. This implies that A and B are functionally the same model (clones) or at least one of the models consistently yields terrible answers. Clones and very poor models contribute nothing to the overall estimate of uncertainty (risk), which agrees well with intuition.

Example: Spot price forecasting of electric energy

Finally, an example application is given in this section. Electric load (power demand, in MW) was modeled as a function of time *t* and weather(*t*) over a span of 600 hours. Figures 2-4 show NDOP's test set performance in forecasting the load uncertainty LU. Figure 2 shows the actual magnitude of the error, | actual load – modeled load | ("DesiredE"), plotted vs. time against the modeled error bound of "2xModelSigma". If the forecasted load uncertainty increased by more than 30% in one hour, an increase in actual load uncertainty was expected to occur within that hour also. Figure 3 shows those results plotted as a series. Figure 4 shows the same results plotted as a scatter plot. Figures 3 and 4 demonstrate that sharp increases in the modeled uncertainty hold some correlation with sharp increases in the actual error.

The spot price ($/MW) for electric energy can be estimated from the power demand. This relationship was estimated by using Lagrangian relaxation. Figures 5 and 6 were obtained by feeding the results from the power demand curves through this non-linear transformation. The price per MW forecasted is shown in Figure 5 and the price uncertainty of the forecasts is shown in Figure 6. The "Ylow($)" and "Yhigh($)" bounds in Figure 5 are the model equivalents of one standard deviation from the model output "Y($)". The true bounds are closer to thrice this deviation, as expected.

How does NDOP affect the bottom line in this case? A more accurate forecasted price distribution generates more potential for (trading) profit and less exposure to risk than a less accurate forecast of the same. The network diversity optimization procedure yields a non-trivial, yet fairly straightforward result that is better than simple statistical techniques.

Conclusions

Applying network diversity optimization to a collection of trained neural networks results in a committee-like prediction and an assessment for how accurate the prediction is likely to be. NDOP can result in good generalization because there are only two free parameters used during the stacking process, α and β. It is based on a common-sense approach to uncertainty and risk management. NDOP is limited to the goodness of the models and the level of diversity between each pair of models. Even if the models have high goodness values, a lack of diversity (the clone scenario) will result in a poor measure of uncertainty. Nevertheless, NDOP may be an effective model for risk-assessment whenever diversity can be achieved along with a reasonable goodness of the models. NDOP was applied to electric load forecasting and then fed through a nonlinear transformation to arrive at a forecasted price distribution for electric energy. Preliminary results show good correlation between the load/price absolute error and the NDOP-modeled load/price uncertainty.

References

Bartlett, E. B. (1994). Dynamic Node Architecture Learning; An Information Theoretic Approach. *Neural Networks, 7(1),* 129-140.

Bartlett, E. B. (1994). A Dynamic Node Architecture Scheme for Layered Neural Networks. *Journal of Artificial Neural Networks,* 1(2). 229-245.

Haykin. S. (1994). *Neural Networks: A Comprehensive Foundation.* New York; MacMillan.

Moller, M. F. (1993). A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. *Neural Networks.* 6(3). 525-533.

Carmichael, C. G., "Experiments in Advancing Supervised-Learning ANN Techniques." M.S. Thesis. Iowa State University, Ames, 1997.

Keehoon, K. "Reliability Assessment of Nuclear Power Plant Fault-Diagnostic Systems Using Artificial Neural Networks," Ph.D. Dissertation Iowa State University. Ames, 1994.

Kim. K. and E. B. Bartlett, "Nuclear Power Plant Fault Diagnosis Using Neural Networks with Error Estimation by Series Association," IEEE Transactions on Nuclear Science. Vol. 43, No. 4. p.p. 2372-2388, August 1996.

Kim K. and E. B. Bartlett, "Error Estimation by Series Association For Neural Network Systems," Neural Computation, Vol. 7, No, 4, p.p. 799-808. 1995.

Klimasauskas, C. C., "Neural Nets Tell Why", Dr. Dobb's Journal, p.p. 16-24. 78-84, April 1991.

Li, K., "From Stein's Unbiased Risk Estimates to the Method of Generalized Cross Validation." The Annals of Statistics. Vol. 13, No. 4, p.p. 1352-1377. 1985.

Sridhar, D. V., "Process Modeling Using Stacked Neural Networks," Ph.D. Dissertation, Iowa State University. Ames, 1996.

Sridhar, D.V., E. B. Bartlett and R. C. Seagrave, "Improving Neural Network Based Predictive Modeling Using Stacked Generalization", Intelligent Engineering Systems Through Artificial Neural Networks: Vol. 5, St. Louis. Missouri, American Society of Mechanical Engineers, November, 1995a.

