ESTIMATION OF RESERVOIR PROPERTIES FROM SEISMIC DATA BY SMOOTH NEURAL NETWORKS

Muhammad M. Saggaf and M. Nafi Toksöz

Earth Resources Laboratory
Department of Earth, Atmospheric, and Planetary Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139

Husam M. Mustafa

Geophysical Research and Development
Saudi Aramco
Dhahran, Saudi Arabia

ABSTRACT

Traditional joint inversion methods require an *a priori* prescribed operator that links the reservoir properties to the observed seismic response. The methods also rely on a linearized approach to the solution that makes them heavily dependent on the selection of the starting model. Neural networks provide a useful alternative that is inherently nonlinear and completely data-driven, but the performance of traditional back-propagation networks in production settings has been inconsistent due to the extensive parameter tweaking needed to achieve satisfactory results and to avoid overfitting the data. In addition, the accuracy of these traditional networks is sensitive to network parameters, such as the network size and training length. We present an approach to estimate the point-values of the reservoir rock properties (such as porosity) from seismic and well log data through the use of regularized back propagation and radial basis networks. Both types of networks have inherent smoothness characteristics that alleviate the nonmonotonous generalization problem associated with traditional networks and help to avert overfitting the data. The approach we present therefore avoids the drawbacks of both the joint inversion methods and traditional back-propagation networks. Specifically, it is inherently nonlinear, requires no *a priori* operator or initial model, and is not prone to overfitting problems, thus requiring no extensive parameter experimentation.

INTRODUCTION

Although seismic data have been used for a long time to delineate the structure of petroleum reservoirs, quantitative amplitude analysis to estimate the spatial distribution of the reservoir rock properties has gained momentum only recently. The advent of precise logging instruments has enabled well log data to provide a very accurate description of the reservoir properties. Unfortunately, this knowledge is available only at the well locations, which usually have a very sparse coverage of the field. Because well drilling is such an expensive endeavor, it is not commercially practical to drill anything but the most essential wells needed to develop the reservoir. Seismic data, on the other hand, is much less expensive to acquire and can be gathered economically over the entire field. The problem, however, is that seismic data provide only an indirect sampling of the rock properties. Therefore, correlating well log and seismic information can provide a natural and convenient way of extending the knowledge of the reservoir rock properties gained at the well locations to the entire field, thus providing a much more complete description of the reservoir.

Traditional methods tackled this issue by joint inversion of well log and seismic data (Angeleri and Carpi, 1982; de Buyl *et al.,* 1986; Anderson, 1996; Brac *et al.,* 1998). One difficulty with these approaches, however, is that they require an *a priori* prescribed theoretical operator that links the rock properties derived from the well log data to the observed seismic response. This relationship between the individual rock properties and the seismic response is certainly not obvious and is exceedingly difficult to characterize because the seismic response is affected by numerous rock characteristics, such as porosity, pore pressure, lithology, and water content. Also, the way these factors combine to affect the seismic data can vary regionally. The accuracy of joint inversion methods is thus hindered by the need to prescribe an *a priori* theoretical relation that is difficult to determine and that frequently holds only under ideal conditions.

Moreover, even though the relation between rock properties and seismic response is decidedly nonlinear, joint inversion methods have to linearize their approach and solve for the final model by perturbing an initial guess and iteratively updating that model by comparing with the observed response until a satisfactory convergence to the final model is reached. This iterative linearization makes the performance and accuracy of the entire procedure heavily dependent on the choice of the initial model. Three potential sources of inaccuracies exist in these joint inversion methods: the selection of the *a priori* operator, linearizing the nonlinear relation, and numerical convergence. The latter two are notoriously dependent on the choice of the initial model. There is, therefore, a need for an alternative approach that is robust, inherently nonlinear, and avoids the difficulties associated with the necessity to pick an initial model and an *a priori* operator. Neural networks provide such an alternative.

Neural networks are deeply rooted in biological origins. Cognitive psychology has developed models for human behavior, learning, and perception, and has linked them with brain physiology. (The link between the physical makeup of the brain and human thinking is entirely nontrivial. In fact, several Nobel Prizes have been awarded for research that addressed that relation, such as feature-extraction models of the visual cortex, goal-directed behavioral models, and propagation of electrical signals along neural cells.) Neural networks attempt to mimic the cognitive thought processes of the brain, and they are often used to emulate some of the vision system and mental processes used by humans to analyze data.

The history of neural network computing goes back to the 1940s, when McCulloch and Pitts (1943) introduced the first neural network computing model. In the fifties, the perceptron model of Rosenblatt (1958) became popular. This simple two-layer network was able to successfully learn to classify input patterns by adjusting the weights of the connections between the nodes. At the time, neural networks looked promising, especially as classical artificial intelligence techniques were becoming too cumbersome to handle the ever more challenging applications. Neural networks were thus expected to solve many problems of the time. However, interest declined as limitations in their performance arose. The perceptron model, for example, was shown to be incapable of classifying simple input patterns that belong to nonseparable vector spaces (Minsky and Papert, 1969).

In the early eighties there was a renewed interest in neural networks, fueled not only by disappointments in classical artificial intelligence methods in solving large, complicated systems, but also encouraged by many innovations in neural computing that laid solid foundations for numerous new neural models, such as recurrent networks (Hopfield, 1982) and the back-propagation rule (Rumelhart *et al.*, 1986). These days, neural networks have wide-ranging applications in several fields that include not only banking, electronics, and defense, but also exploration geophysics (Poulton *et al.*, 1992; Wang and Mendel, 1992; McCormack *et al.*, 1993; Boadu, 1998; Calderon-Macias *et al.*, 1998; Liu and Liu, 1998) and petroleum geology (Baldwin *et al.*, 1990; Rogers *et al.*, 1992; Huang *et al.*, 1996; Saggaf and Nebrija, 2000).

One of the major difficulties with neural network methods, despite their promise and success in research environments, is that in practice their performance has been inconsistent due to the significant amount of parameter tweaking required to achieve satisfactory results. Although they can be made to perform well, this is often only after extensive experimentation with their parameters. In fact, utilizing neural networks has sometimes been described as an art that one develops a feel for slowly. Although this may be acceptable in research settings, it is certainly not so in production environments, where time is critical and staff are less familiar with the inner workings of the methodology.

The selection of optimal neural network parameters can be challenging because traditional neural networks have a nonmonotonous generalization behavior (overfitting, large networks often perform worse than smaller ones in test data). In this paper, we present an approach to estimate the reservoir properties in the inter-well regions of the field by correlating seismic and well log information through the use of neural networks that have inherent smoothness constraints. This smoothness improves the generalization behavior of the network and alleviates the problem of overfitting. This gives rise to a methodology that avoids the difficulties associated with both the traditional joint inversion procedures and traditional back propagation networks that require extensive parameter tweaking and are prone to overfitting. The approach we present here thus has four advantages over traditional methods: (1) it is inherently nonlinear and there is no need to linearize it, so it is adept at capturing the intrinsic nonlinearity of the problem; (2) it is virtually model-free, and no *a priori* theoretical operator is required to link the reservoir properties to the observed seismic response; (3) a starting model is not needed, thus the final outcome is not dependent on the proper choice of that initial guess; and (4) it is naturally smooth, and hence has a more monotonous generalization behavior than traditional neural network methods. In this paper, we describe the background and methodology of our approach, and in a companion paper (Saggaf *et al.*, 2000) we apply the methodology for the estimation of porosity in the inter-well regions of the reservoir.

NEURAL NETWORKS STRUCTURE

Neural networks, in general, are grossly simplified versions of their biological counterparts (Figure 1a), consisting of interconnected nodes (neurons) corresponding to the neural cells in the brain (the human brain has over 50 billion such cells). The nodes respond to input signals by transmitting excitatory and inhibitory signals to other neurons. Thus, the signal is transmitted through the network, and is modified as it travels from one neuron to another. It is the simultaneous operation of those highly interconnected neurons that produces mind-like characteristics that can perform sophisticated analyses.

The nodes in each layer of the network receive input from the nodes of the previous layer and transmit it through to the following layer. As this input propagates through the network, it is acted upon by the weights and transfer function assigned to each layer. A variety of transfer functions (sometimes called activation functions) exist that help give the network its characteristics. Some of the more common ones are the linear, log sigmoidal, and hyperbolic tangent sigmoidal functions. These are defined as follows, respectively:

$$f_l(x) = x \tag{1}$$

$$f_{ls}(x) = \frac{1}{1+e^{-x}} \tag{2}$$

$$f_{ts}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{3}$$

Figure 2 shows example plots of each of these functions. A neural network is characterized by the type of transfer function acting on the output of each layer and linking it to the following layer to which it is connected, in addition to the number of layers in the network and the number of nodes in each layer.

Each network must have an input layer and an output layer. The number of nodes in each of these two layers is dictated by the geometry of the problem at hand. Additionally, the network may also have one or more internal (so-called hidden) layers. The number of these internal layers, the number of nodes in each internal layer, and the type of transfer functions assigned to the internal and output layers represent the network parameters to be selected by the operator. Each layer is often designated by the transfer function assigned to it (acting on its output); for example, a linear layer is one to which a linear transfer function is assigned. It is in this sense that the input layer is sometimes not considered a layer at all, as no transfer function acts on its output and it is assigned no weights; it merely feeds the input directly into the first internal layer. With this notion, then, a network that has an input "'layer," one internal layer with a tangent sigmoidal transfer function, and an output layer with a linear transfer function is considered a two-layer network with a tangent sigmoidal internal layer and a linear output layer. As we discuss below, such a network structure is one of the most widely used. The structure of the network, along with the weights assigned to each layer, completely describe the network.

Neural networks work as model-free point-value estimators for any arbitrary function to be approximated. The process of computing the optimal weights that minimize the error between the observed data and network output is called the training process. A series of sparse data points are fed to the network during the training stage, along with the desired output. After training, the network represents a predictive model of the data and can be used to estimate the data value for any point in the input space (Figure 1b). Training a neural network can thus be viewed as a purely mathematical optimization problem, where the optimal weights are sought so as to minimize the network error. In fact, this is how neural networks are implemented by software (where they are sometimes called artificial neural networks). Several algorithms exist that speed up this optimization (training) step. Once the optimal weights are calculated, the network can be readily applied on nontraining data to derive the prediction output.

Neural networks possess some interesting and advantageous properties that make them suitable for various applications that involve human-like pattern identification, prediction, and analysis (Hrycej, 1992). Some of these properties are:

- **Generalization:** For similar inputs, outputs are similar. The network is able to generalize from the training data, so that its output is similar for input data that are almost, but not quite, the same. In contrast, conventional artificial intelligence has no concept of similarity, since it is based on logic, and the concept of similarity cannot be ascribed to logical rules.

- **Graceful degradation:** A small distortion of the input data produces only proportionally small performance deterioration. In other words, noisy or incomplete input data would not produce considerable deviation in the inference results. Thus, neural networks are well suited to handling geophysical and geological data, where there is an abundance of vagueness, imperfection, incompleteness, and uncertainty.

- **Adaptivity and learning:** Learning is done by example, and the network adapts to changing input. There is no need for manual encoding of rules or prescription of *a priori* operators linking the model data with the observables, as is the case in traditional joint inversion methods. This can be called automatic knowledge extraction; it not only eases and streamlines the task of modeling, but it is also more objective, since it is not affected by human errors and biases induced by manual generation of the inversion operator.

- **Parallelism:** Neurons operate simultaneously, not sequentially, and hence neural networks are well suited for implementations on parallel computers. This is especially useful for large networks.

Undoubtedly, the most important advantage of neural networks is that they represent a model-less approach to inversion, where it is not necessary to specify the underlying physical model. This is very useful in complex problems where the physical relation is hard to identify, or where that relation changes from one group of input data to another.

REGULARIZED BACK-PROPAGATION NETWORKS

Perhaps the most popular type of neural networks are back-propagation networks. These are also sometimes called multi-layer perceptrons or feed-forward networks (back-propagation describes the learning rule of the network, whereas feed-forward describes how the network operates). A typical back-propagation network has one sigmoidal internal layer and a linear output layer. Such a network has been shown to be able to approximate virtually any function of interest as accurately as desired, given a sufficient number of nodes in the internal layer (Hornik *et al.*, 1989). Adding extra internal layers increases the complexity of the network and may allow it to achieve its goal with fewer nodes overall. We shall adhere to two-layer networks in this work, however, as they are much simpler to characterize and build. Mathematically, each node j in layer k is assigned a weight vector that belongs to a vector space of the same dimension as the space of the input to the layer. The output of the layer is computed as the value of the transfer function acting on the inner product of the input vector with the node weights. A scalar that represents an offset (sometimes called the bias) is often added to the product before the transfer function is evaluated. This offset improves the network's ability to fit the data and can be regarded as the equivalent of the y-intercept in linear regression. Thus, denoting the output of the kth layer by $\mathbf{y}_k$, the weight vector assigned to the ith node in the layer by $\mathbf{w}_{k,i}$, and the offset for that node by $b_{k,i}$, then the ith output of that layer (ith element of the vector $\mathbf{y}_k$) is given by:

$$y_{k,i} = f(\mathbf{w}_{k,i} \cdot \mathbf{y}_{k-1} + b_{k,i}) \qquad i : 1 \rightarrow N_k, \tag{4}$$

where f is the transfer function of the layer, N_k is the number of nodes in the layer, and the inner product between the two vectors is performed. For the typical two-layer network we mentioned above, the network output $\mathbf{y}_2$ is therefore given by:

$$\begin{aligned} y_{1,i} &= f_{ts}(\mathbf{w}_{1,i} \cdot \mathbf{x} + b_{1,i}) \qquad i : 1 \rightarrow N_1, & (5) \\ y_{2,i} &= \mathbf{w}_{2,i} \cdot \mathbf{y}_{1,i} + b_{2,i} \qquad i : 1 \rightarrow N_2, & (6) \end{aligned}$$

where $\mathbf{x}$ is the input vector to the network, f_{ts} is the tangent sigmoidal transfer function defined in equation (3), and N_1 and N_2 are the number of nodes in the internal and output layers, respectively.

For a simple network with a single set of connections (i.e., with only an input and an output layer—no internal layers), the weights can be computed by directly solving the network with a standard optimization method (e.g., steepest descent or conjugate gradient). For more complicated networks, the back-propagation rule represents a more efficient way of attaining the optimal weights than the direct solution of the network. In this case, the weights for each layer are obtained separately. In each iteration, the weights for the outermost layer are computed as before (direct solution), since all required information is available there (input and prediction error). For internal layers, however, the prediction error is not available. For these layers, the error is successively propagated back through the outermost layers, and the solution is computed for those internal layer weights (Figure 1c). The name back-propagation stems from this training method, which is also called the generalized delta rule.

As mentioned in the introduction, one of the obstacles that has reduced the usability of neural networks and hindered wider adoption in production environments is the extensive parameter tweaking required to obtain accurate and consistent results. This is because the relative complexity of the network to the complexity of the data is an important variable that affects the performance and generalization behavior of traditional back-propagation networks. A network that is too complex relative to the data results in overfitting: it performs well on the training data set, driving the misfit between the observed data and network output virtually to zero, but tends to produce a rough output that is a poor interpolator between the training points. Thus, the network solution is all but useless as a predictive model. On the other hand, a network that is too simple relative to the complexity of the data would invariably fail to capture the intricacies of the data characteristics and would thus underfit the data.

We illustrate this shortening with an example. We construct a nonlinear model according to the deterministic relation:

$$T(z) = B_1 - B_2 e^{-B_3 z}, \tag{7}$$

where $B_1 = 1$, $B_2 = 2$, and $B_3 = 1$. Noise-free data were generated according to this model for 100 points spanning an interval form 0 to 5. Of these data, a subset of 10 points was selected, and the network was trained on this training subset. After training, the entire data set was recreated by applying the network on the whole data range. The recreated (estimated) data are compared to the true data to gauge the accuracy of the network prediction.

We start with a network that is just large enough to provide an adequate fit to the data. This network has a typical back-propagation structure with a tangent sigmoidal internal layer of two nodes, and a linear output layer. The result is shown in Figure 3a, which compares the actual and estimated data values (the figure also shows the points used to train the network). It can be seen that the estimated and actual data overlap significantly, indicating that the network prediction was quite accurate. This is a good result, especially keeping in mind that unlike model-based inversion, the network had no knowledge of the underlying operator that was used to create the data. The same test was rerun on data that have been encumbered by 10% random Gaussian noise (relative to the mean of the data), and the result is shown in Figure 3b. Again, the fit between the actual and estimated data is excellent.

We next use too large a network—one of a similar structure to the above but with 50 nodes in the internal layer. The result appears in Figure 4a. The network was able to fit the training points quite well (in fact, much better than before). However, the model it produced was so rough that it is a poor interpolator of the data between the training points, and thus the predictive model generated by this network is all but useless. The corresponding result for the data encumbered by 10% noise is shown in Figure 4b, and is equally poor. Next, we give an example in which a network that is too simple for a given problem underfits the data. The same data range described above was used to generate 100 data points according to the relation:

$$T(z) = \sin(B_1 z), \tag{8}$$

where $B_1 = \frac{\pi}{2}$. A ten-point training subset was employed to train a network of a similar structure as discussed above, which has five nodes in its internal layer. The entire data set was subsequently recreated using this network and compared to the actual data. The result is shown in Figure 5a, where it can be seen that the network, which is just large enough to provide an adequate fit to this problem, was quite successful in predicting the data. Figure 5b shows the result of a network that is similar in structure but slightly smaller. This network has just two nodes in its internal layer, and is evidently much too weak to sufficiently fit the data in this case. This network thus produced too smooth a model that severely underfits the data.

Choosing a back-propagation network of the right size, therefore, is an important factor in attaining a reasonable predictive model. The obstacle is that it is difficult to determine beforehand just the right size for the problem at hand. Though the optimal network size can be estimated by systematic cross-validation tests, this can be a very lengthy and compute-intensive process that is not suitable for production settings. The overfitting problem is not limited to neural network methods; it is also encountered often in various applications of function fitting. Choosing too complex a function not only wastes valuable compute resources (a tolerable cost), but also results in a rough solution that is not usable as a predictive model. We next investigate three approaches that can help alleviate the problem.

Regularization

As we mentioned previously, the optimal network weights are computed by minimizing the misfit error between the training data and the network output. The objective function is thus the mean square error of the misfit:

$$OF = \frac{1}{N}\sum_{i=1}^{N} e_i^2, \tag{9}$$

where e_i is the misfit error for the ith training observation and N is the total number of observations in the training data set. The nonmonotonous generalization behavior of the network can be alleviated by modifying the objective function such that not only is the misfit error reduced, but so are the network weights (and offsets):

$$OFR = (1-\lambda)\frac{1}{N}\sum_{i=1}^{N} e_i^2 + \lambda\frac{1}{M}\sum_{i=1}^{M} {w_i}^2, \tag{10}$$

where w_i is the ith network weight (or offset) and M is the total number of weights and offsets in the network.

The modified objective function results in a network that not only fits the data, but also has small weights and offsets that give rise to smaller variations in the network output and thus yield a smoother result in the output space that is less likely to overfit the data. We will call such a network a regularized back-propagation network. Note that, in essence, this approach is not unlike that of Tikhonov regularization (Tikhonov and Arsenin, 1977), but with a slight distinction. Exploration geophysicists are familiar with Tikhonov-style regularization, which smoothes the actual model (e.g., velocity), since the model itself is the desired outcome in many applications (Constable *et al.,* 1987; Sambridge, 1990). However, in our case we are interested in smoothness in the data space (network output) rather than in the model space (network weights), so we constrain the network weights to be small (rather than smooth) to yield a smooth network output. The regularization parameter λ determines the relative emphasis of smoothness versus degree of fit to the training data. A small value of λ results in an unregularized network. On the other hand, if λ is too large, then the network would not adequately fit the data. The optimal smoothing parameter need not be prescribed manually; it can be determined automatically by statistical Bayesian interpolation (MacKay, 1992; Foresee and Hagan, 1997), where the regularization parameter is related to a random distribution of the network weights.

We return now to the overfitting example given earlier. In this case, we use the same large network (50 nodes in the internal layer) as before, but the objective function is modified according to equation (10) to yield a regularized back-propagation network. The regularization parameter was automatically selected by Bayesian interpolation. The result is shown in Figure 6a, where it is evident that the model produced here represents a much better predictor of the data than the traditional back-propagation network of

Figure 4a, and is comparable to the model produced by the traditional network of the optimal size (Figure 3a). In fact, Figures 6a and 3a look almost identical, a testament to the effectiveness of this method (the regularized network was more accurate, as it produced a smaller RMS error, 0.2%, versus 1% for the traditional network of optimal size). Figure 6b shows the result when the regularized back-propagation network was built from the data that have been encumbered by 10% noise. Again, this looks similar to the result obtained by the neural network of the optimal size (Figure 3b), even though the regularized network utilized here was much larger than the optimal.

Utilizing regularized back-propagation networks thus eliminates the need for extensive parameter tweaking and produces networks that have more consistent results than their traditional back-propagation counterparts. The operator need only pick a network of adequate size; there is no danger of overfitting the data if the network was too large. In fact, the operator is encouraged to err on the safe side and overestimate the network size required for a given problem. The only penalty incurred here is slightly increased training time—a tolerable cost given today's compute resources and the general compute-efficiency of neural networks. We believe that regularized back-propagation networks would be quite conducive to more widespread use of neural networks in production settings, where there are usually no sufficient human resources to handle the extensive network parameter selection and experimentation required by traditional back-propagation networks.

Redundant Data Interpolation

Smoothness in the data space may also be achieved by redundant data interpolation. This is done by interpolating between the training data points with a smooth function, performing the training on the expanded data set, and then discarding the interpolated data after training. This sequence produces a smoother solution even for larger networks and helps stabilize network training. The reason for this is that interpolating the data increases the data complexity, and therefore reduces the ratio of the complexity of the network relative to that of the data. Figure 7a shows the result when the same traditional 50-node network used before was utilized, albeit this time the 10-point training data set was locally interpolated by cubic splines to 50 points prior to training, and the network was trained on the interpolated data. Again, the network output is much smoother than in the unregularized case, and it looks comparable to that of the traditional network of the optimal size (Figure 3a) or the regularized back-propagation network (Figure 6a). Overfitting has thus been alleviated even though the network is much larger than needed.