Sridhar. D.V., E. B. Bartlett and R. C. Seagrave, "Application of Stacked Generalization for Neural Network Based Dynamic Modeling of a Chemical Process," World Congress on Neural Networks. International Neural Network Society. Washington D.C., Vol. 2 pp. 66-69. July, 1995b.

Werbos, P.J., "The Roots of Backpropagation", John Wiley k Sons, Inc., New York, 1994.

Wolpert, D. H., "Stacked Generalization," Neural Networks, Vol. 5(2), pp.241-259, 1992.

Figures

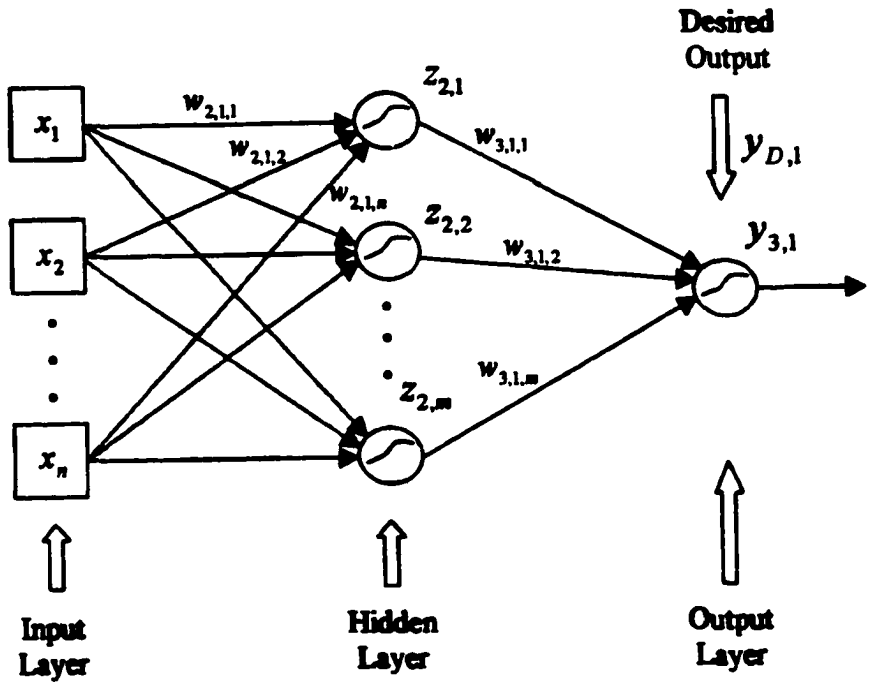


Figure 1. Typical architecture of a single hidden layer neural network

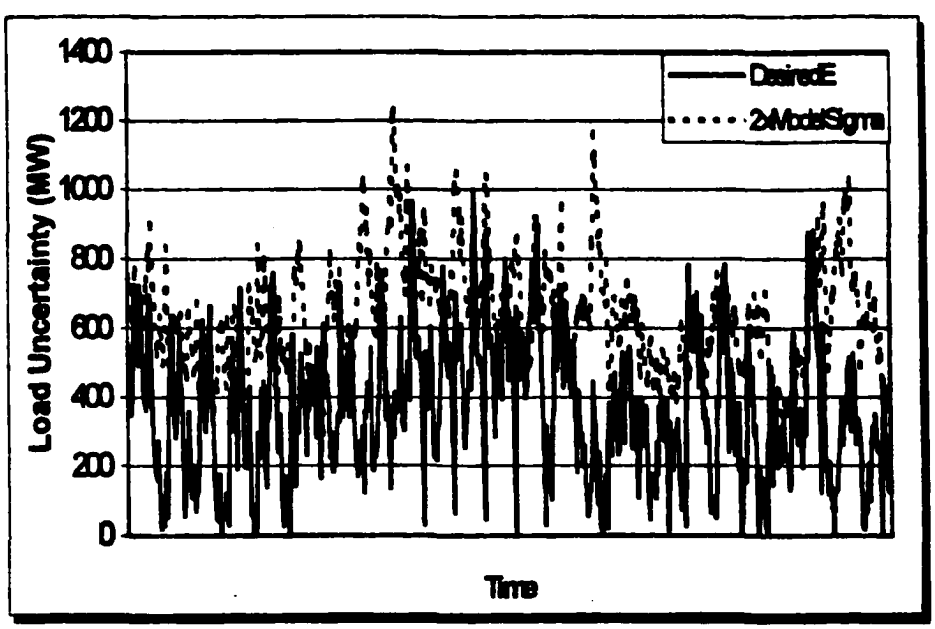


Figure 2. Load uncertainty vs. time for the desired |error| and 2 x the modeled sigma