The advantage of the redundant data interpolation approach over regularized back-propagation networks is that this method would never underfit the data (assuming the network is large enough) since the expanded training data set is derived wholly from the original data, whereas, as we mentioned above, an improper manual selection of a fixed regularization parameter can produce a regularized back-propagation network that is too smooth to adequately fit the data. (This is averted, however, by the automatic selection of the regularization parameter via the Bayesian interpolation method we mentioned earlier.) The disadvantage of redundant interpolation is that it is difficult to determine beforehand the amount of interpolation required for a given problem, which means that this approach does not completely solve the parameter tweaking issue. Moreover, each fold of interpolation doubles the time required to train the network, and so this method can be quite demanding in its compute and storage requirements, and is not suitable for large data sets.

Also note that the redundant data interpolation approach is less resistant to noise in the data than the regularized back-propagation method, since interpolation is carried here over the noisy data. Thus, spurious characteristics, induced by noise in the data, may be introduced into the solution, as the training data set is honored here more faithfully. Figure 7b shows the result when redundant data interpolation was utilized to obtain the 50-node network for the data that have been encumbered by 10% noise. Although the final outcome is smooth, the accuracy is not as good as that of the network of the optimal size (Figure 3b) or the regularized back-propagation network (Figure 6b). The data trend characteristics were not recreated as closely, since the original noisy data set is fitted more faithfully here.

Validating Data Set

As training progresses for a traditional back-propagation network, the misfit error is increasingly driven to ever smaller values. While this indicates that longer training times cause the network to fit the data well, it is not a desirable effect. As the misfit error gets progressively smaller, the network starts to memorize the training data set and gradually loses it ability to generalize well. In other words, the network will overfit the data and will not represent a good predictive model. While in the previous two approaches overfitting was alleviated by constraining the network to produce smoother output, overfitting may also be lessened by stopping the training phase early to prevent the network from losing its generalization behavior. The result is a network that fits the data less perfectly but is more useful as a predictive model because it can generalize well to data it has not encountered. Huang *et al.* (1996) employed this method to predict permeability from well logs.

In the validating data set approach, the data available for training are split into a training data set and a validating data set. The validating data set is not used directly in the training. Instead, the network error on this set is merely monitored at each stage during training. Initially, the error for both the training and validating sets will decrease, but as training progresses, the error on the validating data set will eventually start to increase (while it continues to decrease on the training data set). This indicates that the network has started to memorize the training data and is losing its generalization ability. At this point, training is stopped and the weights obtained at the minimum validation error are used to build the network.

Figure 8a shows the result when the same traditional 50-node network as above was utilized, except this time the data set was split into a training set and a validating set, and training was stopped once the validation error started to increase. The training and validation data points are also indicated in the figure. Unfortunately, because half the original data available for training were used for validation rather than training, this dramatically decreased the size of the training data set, and the network accuracy was therefore quite poor. The test was rerun, but this time the entire original training data set was employed for training the network, and an additional, distinct subset of the data was used for validation. The result, along with the training and validation points, is shown in Figure 8b. This result is an improvement over the result of Figure 8a (the RMS error decreased from 32% to 15%), and over that traditional network (Figure 4a). However, the accuracy obtained here is nowhere near that of the traditional network of the optimal size (Figure 3a) or the regularized back-propagation network (Figure 6a).

Although the validating data set method is workable, we believe it to be much less effective and less accurate than the regularized back-propagation method. Its chief disadvantage is that there is a reduced amount of data on which to train the network, since part of the training data are used for validation and not utilized directly in training the network. This is problematic when the data available for training are limited, which is not an uncommon scenario. In addition, the final solution is sensitive to the rather arbitrary split of the data into training and validation subsets, which makes the network solution less consistent and adversely affects its reproducibility. On the other hand, the training time required by this method is smaller than those of the first two approaches mentioned above, so this can be a favorable method when a massive training data set is to be employed.

Of the three approaches we presented here, we believe the regularized back-propagation method to be the most robust, accurate, and consistent. Its compute time is moderate and it does not waste valuable training data or require redundant interpolation. We will thus focus on this method in subsequent discussion.

RADIAL BASIS NETWORKS

Radial basis networks are in general similar in structure to back-propagation networks, but with some important distinctions that cause them to operate in an entirely different manner. A typical radial basis network has one internal layer and an output layer. The output layer is usually linear. The internal layer, on the other hand, is always a radial basis layer. Such a layer differs from those in back-propagation networks in three aspects. First, the transfer function is the radial basis function, given by:

$$f_{rb}(x) = e^{-x^2}. \tag{11}$$

An example plot of this function is shown in Figure 9a. Second, instead of computing the inner product between the input vector and the layer weights, the Euclidian distance of the two is calculated. Third, this distance is divided by the offset rather than added. The responses of all neurons of the radial basis layer are then aggregated in the linear layer. The output of a typical two-layer radial basis network is thus given by:

$$\begin{aligned} y_{1,i} &= f_{rb}(\|\mathbf{w}_{1,i} - \mathbf{x}\|/b_{1,i}) \quad l : 1 \to N_1, & (12) \\ y_{2,i} &= \mathbf{w}_{2,i} \cdot \mathbf{y}_{1,i} + b_{2,i} \quad i : 1 \to N_2, & (13) \end{aligned}$$

where the symbols are defined as in equations (5) and (6).

From Figure 9a we see that the radial basis function acts as a discriminator that produces a high value whenever the input vector is similar to the neuron (i.e., when the Euclidian distance in equation (12) between the input vector and the neuron is small). The purpose of the offset is to modify the neuron's region of sensitivity to the input vector. A larger offset causes the neuron to be sensitive to a larger region of the input space, whereas a smaller offset restricts the neuron to respond only to the narrow band of the input space that is most similar to that neuron. The different plots in Figure 9a correspond to radial basis functions in which various offsets are utilized, according to the equation:

$$f_{rb}(x) = e^{-x^2/b}. \tag{14}$$

Thus, a radial basis network can be thought of as performing interpolation in the model space (in contrast to the more familiar interpolation in the data space). It is in this sense that radial basis networks achieve natural smoothness, and this intrinsic smoothness is controlled by varying the offset in the network's radial basis layer. As we mentioned, a large offset causes the neurons to respond to large areas of the input space, thus increasing smoothness by increased overlapping of the neural responses at the expense of reduced fit to the data. On the other hand, a small offset would severely restrict any overlap in the neural responses and would thus produce a network with poor generalization. The offset is an important parameter for radial basis networks; the size of the network is not as critical in affecting the network accuracy. Figure 9b shows a similar example to that of Figure 3a, except here a radial basis network is used. With a reasonable choice of the offset, the network produced an excellent predictive model that recreated the entire data set very well (Figure 9b). With too large an offset, though, the network produced an output that is too smooth and thus severely underfit the data (Figure 10a, where the offset used was 10) or was completely insensitive to the data (Figure 10b, where the offset was 100). On the other hand, when the offset is too small, the network generalization was poor such that it would give meaningful predictions only in narrow zones around the training data points (Figure 11a, where the offset was 0.2), and that behavior becomes worse as the offset is reduced further (Figure 11b, where the offset was 0.01).

The advantage of radial basis networks is that they are simple to build. In fact, designing an exact radial basis network with zero misfit error can be done in a fraction of the time needed to train an equivalent back-propagation network (Chen *et al.*, 1991). There are drawbacks, however. Radial basis networks are often much larger than their back-propagation counterparts, especially if numerous training vectors are utilized. In fact, the exact method produces a radial basis network with as many nodes in the radial basis layer as there are input vectors. Although alternate methods exist that produce smaller radial basis networks than the ones obtained by the exact method, those smaller networks are still usually larger than their back-propagation counterparts, since the sigmoidal neurons of the back-propagation networks respond to much larger areas of the input space than the radial basis neurons. Therefore, more radial basis neurons are usually needed for a good fit to the input. Because of their size, radial basis networks are often slower to apply at the forward modeling phase than back-propagation networks.

Although radial basis networks have inherent smoothness by design, such smoothness depends on the offset parameter, which has to be selected manually beforehand. This selection is subjective (not methodological) and the choice affects the results greatly. As we mentioned earlier, regularized back-propagation networks can be designed with automatic selection of the regularization parameter. Therefore, as far as smoothness is concerned, radial basis networks may be superior to traditional back-propagation networks, but not as versatile as regularized back-propagation networks. Still, both regularized back-propagation and radial basis networks are useful, and they tend to produce rather comparable results in general. We investigate the application of both types of networks in subsequent sections, and comment on the cases where the application of either may be more optimal.

INPUT/OUTPUT ARCHITECTURE

We discuss two distinct geometries for setting up the network with regard to its input and output vectors. During training, in both architectures, a seismic trace near a well is presented to the network input while the observed reservoir property in the well (we focus on porosity) is presented at the output. After training, seismic traces in the inter-well regions are fed to the network to estimate the porosity there. Also, in both architectures, either a back-propagation or a radial basis network may be used. In other words, these architectures describe the input/output geometry and are independent of the network type and structure.

Columnar Architecture

Columnar architecture represents the conventional geometry that is most often used with neural networks. In this setup, each seismic trace is represented by an input vector, and hence the network has as many input nodes as there are time samples in the seismic trace. At the network output, each node can represent either an instantaneous time sample of the reservoir porosity or an interval average of that porosity. The number of nodes in the input and output layers are decoupled and there is no requirement for these two numbers to be the same. A schematic of the architecture is shown in Figure 12a.

Columnar geometry allows the most flexibility in terms of setting up the problem, since any number of output samples can be ascribed, independent of the number of input samples. The downside, though, is that the size of the back-propagation network required here is significantly increased by any rise in the number of time samples of the seismic trace, since the number of nodes in the input layer is directly tied to the number of samples in the input. For a typical back-propagation network that has 50 nodes in its internal layer, each extra time sample adds 50 extra neural weights to be determined (and one extra offset). On the other hand, the number of input vectors presented to the network is equal to the number of traces in the training data set, which is usually small. This results in a radial basis network of manageable size for this geometry. Therefore, for columnar architecture, radial basis networks are preferred, since the required network is comparatively faster to build than the equivalent back-propagation network in this case. In general, back-propagation networks are more suitable to problems with numerous but small input vectors, whereas radial basis networks are more effective in cases with few but large input vectors.

Serialized Architecture

In serialized architecture, each sample in the seismic trace is fed to the network in sequence, and the output represents the instantaneous porosity at that time sample. The entire seismic samples for all traces are thus streamed through the network one by one in continuum. This simple geometry is shown in Figure 12b. It is too optimistic, however, to expect such an unsophisticated network to be powerful enough to capture the intricacies of anything but the simplest of cases, and indeed such a network performs rather poorly in our data tests. The reason behind this is that, compared to the columnar architecture, this simple serialized architecture not only has very little vertical spatial coverage of the data, but also there is no delineation of the data belonging to one trace from the next, thus undermining the correlation characteristics of the input, which we believe is important.