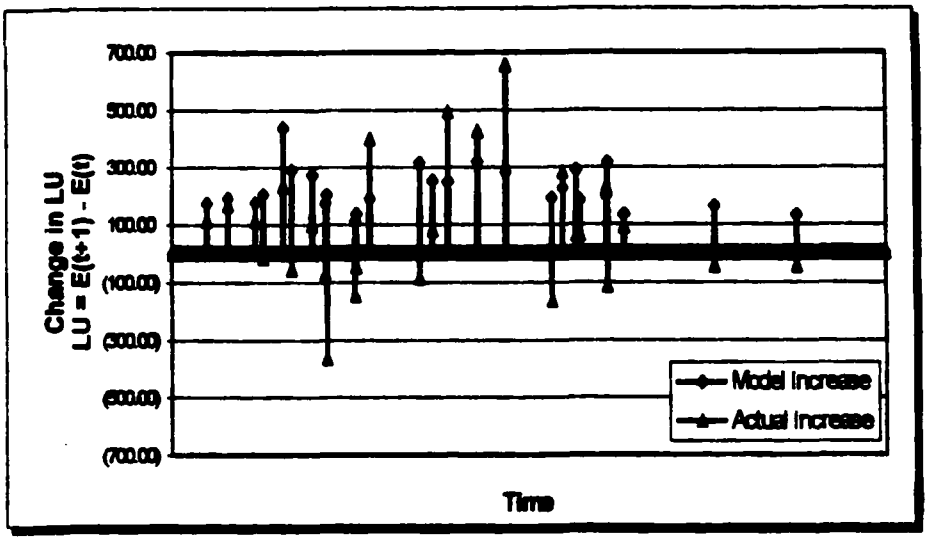


Figure 3. Forecast accuracy of 1 hour ahead changes in load uncertainty for modeled d(LU)/dt > 30%