The above shortcomings can be readily remedied, however, by: (1) extending the vertical spatial coverage of each input vector by including the sample just before the current input sample and the sample just after it, and (2) adding depth information, which is simply the sample time index in this case. Hence, instead of using just one input node, four nodes are utilized. The first receives the seismic sample at t_i, the second receives the seismic sample at t_{i-1}, the third accepts the seismic sample at t_{i+1}, and the fourth receives the index i itself. The input is fed sequentially for each time sample t_i in the seismic trace (when t_i is the first sample or last sample in the trace, zero is used for t_{i-1} and t_{i+1}, respectively). A schematic of serialized architecture is shown in Figure 12c. Of the two remedies above, the first adds vertical spatial information and enhances the spatial coverage of the input, while the second delineates each well data set (sharp discontinuities at the boundaries between the well data sets) and moderates the input by adding a low frequency integrator component that enhances the continuity and smoothness of the output.

In contrast to the simple one-input-node architecture of Figure 12b, this improved serialized configuration performs as well as the columnar configuration. Even more spatial correlation can be added to enhance the effectiveness of this architecture by

Columnar Architecture	Serialized Architecture
Number of input samples need not equal number of output samples.	Number of input samples must equal number of output samples.
Size of input layer equals size of input trace.	Size of input layer is independent of size of input trace.
Produces few input vectors.	Produces numerous input vectors.
Produces large input vectors.	Produces small input vectors.
Requires large back-propagation networks.	Requires small back-propagation networks.
Requires small radial basis networks.	Requires large radial basis networks.

Table 1: Summary of the properties of the columnar and serialized network input/output architectures.

adding more points at the sides of the current input sample. As a matter of fact, as the number of such points increases, the serialized architecture approaches the columnar one in the limit. However, the current geometry (Figure 12c), with just one extra sample at each side and the depth index, performed satisfactorily for our purpose. Furthermore, as the number of additional points is increased, edge inconsistencies due to the zeros at the beginning and end of the seismic trace start to become significant.

The advantage of serialized architecture is that it requires rather small back-propagation networks to handle even large seismic traces. In fact, in this case, the size of the back-propagation network is totally independent of the size of the seismic trace or the number traces in the training data set. This makes this configuration quite suitable to back-propagation networks. On the other hand, since the number of input vectors is significantly increased by serializing them, serialized architecture requires very large radial basis networks, which makes utilizing such networks for this configuration inefficient. Additionally, the number of time samples for both the input (seismic trace) and output (porosity) has to be the same. This limitation may be alleviated, however, by subsequent averaging of the output if interval averages of porosity are desired. Table 1 summarizes the properties of the columnar and serialized architectures. In general, though, with equivalent networks used in the two architectures, the serialized configuration tends to be more efficient and less sensitive to the network parameters (such as the selection of the radial basis network offset). The latter is perhaps due to the enhanced smoothness provided by the low frequency depth index integrator.

RELATION TO MODEL-BASED INVERSION AND GLOBAL METHODS

As we stated previously, neural networks represent a more suitable approach to nonlinear problems than linear inversion since they are inherently nonlinear. They are also more effective than model-based nonlinear inversion in complex problems where the relation between input and output cannot be described by *a priori* analytical or numerical models. However, where the physical model is reasonably accurately known, model-based inversion yields better performance. Also, knowledge of the exact underlying model makes the solution more resistive to noise and helps avoid fitting that noise, since the predictive model is constrained to have the shape of the exact underlying model.

For the synthetic data constructed from equation (7), the neural network that produced the results of Figure 6a required about 1 Mflops (million floating point operations), or roughly 0.6 seconds to build. To compare, the system was also solved by model-based nonlinear inversion utilizing equation (7) as the operator (i.e., using the exact model) via the Levenberg-Marquardt algorithm (the algorithm we use throughout this work). In that case, it took the system 0.1 Mflops (about 0.1 seconds) to converge. Hence, neural networks take more time to arrive at the solution, but require no knowledge of the underlying operator. These comparisons are only approximate, of course, as several other factors affect the computations (for example, the starting model for the nonlinear model-based inversion). In addition, because the times quoted here represent real time rather than CPU time, the relation between the number of floating-point operations and run times is not always linear.

Nonlinear model-based inversion can also be used to solve directly for the network weights. In other words, the network output can be regarded as a black-box numerical function to be approximated by using the network itself as the model and attempting to compute its weights. This represents a direct solution to the network where the network layers are treated simultaneously rather than separately, and it is therefore a less efficient approach than using the back-propagation rule. However, it may be necessary to utilize the direct solution method when application of the back-propagation technique becomes difficult, such as when employing complex regularization or constraining the solution by some rules. As an example, the solution to the network used in the tests mentioned at the beginning of this section was computed directly by nonlinear inversion. The network solution was similar to that obtained by the back-propagation rule, but this time it took the system 2 Mflops (about 1.2 seconds) to converge, about twice as much compute time as the back-propagation method.

The solution space for the network weights can be quite complex and may have numerous flat valleys and several local minima (Hagan *et al.,* 1995). Figure 13 shows the error surface for a simple one-node network with respect to the two network coefficients (weight and offset), and it does indeed look complicated even for this simple network. Global search methods (such as simulated annealing and genetic algorithms) can be used to obtain the network weights that represent the global network solution. This represents an improvement over the local optimization solutions discussed above (including back-propagation) at the expense of greater run time. As an example, the solution for the network used in the previous tests was obtained using genetic algorithms (seven binary digits were used, the zero position was taken to be three, the crossover and mutation probabilities were fixed at 0.7, and an initial population of 500 was employed). The method converged to a solution that is similar to the ones obtained previously (it seems this problem is simple enough that the global solution was arrived at previously

Method	Mflops	Time (seconds)
Exact Model	0.1	0.1
Back-Propagation	1	0.6
Direct Solution (Levenberg-Marquardt)	2	1.2
Direct Solution (Genetic Algorithms)	50	60

Table 2: Summary of the compute operations and times needed to obtain the inversion model parameters and the neural network weights using various techniques. The direct solution methods treat the network as a black-box and solve for its weights directly, thus bypassing the back-propagation rule. Mflops = million floating point operations.

even with local optimization algorithms), but the compute time was enormous: about 50 Mflops (over one minute). This is at least an order of magnitude more compute time than before. However, this may be a price worth paying if local minima become a problem. Table 2 summarizes the compute times for the different approaches discussed here.

SUMMARY AND CONCLUSIONS

Traditional joint inversion methods have the drawback of requiring an *a priori* prescribed operator that links the reservoir properties to the observed seismic response. Such operators are difficult to characterize, they vary regionally, and often hold only under ideal conditions. A further disadvantage is that these methods rely on a linearized approach to the solution and are thus heavily dependent on the selection of the starting model. Neural networks provide a useful alternative that is inherently nonlinear and completely data-driven, requiring no initial model and no explicit *a priori* operator linking the measured rock properties to the seismic response. However, the performance of traditional neural networks in production settings has been inconsistent due to the extensive parameter tweaking needed to achieve satisfactory results. The reason the results of traditional networks are sensitive to the network parameters (such as the size and training length) is that these networks have a nonmonotonous generalization behavior and tend to overfit the data if the network complexity is high relative to that of the data.

In this paper, we presented an approach to estimate reservoir rock properties (such as porosity) from seismic data through the use of regularized back propagation and radial basis networks. These networks have inherent smoothness characteristics that alleviate the nonmonotonous generalization problem associated with traditional networks and help to avoid overfitting the data. Our approach has several advantages over the traditional joint inversion and neural network methods, as it is inherently nonlinear, requires no *a priori* operator or initial model, and is not prone to the overfitting problems that hinder the effectiveness of traditional networks. Therefore it requires no extensive parameter experimentation.

Traditional back-propagation networks are regularized by adjusting their objective function such that not only is the training misfit error reduced, but the network weights are reduced as well. This gives rise to smaller variations in the network output and thus yields smoother results that are less likely to overfit the data. Other approaches that aim to alleviate the network's tendency to overfit the data include redundant data interpolation (where the training data is interpolated by a smooth function prior to training) and early stopping of the network training by monitoring the error on a validating data set. However, we have demonstrated that both of these approaches are less accurate and less flexible than the regularization method. The former is compute-intensive and less resistive to noise in the data, since the interpolation is carried on the noisy data and the noise is fitted faithfully, while the latter results in a reduced training data set, which can be detrimental to network accuracy, and is also sensitive to the arbitrary split of the data available for training into training and validation subsets. The regularization method, on the other hand, does not waste valuable training data or require redundant interpolation, and its regularization parameter can be estimated automatically within the technique.

Radial basis networks can be thought of as performing interpolation in the model space by overlapping their neural responses to the input, and it is in this sense that they achieve natural smoothness. Although these networks are simpler to build than the corresponding back-propagation networks, large radial basis networks are required when the training data set contains numerous input vectors. Also, the network offset, which controls the amount of smoothness exhibited by the network, has to be prescribed manually and the choice affects the network output significantly. In terms of smoothness, then, radial basis networks may be superior to traditional back-propagation networks, but not as versatile as regularized back-propagation networks.

We presented two architectures for setting up the input/output network geometry: the columnar and serialized architectures. The former allows the most flexibility, since the number of output samples of the network is independent of that of the input, but requires the size of the input layer to be as large as the size of the input vector and thus yields very large back-propagation networks. The latter requires the number of samples of the network input and output to be the same, but the size of the resulting back-propagation network is independent of the size of the input vector or the number of vectors in the training data set, and thus produces much smaller back-propagation networks. Serialized architecture proved to be as effective as the columnar configuration when extra samples are used besides the current input sample (which enhances the vertical spatial coverage of the input) and the time index is added to the input vector (which delineates each well data set and moderates the input by adding a low frequency integrator component). Since back-propagation networks favor small but numerous input vectors, while radial basis networks favor few but large input vectors, serialized architecture is more appropriate for back-propagation networks, while columnar architecture is more efficient for radial basis networks.

Finally, traditional nonlinear model-based inversion techniques (including global search methods such as simulated annealing and genetic algorithms) can be used to solve directly for network weights. Although this is much less efficient than the back-propagation rule, the direct solution is an alternative that can be pursued when application of the back-propagation rule is difficult (such as when applying complex regularization or constraining the solution), or when local minima in the error surface of the network become a problem that requires global optimization in order to avoid it.

ACKNOWLEDGMENTS

We would like to thank Saudi Aramco for supporting this research and for granting us permission for its publication. This work was also supported by the Borehole Acoustics and Logging/Reservoir Delineation Consortia at the Massachusetts Institute of Technology.

REFERENCES

Anderson, J.K., 1996, Limitations of seismic inversion for porosity and pore fluid: Lessons from chalk reservoir characterization exploration, *Expanded Abstracts, 66th Ann. Internat. Mtg., Soc. Expl. Geophys.,* 309–312.

Angeleri, G.P. and Carpi, R., 1982, Porosity prediction from seismic data, *Geophys. Prosp., 30,* 580–607.

Baldwin, J.L., R.M. Bateman, and C.L. Wheatley, 1990, Application of a neural network to the problem of mineral identification from well logs, *The Log Analyst, 3,* 279–293.

Boadu, F.K., 1998, Inversion of fracture density from field seismic velocities using artificial neural networks, *Geophysics, 63,* 534–545.

Brac, J., Dequirez, P.Y., Jacques, C., Lailly, P., Richard, V., and Nhieu, D., 1998, Inversion with a priori information: A step toward consistent and integrated interpretations of reservoirs, *AAPG Bull., 72,* 991.