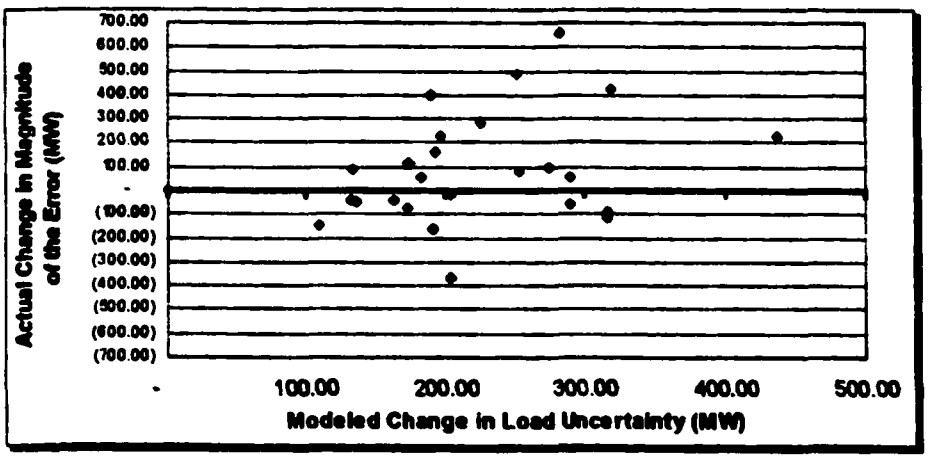


Figure 4. Scatter plot of Figure 3

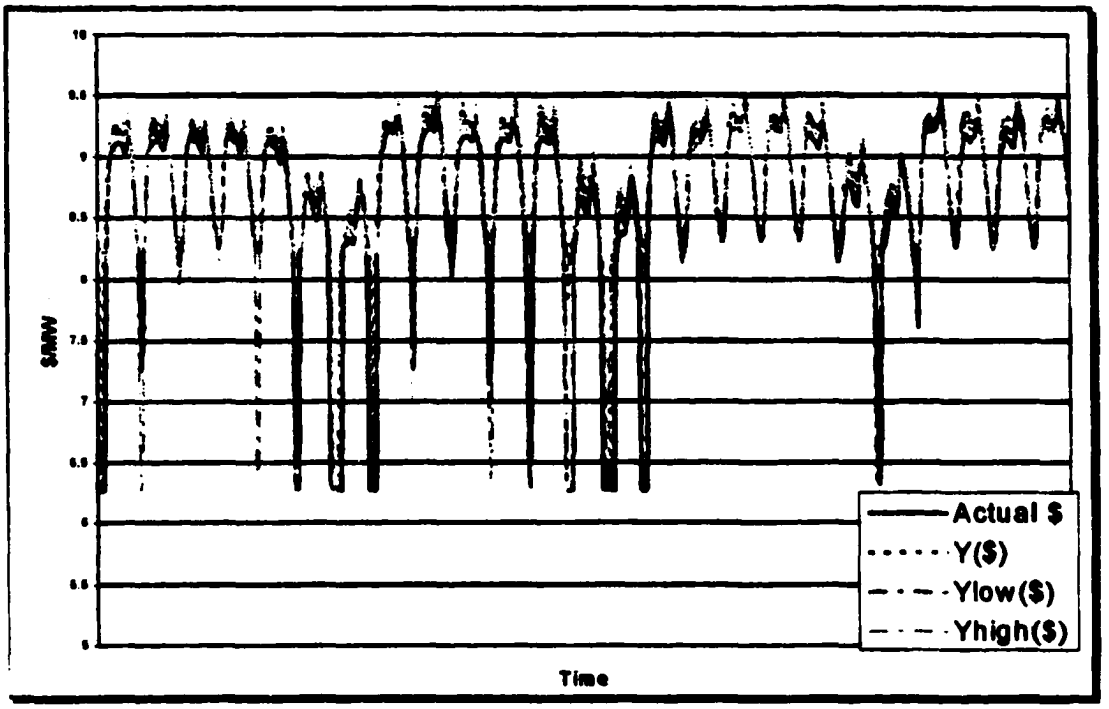


Figure 5. $/MW forecasted as a function of time t and weather(t)

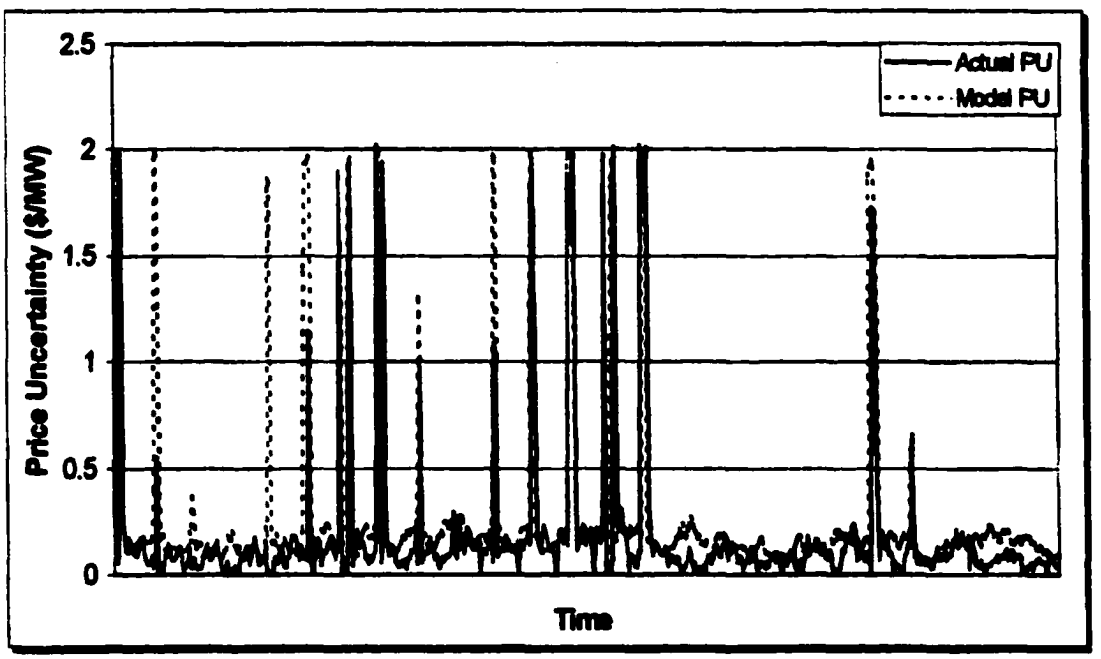


Figure 6. $/MW uncertainty forecasted as a function of time t and weather(t)

APPENDIX C. IMPROVING FORECASTING EFFORTS USING ARTIFICIAL NEURAL NETWORKS

A paper accepted by Artificial Neural Networks In Engineering (ANNIE, 2001)

Bradley Temeyer, William Gallus, Christopher Anderson,
Department of Atmospheric and Geological Sciences,

Craig Carmichael and Eric Bartlett,
Department of Electrical and Computer Engineering,
Iowa State University, Ames, IA, 50011

Abstract

A neural network was devised by Hall et al. (1999) improving weather prediction of probability of precipitation (PoP) and quantitative precipitation amount forecasts (QPF) in the Dallas-Forth Worth, TX region. Their results indicated a strong potential for improving precipitation forecasting in the Southern Plains, but questions remained over the applicability of their approach to other parts of the United States.

Influences on storm systems in the Midwest differ from those in Texas. In this study, we hope to expand on the list of meteorological parameters studied in the past by Hall et al. (1999) for warm season precipitation events by studying events between May 1st and September 30th. Our goal is to obtain similar or better results in precipitation forecasting as were obtained in other studies.