Calderon-Macias, C., Sen, M.K., and Stoffa, P.L., 1998, Automatic NMO correction and velocity estimation by a feedforward neural network, *Geophysics, 63,* 1696–1707.

Chen S., Cowan, C.F., and Grant, P.M., 1991, Orthogonal least squares learning algorithm for radial basis networks, *IEEE Transactions on Neural Networks, 2,* 302–309.

Constable, S.C., Parker, R.L. and Constable, C.G., 1987, Occam's inversion—A practical algorithm for generating smooth models from electromagnetic sounding data, *Geophysics, 52,* 289–300.

de Buyl, M., Guidish T., and Bell, F., 1986, Reservoir description from seismic lithologic parameter-estimation, *J. Petrol. Tech., 40,* 475–482.

Foresee, F.D. and Hagan, M. T., 1997, Gauss-Newton approximation to Bayesian regularization, *Proc. 1997 Internat. Joint Conf. on Neural Networks,* 1930–1935.

Hagan, M.T., Demuth, H.B., and Beale, M., 1995, *Neural Network Design,* PWS Pub. Co.

Hopfield, J.J., 1982, Neural networks and physical systems with emergent collective computational abilities, *Proceed. National Acad. Sciences, 79,* 2554–2558.

Hornik, K., Stinchcombe, M. and White, H., 1989, Multilayer feedforward networks are universal approximators, *Neural Networks, 2,* 359–366.

Hrycej, T., 1992, *Modular Learning in Neural Networks: A Modularized Approach to Neural Network Classification,* John Wiley & Sons, NY.

Huang, Z., Shimeld, J., Williamson, M., and Katsube, J., 1996, Permeability prediction with artificial neural network modeling in the Venture gas field, offshore eastern Canada, *Geophysics, 61,* 422–436.

Liu, Z. and Liu, J., 1998, Seismic-controlled nonlinear extrapolation of well parameters using neural networks, *Geophysics, 63,* 2035–2041.

MacKay, D.J., 1992, Bayesian interpolation, *Neural Computation, 4,* 415–447.

McCormack, M.D., Zaucha, D.E., and Dushek, D.W., 1993, First-break refraction event picking and seismic data trace editing using neural networks, *Geophysics, 58,* 67–78.

McCulloch, W.S. and Pitts, W.H., 1943, A logical calculus of ideas imminent in nervous activity, *Bull. Math. Biophysics, 5,* 115–133.

Minsky, M. and Papert, S., 1969, *Perceptrons*, MIT Press, Cambridge, MA.

Poulton, M.M., Sternberg, B.K., and Glass, C.E., 1992, Location of subsurface targets in geophysical data using neural networks, *Geophysics, 57,* 1534–1544.

Rogers, S.J., Fang, J.H., Karr, C.L., and Stanley, D.A., 1992, Determination of lithology from well logs using a neural network, *AAPG Bulletin, 76,* 731–739.

Rosenblatt, F., 1958, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Rev., 65,* 386–408.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J., 1986, Learning representations by back-propagating errors, *Nature, 323,* 533–536.

Sambridge, M.S., 1990, Nonlinear arrival time inversion—constraining velocity anomalies by seeking smooth models in 3-D, *Geophys. J. Internat., 102*, 653–677.

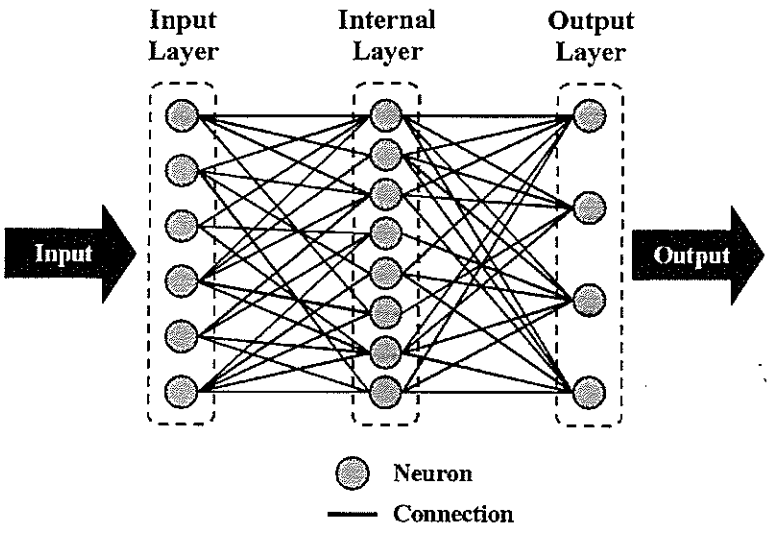
Saggaf, M.M. and E.L. Nebrija, 2000, Estimation of lithologies and depositional facies from wireline logs, *AAPG Bulletin, 84,* in publication.

Saggaf, M.M., Toksöz, M.N., and Mustafa, H.M., 2000, Application of smooth neural networks for inter-well estimation of porosity from seismic data, this report, 3-1–3-26.

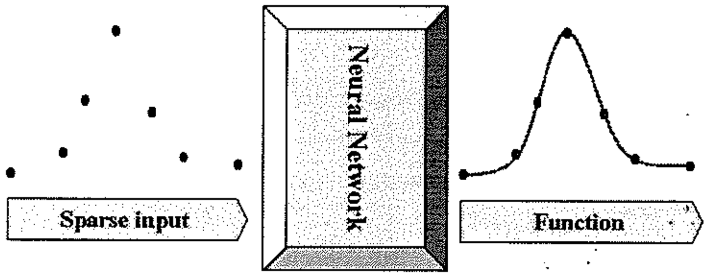
Tikhonov, A.N. and Arsenin, V.Y., 1977, *Solutions of Ill-Posed Problems,* V.H. Winston and Sons.

Wang, L.-X. and Mendel, J.M., 1992, Adaptive minimum prediction-error deconvolution and source wavelet estimation, using Hopfield neural networks, *Geophysics, 57,* 670–679.

(a)



(b)



(c)

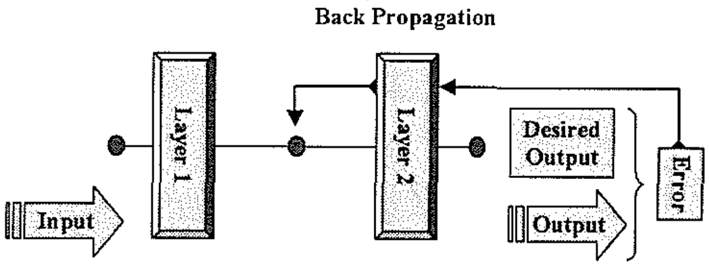
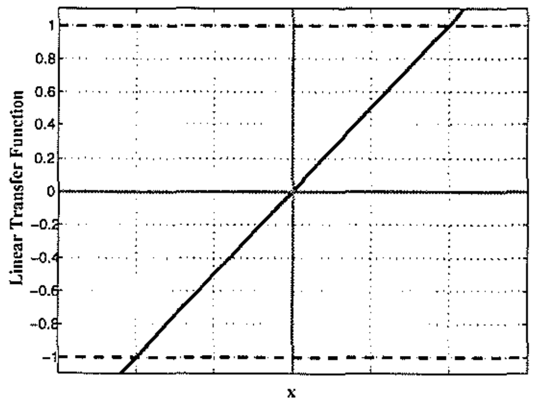
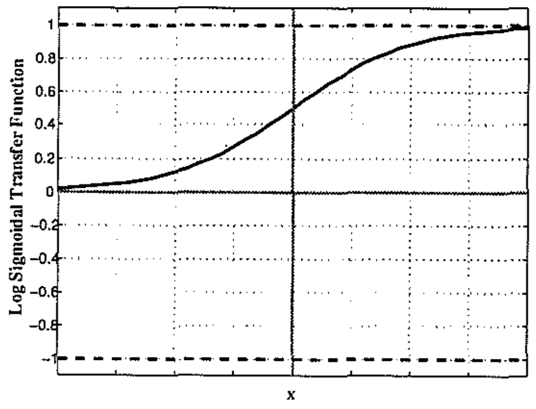


Figure 1: Neural networks: (a) typical structure of the network, data are passed from input to output, going through the nodes (neurons) in each layer, and is modified along the way; (b) the network acts as a function approximator and constructs a predictive model for an unknown function from a sparse set of data points presented to the network during training; and (c) in the back-propagation rule, the error is successively propagated back through the outermost layers, and the solution is thus computed for each internal layer.

(a)



(b)



(c)

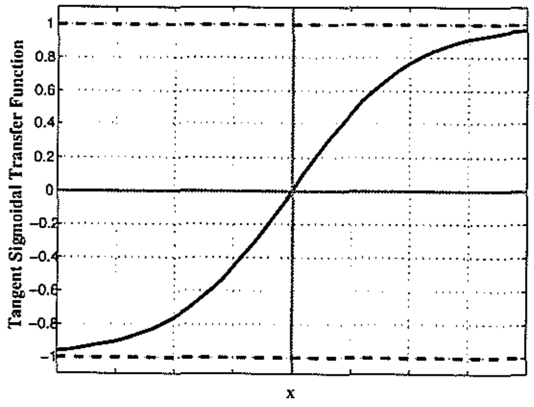
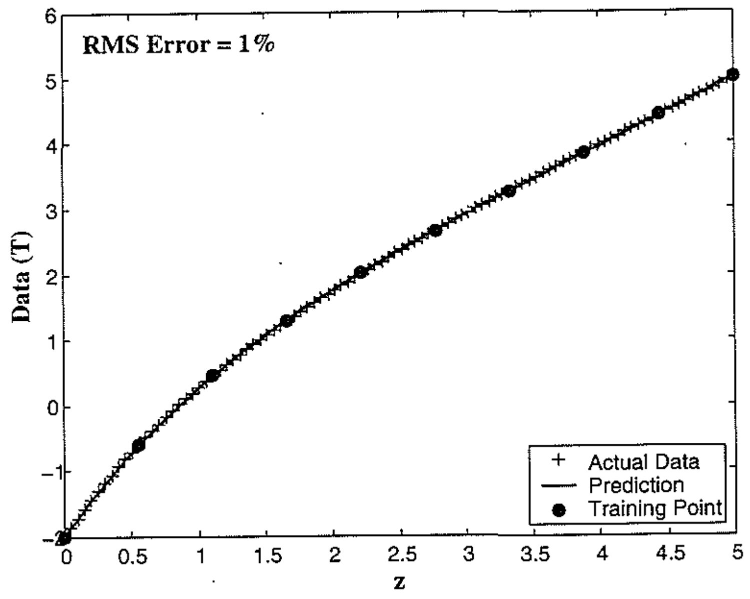


Figure 2: Typical plots for the (a) linear, (b) log sigmoidal, and (c) hyperbolic tangent sigmoidal transfer (activation) functions.