Introduction

Accurate forecasts of rainfall amount are important in forecasting hydrometeorological events, like flash flooding. Quantitative precipitation forecasts (QPF) are provided by numerical weather models as one of the output parameters, but predicting the amount of precipitation that would fall over a certain area in a specified time period has proven to be difficult (Fritsch et al. 1998) from these models, which generally use finite difference forms of the governing atmospheric equations solved on a three-dimensional grid. Possibly because of the complex nature of precipitation processes, in which rainfall gradients can vary significantly over small distances, significant gains in estimating the QPF in summertime events have not been obtained through the model output despite increases in model resolution and complexity. In addition, convection is often times initiated by mesoscale features, which are often accompanied by subtle features that the models have a hard time capturing. Much of summertime precipitation is associated with convection, which contains mesoscale processes that are too small to be resolved adequately with the higher resolution grid models.

Neural networks have been tested in some regions across the United States to aid in estimating QPF. Their use is a significant departure from normal meteorological procedure, which is to rely heavily on the physically and dynamically-based prognostic grid point models. In a study by Hall et al. (1999), a neural network was trained with observations from a network of rain gauges to estimate the probability of precipitation (PoP), and the average precipitation amount that would fall over the Dallas - Fort Worth area in a 24-hour period. In a similar study, by Kuligowski and Barros (1998), a backpropagation neural network was

used to estimate the amount of precipitation in the Mid-Atlantic Region for four different time periods within a 24-hour period.

These two studies serve as a motivation for this project. In the research presented here, a neural network was used to predict the amount of precipitation that would fall in a 24-hour period in a Midwest region, specifically in and around Omaha, NE. This study used the output from the Eta Model from the National Centers for Environmental Prediction (NCEP). The Eta Model was also used in the study by Hall et al. It also made use of information gathered from upper air soundings from Omaha. This study differs from prior studies because it is in a midwestern setting, where the primary influences on the precipitation process can differ from other regions. One of these differences is the presence and importance of the low level jet. The Southern Plains are near the source region for the low level jet, which causes lower level divergence in this area (Bluestein 1993). However, in the Upper Midwest, the area experiences low-level convergence at the head of the low level jet, which can lead to convection initiation. The low level jet also aids in providing moisture to the Midwest. In this region, a maximum in the amount of precipitation that falls occurs during the nighttime hours when the low level jet is the strongest. Additional errors in the models occur because most nocturnal events consist of elevated convection (thunderstorm updrafts originate 1-2 km above ground instead of near the surface; see Bluestein for a general discussion), and model convective parameterizations assume surface-based convection (e.g., Kane and Fritsch 1993). Compared with the southern Plains, there may also be stronger flow aloft in the Upper Midwest because of its closer proximity to the summer position of the upper level jet stream, causing this area to have a greater amount of wind shear. Two types of weather systems called Mesoscale Convective Systems (MCS) and

Mesoscale Convective Complexes (MCC) are fairly prevalent in the Upper Midwest, whereas these types of systems are not as common in the Southern Plains during summer (Maddox 1983, Cotton and Tripoli 1989). Because of these different influences on the precipitation process, numerous parameters were explored in the present study to attempt to determine the best parameters to aid in predicting precipitation. The data set used, the parameters explored, and results will be discussed in further detail in the upcoming sections of this paper.

Data set

Precipitation gauges from the Omaha area were used in this study. Precipitation verification data were obtained through information from the National Climatic Data Center (NCDC). Precipitation gauges located in northwest Omaha (Valley National Weather Service office) and Omaha-Epply Airport were averaged to determine the amount of precipitation that fell in the Omaha area. When available, precipitation information from two surrounding towns, Oakland, Iowa and Glenwood, Iowa, were also used in this study. The predictand for this study was the 24-hour precipitation amount for the Omaha area. The predictors used in this study were variables obtained through weather balloon (upper air) data and the 12z Eta model output for the Omaha area.

The present study focused on forecasts of summertime precipitation, since this is when the models have the most trouble forecasting precipitation, possibly because of the large spatial variability in extreme precipitation events (Smith and Bradley 1994).

The current data set spans the periods from June 1st 1998 through July 31st 1998 and May 1st 1999, through September 30th, 1999, a total of 214 days. Because of missing upper

air data and Eta model information, 47 days were omitted from the final data set. The sample size is currently limited, so there is ongoing research to expand this data set. When upper air data were unavailable for certain parameters, Eta model 00 hour forecast information (the model initialization) was used. Because the initialization is heavily based upon actual observations, this substitution should not pose a problem.