(a)



(b)

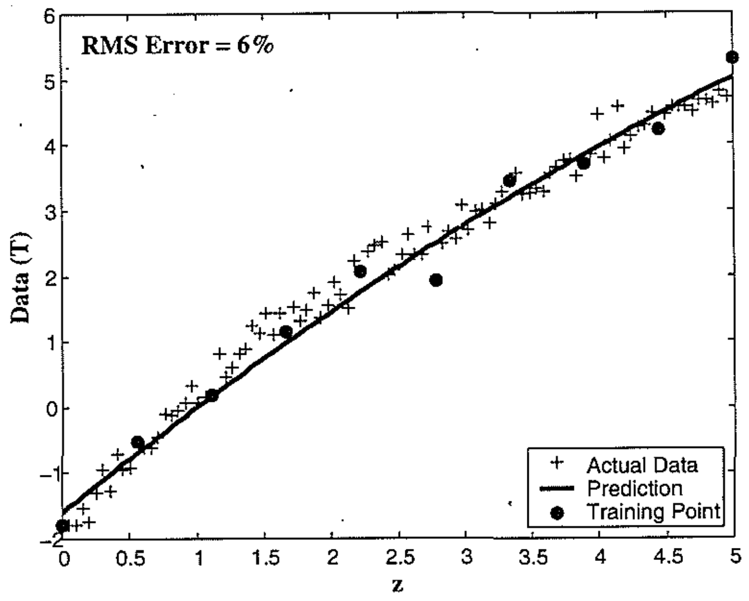
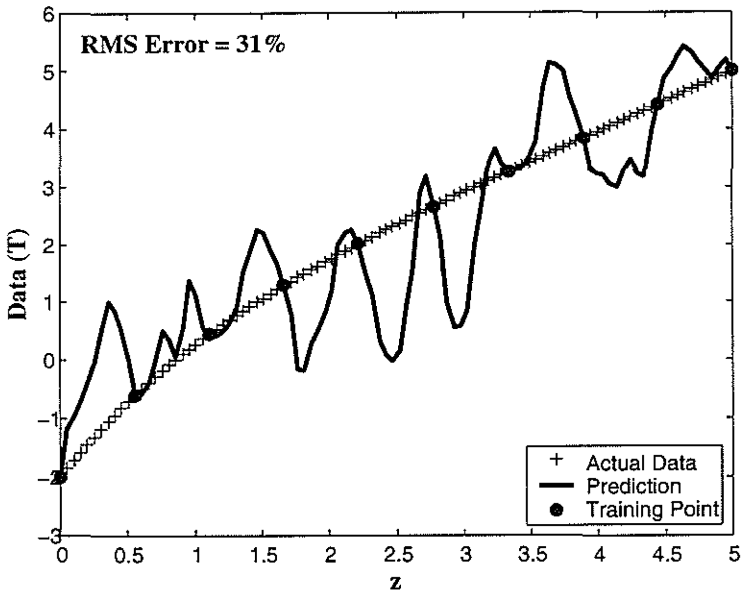


Figure 3: For a traditional back-propagation network that has two nodes in its internal layer, a comparison of the original data series and the network prediction when the data are: (a) noise-free, and (b) encumbered by 10% noise relative to the mean of the data. The training data points are indicated on the plots.

(a)



(b)

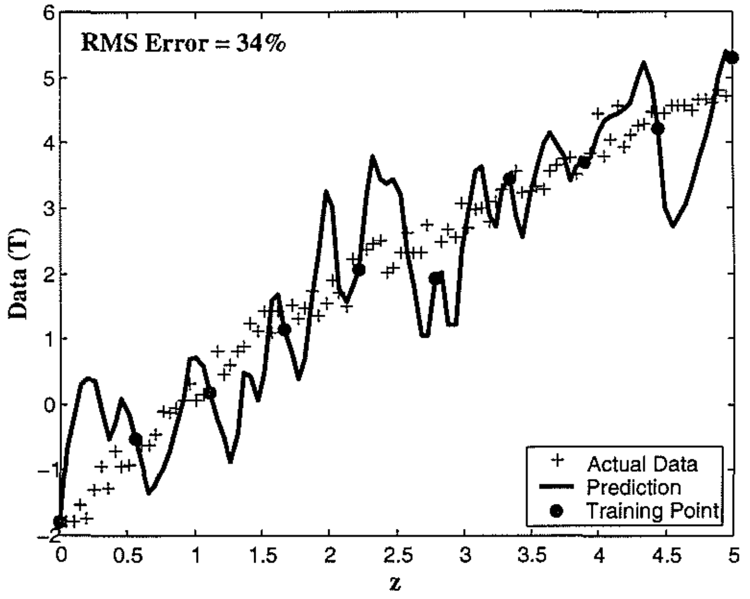
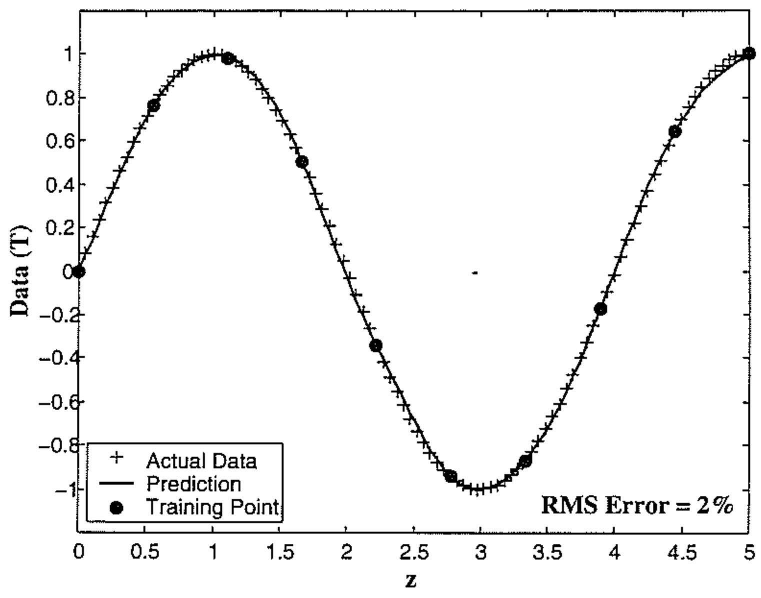


Figure 4: For a traditional back-propagation network that has 50 nodes in its internal layer, a comparison of the original data series and the network prediction when the data are: (a) noise-free, and (b) encumbered by 10% noise relative to the mean of the data. The training data points are indicated on the plots. Overfitting is quite evident here: the training data points are fit very well, but the network fails to generalize well to other points in the data.

(a)



(b)

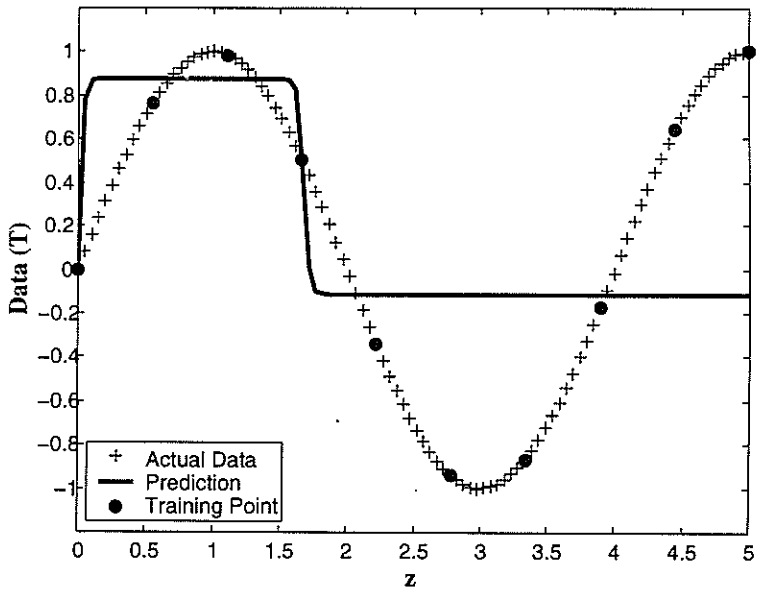
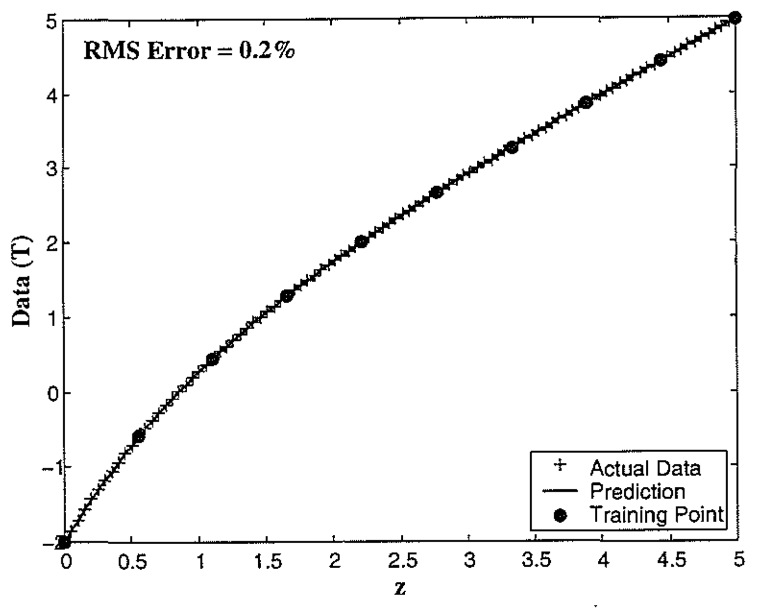


Figure 5: Attempt to estimate a sine function from a sparse training subset of the data: (a) network with five nodes in its internal layer gives rise to an adequate fit to the data since the network is powerful enough to model the data, while (b) smaller network that has two nodes in its internal layer results in underfitting, as the network is not complex enough to capture the intricacies of the data.

(a)



(b)

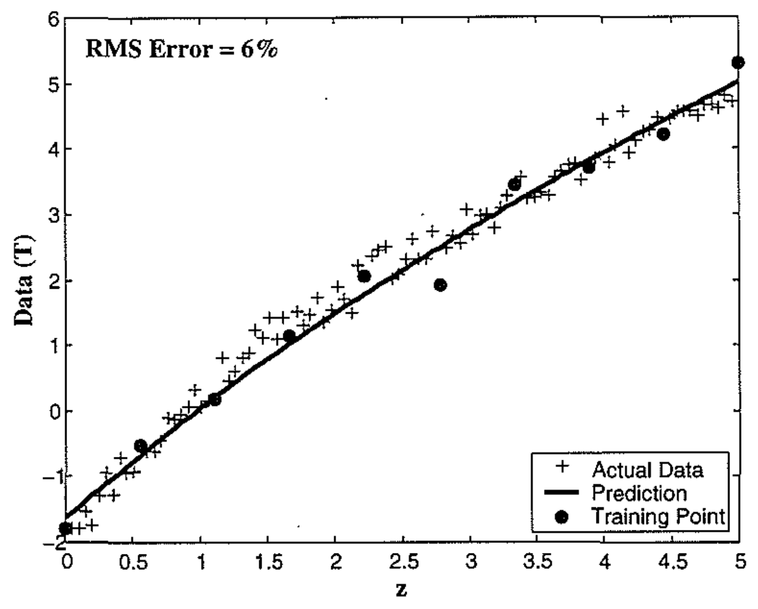
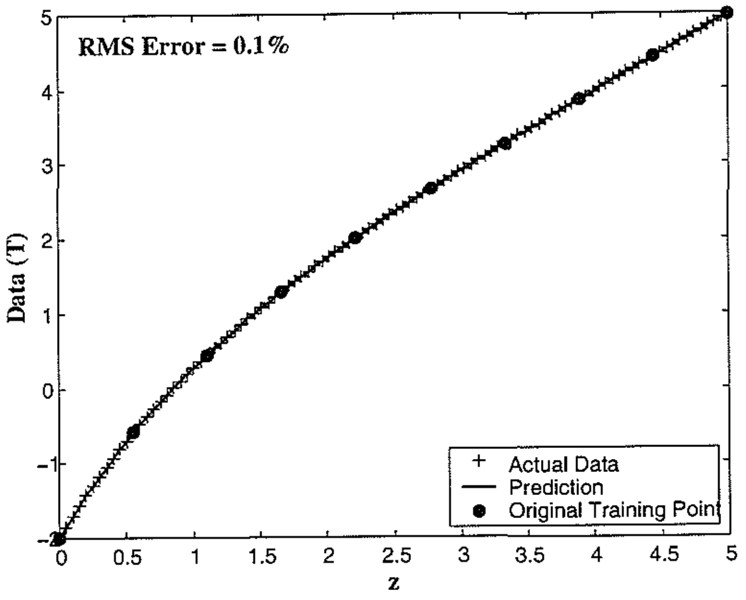


Figure 6: For a regularized back-propagation network that has 50 nodes in its internal layer, a comparison of the original data series and the network prediction when the data are: (a) noise-free, and (b) encumbered by 10% noise relative to the mean of the data. The training data points are indicated on the plots. Overfitting was averted even though the network complexity is large compared to that of the data.

(a)



(b)

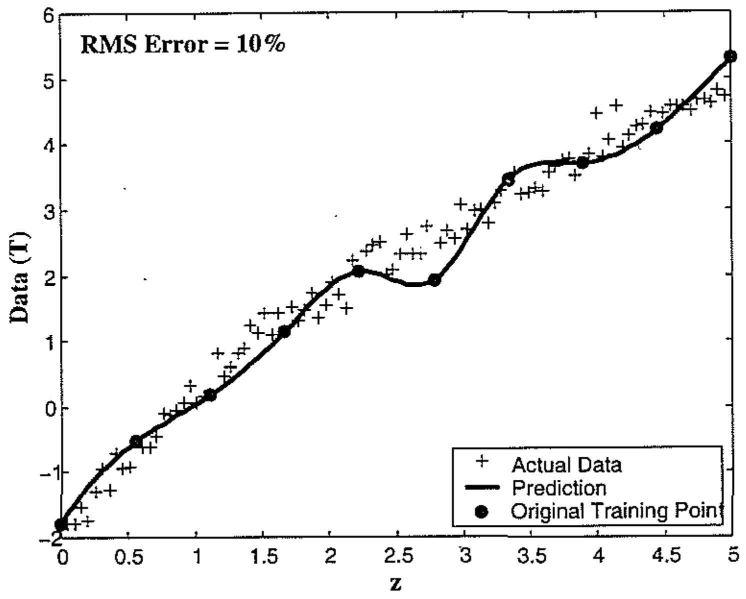
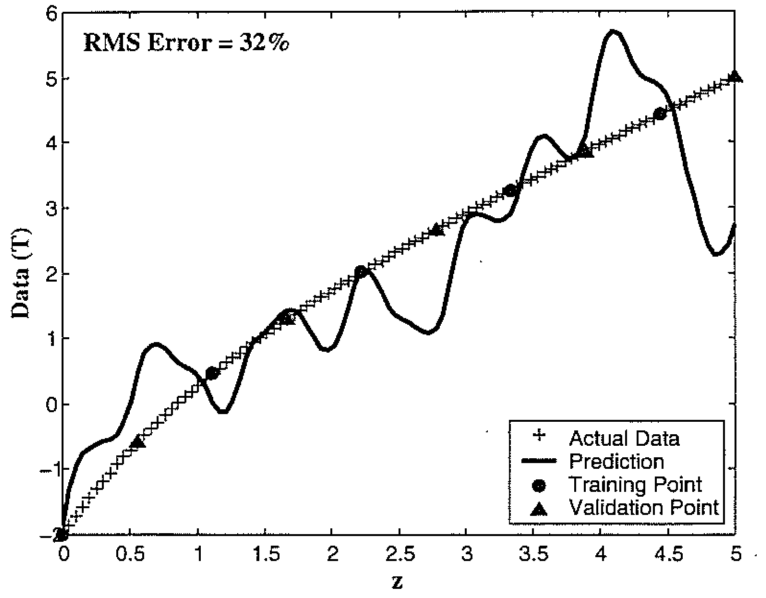


Figure 7: For a traditional back-propagation network that has 50 nodes in its internal layer and whose training data have been interpolated by cubic splines prior to training, a comparison of the original data series and the network prediction when the data are: (a) noise-free, and (b) encumbered by 10% noise relative to the mean of the data. The training data points are indicated on the plots. Note how the network in (b) loses some accuracy since it fits the noise interpolated in the data.

(a)



(b)

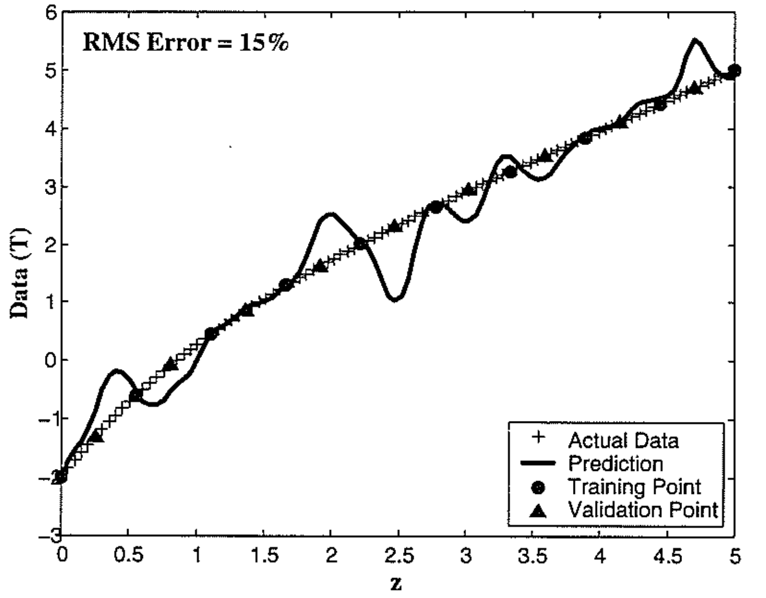
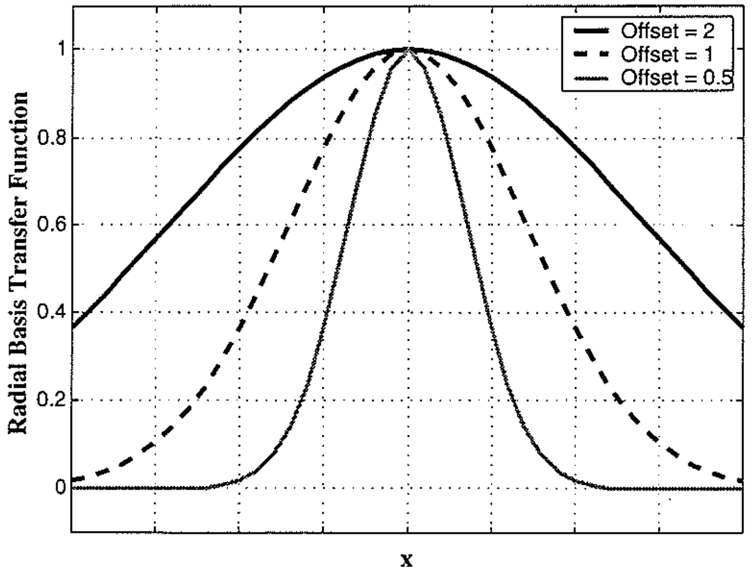


Figure 8: For a traditional back-propagation network that has 50 nodes in its internal layer and whose training was stopped once the error on a validating data set started to increase, a comparison of the original data series and the network prediction when the data are noise-free: (a) the original training data set was split into a training set and a validation set, and (b) the entire original training data set was used for training and an additional set was used for validation. Note how the network purposely does not attempt to fit the training data perfectly to avoid memorizing the training data set and losing its ability to generalize well.

(a)



(b)

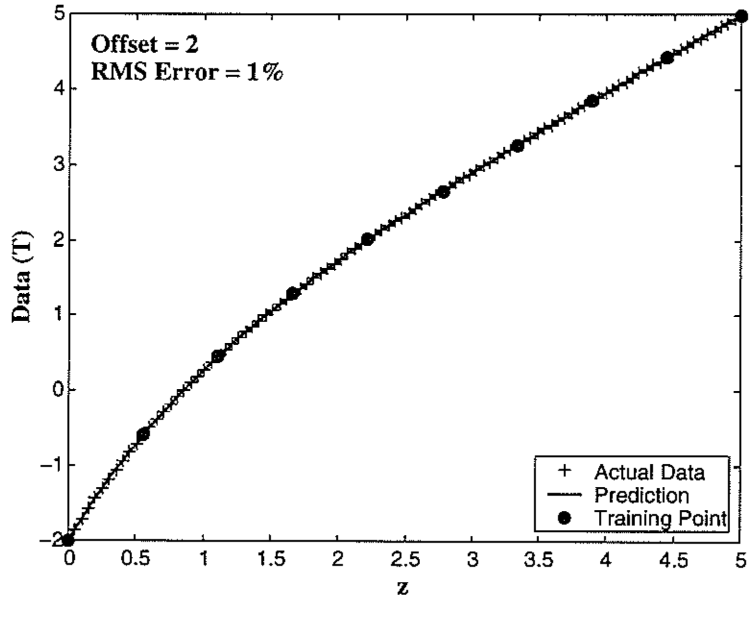
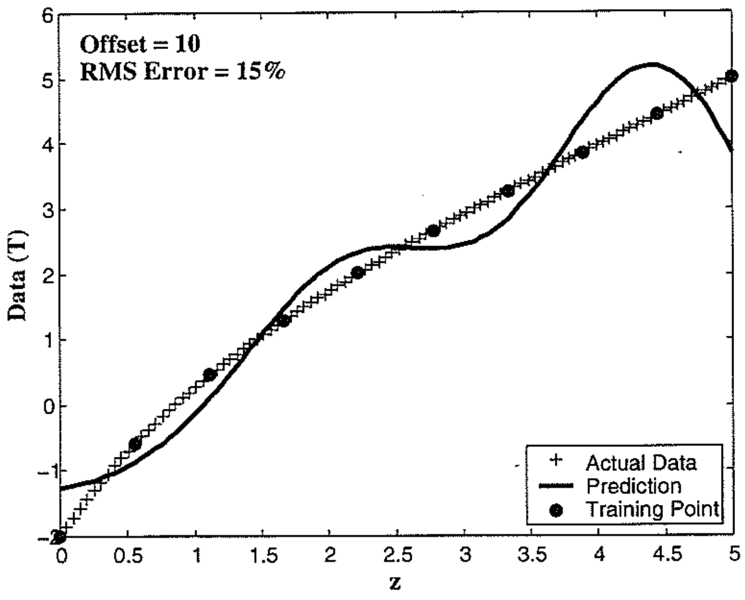


Figure 9: Radial basis networks: (a) typical plots of the radial basis transfer function with various offsets, the function becomes sensitive to more of the input space as the offset is increased, and (b) for a radial basis network that has an optimal offset, a comparison of the original data series and the network prediction when the data are noise-free. The training data points are indicated on the plot.