Parameters dealing with the physical properties of precipitation formation including water-vapor distribution, vertical velocity, and temperature profiles in the atmosphere were examined. These parameters are similar to the those explored in Hall et al. (1999) with the exception of a few parameters that were added in the present study. Parameters from the 12z Eta model output similar to those in the Hall et al. study included:

1. Precipitable Water (00 hour forecast)
2. Precipitable Water Change (24 hour change in PW)
3. 850 mb Theta-E (00 hour forecast)
4. 850 mb Theta-E Advection (12 hour forecast)
5. 850 mb Wind (speed and direction) (00 hour forecast)
6. 850 mb Mixing Ratio (12 hour forecast)
7. 700 mb Theta-E (00 hour forecast)
8. 700 mb Vertical Velocity (12 hour forecast)
9. 700 mb Temperature Advection (12 hour forecast)
10. 700 mb Wind (speed and direction) (00 hour forecast)
11. 500 mb Wind (speed and direction) (00 hour forecast)
12. 500 mb Vorticity (12 hour forecast)

13. 850-300 mb Thickness (12 hour forecast)
14. 850-300 mb Differential Divergence (12 hour forecast)
15. 1000-850 mb Moisture Divergence (12 hour forecast)
16. 250 mb Divergence (12 hour forecast)
17. 700- 500 mb Lapse Rate (00 hour forecast)
18. K-index (upper air sounding)

Other parameters examined that were not part of the Hall et al. study included

1. Lifted Index (LI) (upper air sounding)
2. Convective Available Potential Energy (CAPE) (upper air sounding)
3. Convective Inhibition (cin) (upper air sounding)
4. 850 mb Divergence (12 hour forecast)
5. 850 mb Height (12 hour forecast)
6. 700 mb Mixing Ratio (12 hour forecast)
7. 700 mb Temperature (00 hour forecast)
8. 500 mb Temperature (00 hour forecast)
9. 300 mb Divergence (12 hour forecast)
10. 300 mb Height (12 hour forecast)

Results

The data were trained on a neural network with a single hidden layer, and a 5 hidden node multiplayer-perceptron with a hyperbolic tangent transfer function. The test set was

comprised of 25 percent of the total number of days used. Work is currently continuing to expand the training and test set. Two types of tests were run to test the importance of each variable in the model. The first test for usefulness made use of a smooth sensitivity analysis in normalized space. The second test measured the change in the root mean squared (RMS) error when a variable was left out of a model run. Both of these tests were applied to each variable. To test for importance, the neural network was trained on normalized data. Next, the neural network was probed in an attempt to find inputs that were most important. Next, each input was deleted, and the average value for the input was substituted into the model. The increased error was then calculated and stored as a result. Using the sensitivity analysis, the flowing variables were determined to be significant:

1. 850 mb Wind direction (00 hour forecast)
2. Convective Inhibition (cin) (upper air sounding)
3. Precipitable water (00 hour forecast)
4. K index (upper air sounding)
5. 500 mb Temperature (00 hour forecast)
6. 500 mb Vorticity (12 hour forecast)
7. 700 mb Mixing ratio (12 hour forecast)
8. 500 mb Wind direction (00 hour forecast)

This can be seen by the slight decrease in importance in Figure 1. Applying the second test for importance, the following variables were determined to be important:

1. 500 mb Temperature (00 hour forecast)
2. 850 mb Wind direction (00 hour forecast)
3. 300 mb Height (12 hour forecast)
4. Precipitable Water (00 hour forecast)
5. K index (upper air sounding)
6. 850 mb Theta-E advection (12 hour forecast)
7. 500 mb Vorticity (12 hour forecast)
8. Convective Inhibition (cin) (upper air sounding)
9. 850 - 300 Differential Divergence (12 hour forecast)
10. 700 mb Temperature advection (12 hour forecast)
11. 700 Vertical velocity (12 hour forecast)

This can be noted by the decrease in importance in Figure 2. When these results were combined, the following variables were determined to be important as seen in Figure 3:

1. 500 mb Temperature (00 hour forecast)
2. 850 mb Wind Direction (00 hour forecast)
3. Precipitable Water (00 hour forecast)
4. Convective Inhibition (cin) (upper air sounding)
5. K index (upper air sounding)
6. 500 mb Vorticity (12 hour forecast)
7. 850 mb Theta-E advection (12 hour forecast)
8. 300 mb Height (12 hour forecast)

9. 850-300 mb Differential Divergence (12 hour forecast)

10. 700 mb Temperature advection (12 hour forecast)

Because of the importance of nonlinear effects in the atmosphere, it can be difficult to explain directly the role that each of the above terms plays. However, some explanations can be offered to support the above results. For instance, when the air at 500 mb is unseasonably warm, more moisture can be held at this level. Warm temperatures at 500 mb are often associated with tropical weather systems, which often carry much more moisture than the average air mass. The importance of the direction of the 850 mb wind could possibly be related to the low level jet mentioned earlier (which generally supplies moisture to the Upper Midwest from a south or southwest direction). The amount of precipitable water may be important because it is a measure of how much liquid is present in a vertical column of air. Of course as the amount of water in a vertical column increases, the amount of liquid that could precipitate out also increases.