(a)



(b)

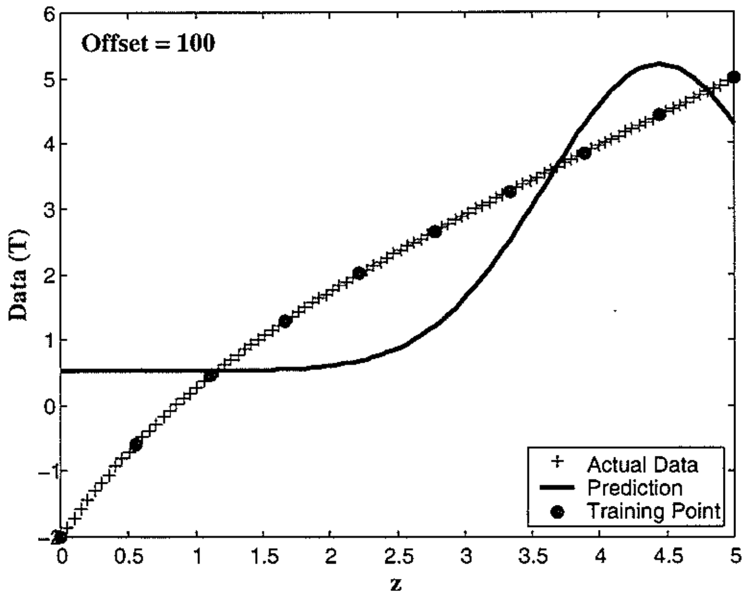
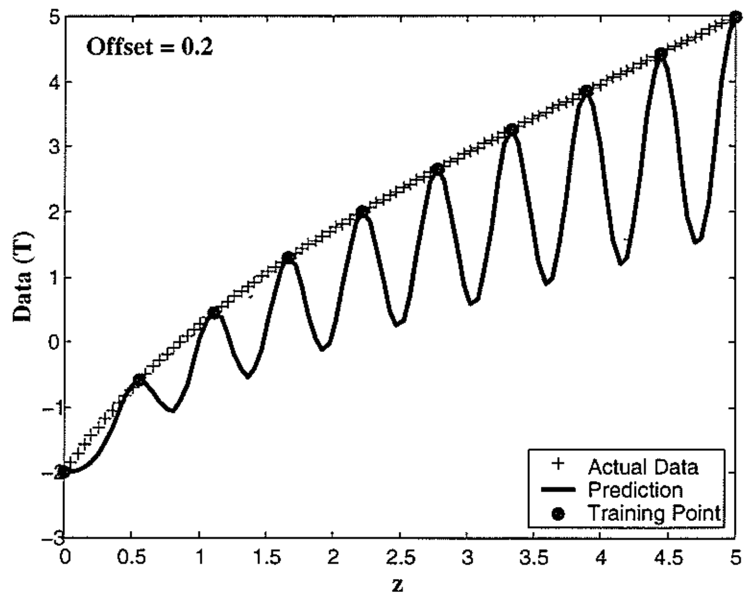


Figure 10: For a radial basis network that has too large an offset, a comparison of the original data series and the network prediction when the data are noise-free: (a) the offset is too large and the resulting output is too smooth, thus underfitting the data, and (b) the offset is so large that the network is no longer sensitive to the input. The training data points are indicated on the plots.

(a)



(b)

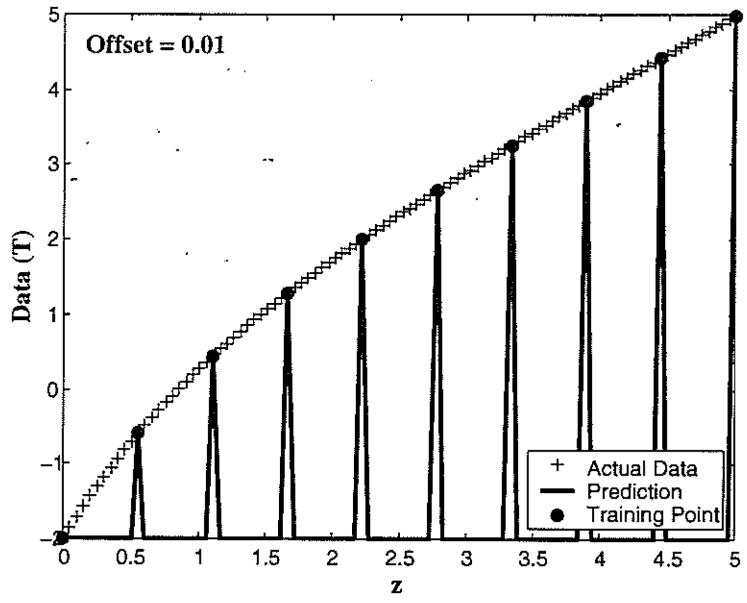


Figure 11: For a radial basis network that has too small an offset, a comparison of the original data series and the network prediction when the data are noise-free: (a) the offset is too small and the resulting output is poor for everything but the zones close to the training data since the network is sensitive only to narrow zones around its training points, and (b) with an even smaller offset, the zones of sensitivity become even narrower. The training data points are indicated on the plots.

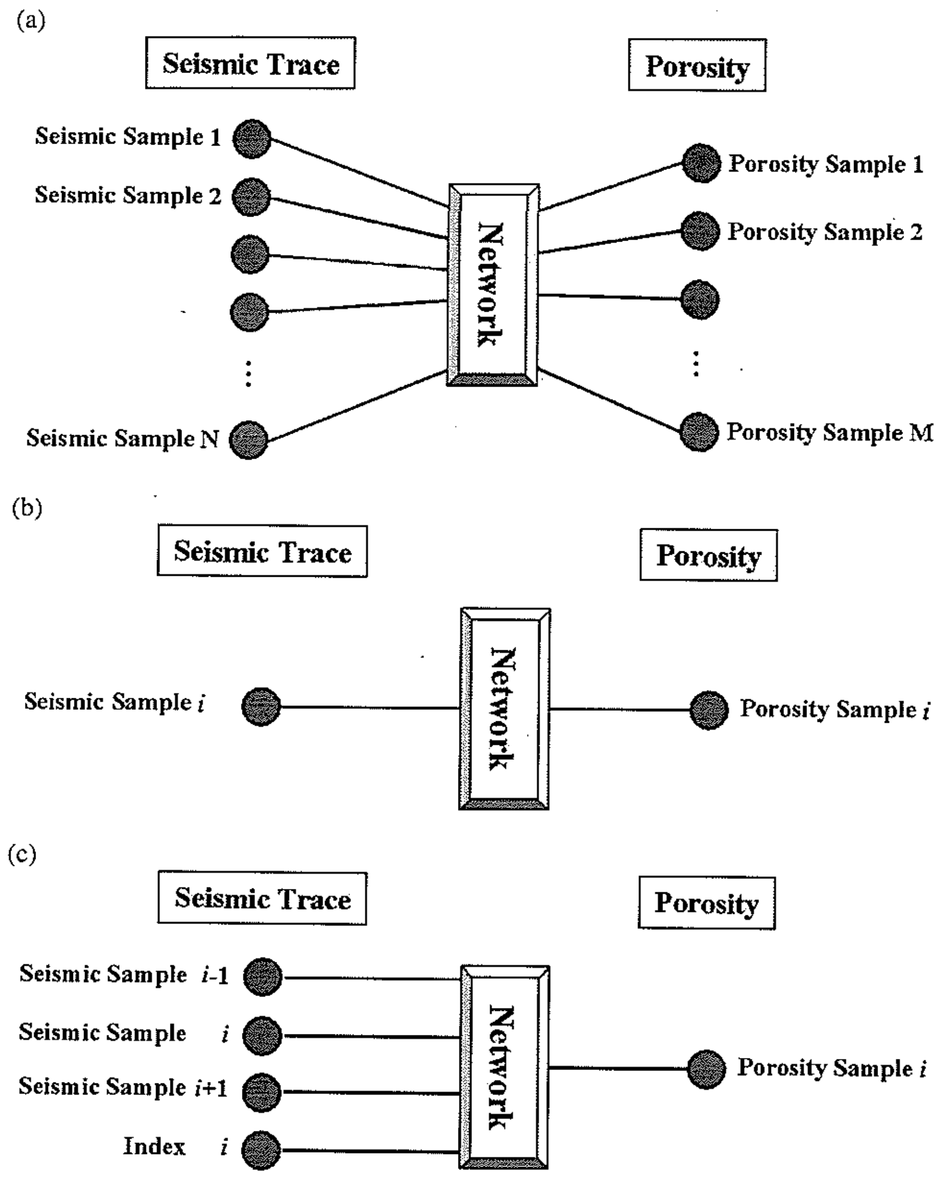


Figure 12: Schematic of the input/output architectures: (a) columnar architecture, where the number of input nodes need not be equal to the number of output nodes, but large input vectors require a large back-propagation network; (b) simple serialized architecture; and (c) expanded serialized architecture. For serialized architectures, the number of samples in the input vector must be equal to the number of samples in the output vector, but the size of the back-propagation network is independent of the number of samples in the trace or number of traces in the training data set.

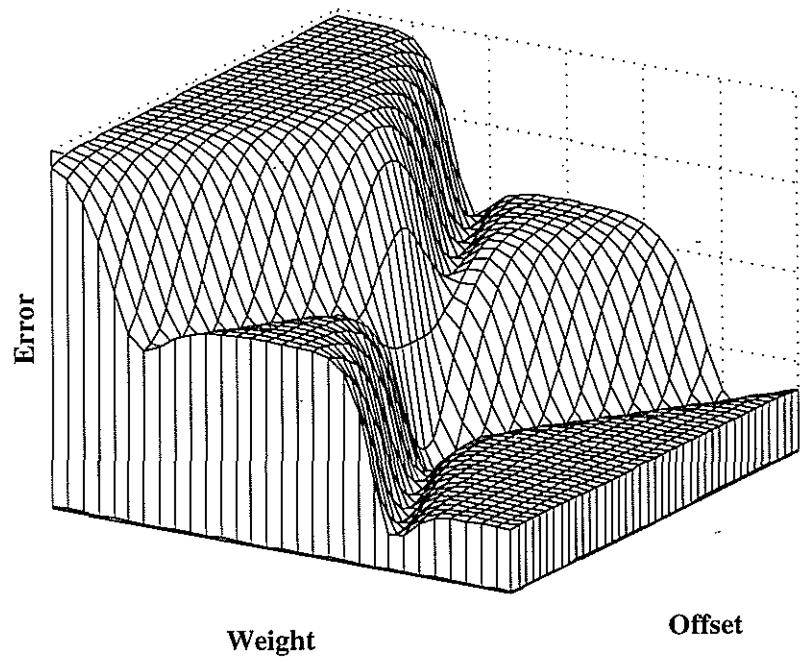


Figure 13: Error surface for a simple one-node network with respect to the two network coefficients (weight and offset). Note how complicated the surface looks, even for such a simple network.