Over the training set, the model had a correlation coefficient of 0.56. Similarly, the test set had an overall r-squared value of 0.45. Combined, the overall correlation coefficient was 0.52. While these values may seem small, it should be noted that warm season precipitation prediction is extremely difficult, and traditional methods generally have very low skill scores (e.g., Gallus and Segal, 2001). For instance, the equitable threat score for the month of August 2000 for the entire United States in two commonly used meteorological models was only around .10 (Baldwin, Storm Prediction Center, 2001, personal communication). It is possible that skill scores using our current technique would be

comparable or even better. The computation of these scores will be performed in the future using the neural network discussed in this paper.

Conclusions

A neural network-based precipitation forecasting system similar to that shown by Hall et al. (1999) to improve forecasting of rainfall in the southern Plains of the United States is being tested in a portion of the Midwest. Differences in the relative importance of parameters used in the scheme versus those used in the Hall et al. study may reflect differences in the primary rainfall-forcing mechanisms important in these two different regions. It is hoped that the current technique will improve rainfall forecasting over the Midwest. Further work to expand the data set and to analyze the results is ongoing. These results will be presented at the upcoming conference.

References

Brooks, H.E. and D.J. Stensrud, 2000: "Climatology of Heavy Rain Events in the United States from Hourly Precipitation," Monthly Weather Review, Vol. 128, pp. 1194-1201.

Bluestein, H.B, 1993: Synoptic-Dynamic Meteorology in Midlatitudes Volume II Observations and Theory of Weather Systems, Oxford University Press, New York, pp. 430-431, 544-545.

Cotton, W.R., and Tripoli G.J., 1989: "Precipitation Life Cycle of Mesoscale Convective Complexes over the Central United States," Monthly Weather Review, Vol. 117, pp. 784-808.

Gallus, W.A., Jr., and M. Segal, 2001: "Impact of Improved Initialization of Mesoscale Features on Convective System Rainfall in 10 km Eta Simulations," Weather and Forecasting, Vol. 16, (accepted).

Fritsch, J.M., and Coauthors, 1998: "Quantitative Precipitation Forecasting: Report of the Eighth Prospectus Development Team, U.S. Weather Research Program," Bulletin of American Meteorological Society, Vol. 79, pp. 285-299.

Hall, T., Brooks H.E., and Doswell III C.A., 1999: "Precipitation Forecasting Using a Neural Network," Weather and Forecasting, Vol. 14, pp. 338-345.

Kain, J.S., and Fritsch, J.M., 1993: "Convective Parameterization for Mesoscale Models: The Kain – Fritsch Scheme. The Representation of Cumulus Convection in Numerical Models," K.A. Emanual and D.J. Raymond, Eds., American Meteorological Society, 246 pp.

Kuligowski, R. J., and A. P. Barros, 1998: "Localized Precipitation Forecasts from a Numerical Weather Predicition Model Using Artificial Neural Networks," Weather and Forecasting, Vol. 13, pp. 1194-1204.

Maddox, Robert A., 1980: "Mesoscale Convective Complexes," Bulletin American Meteorological Society, Vol. 61, pp. 1374-1387.

Maddox, Robert A., 1983: "Large – Scale Meteorological Conditions Associated with Midlatitude, Mesoscale Convective Complexes," Monthly Weather Review, Vol. 111, pp. 1475-1493.

Smith J.A. and A.A. Bradley, 1994: "The Space Time Structure of Extreme Rainfall in the Southern Plains," Journal of Applied Meteorology, Vol. 33, pp 1402-1417.

Figures

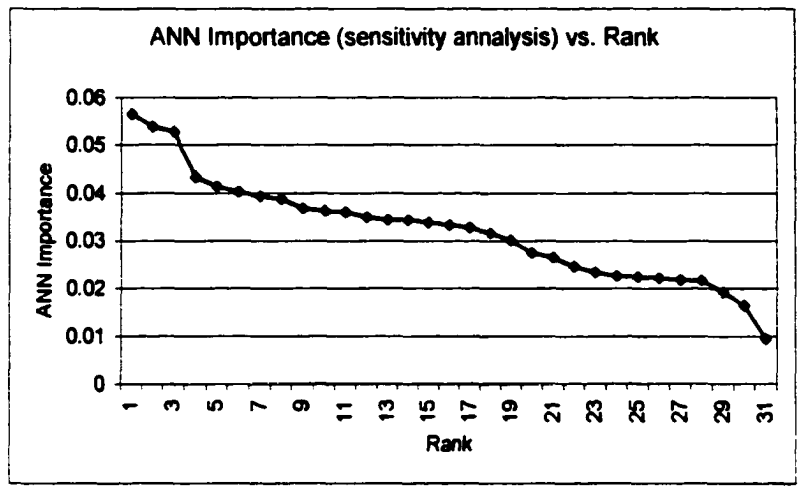


Figure 1: Importance rank for sensitivity analysis test

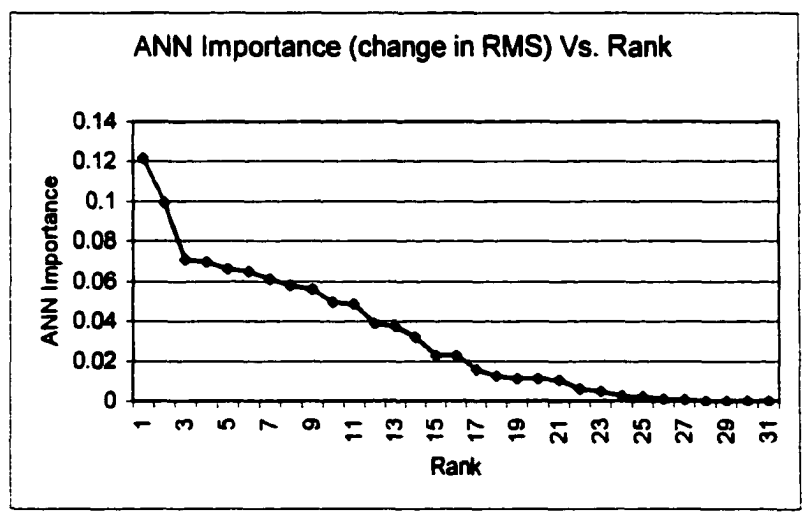


Figure 2. Importance rank for change in RMS test

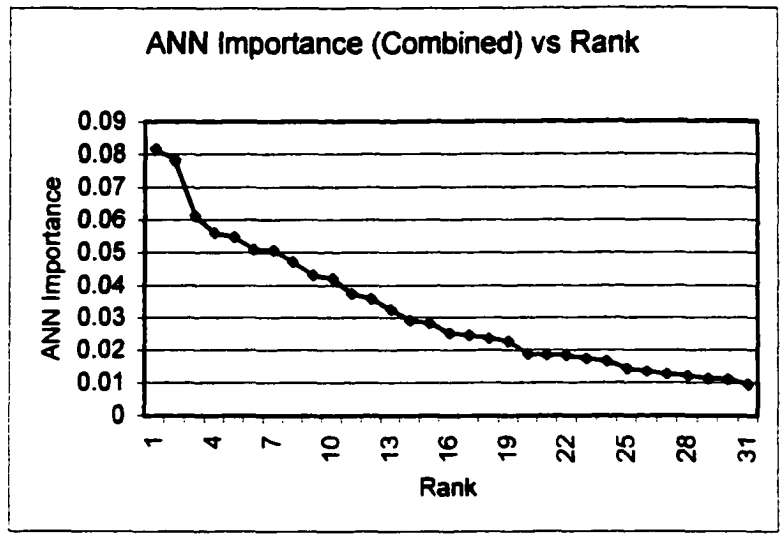


Figure 3. Combined importance test

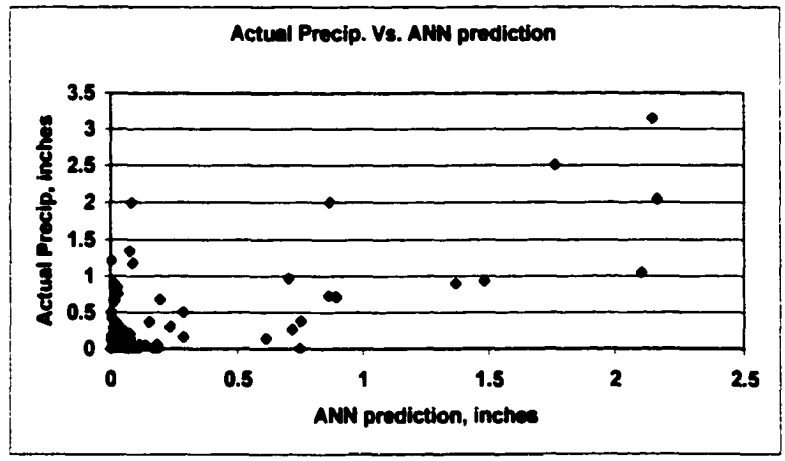


Figure 4. Precipitation predicted by neural network versus actual measured precipitation