Process Modeling Using Stacked Neural Networks

Dasaratha V. Sridhar and Richard C. Seagrave
Dept. of Chemical Engineering, Iowa State University, Ames, IA 50011

Eric B. Bartlett
Dept. of Mechanical Engineering, Iowa State University, Ames, IA 50011

A new technique for neural-network-based modeling of chemical processes is proposed. Stacked neural networks allow multiple neural networks to be selected and used to model a given process. The idea is that improved predictions can be obtained using multiple networks, instead of simply selecting a single, hopefully optimal network, as is usually done. A methodology for stacking neural networks for plant-process modeling has been developed. This method is inspired by the technique of stacked generalization proposed by Wolpert. The proposed method has been applied and evaluated for three example problems, including the dynamic modeling of a nonlinear chemical process. Results obtained demonstrate the promise of this approach for improved neural-network-based plant-process modeling.

Introduction

In the chemical industry, nonlinear models are typically required for process control, process optimization, and prediction of process behavior. Development of such models is often a difficult task for processes that are complex or poorly understood. When theoretical modeling is difficult, empirical, data-driven modeling provides a useful alternative. In recent years, artificial neural networks (ANNs) have been proposed as a promising tool for identifying empirical process models from process data (Bhat and McAvoy, 1990; Pollard et al., 1992). Neural networks are very useful because of their ability to model complex nonlinear processes, even when process understanding is limited (Mah and Chakravarty, 1992). These neural network models can be used for prediction, provided the process correlation structure does not change (MacGregor, 1991). Typically, the main objective in ANN modeling is to accurately predict steady-state or dynamic process behavior in order to monitor and improve process performance. Some of the important applications of ANNs in chemical engineering include the following:

- Fault diagnosis in chemical plants: Hoskins and Himmelblau (1988); Venkatasubramanian and Chan (1989)
- Dynamic modeling of chemical processes: Bhat and McAvoy (1990); Ydstie (1990); Pollard et al. (1992)
- System identification and control: Psichogios and Ungar (1991)
- Sensor data analysis: Kramer (1992)
- Prediction of product quality: Joseph et al. (1992)
- Chemical composition analysis and property prediction: McAvoy et al. (1989); Piovoso and Owens (1991)
- Inferential control: DiMassimo et al. (1992)

The typical approach to ANN-based process modeling has been to consider a number of candidate models, and to select one model that is expected to best predict the process outputs, given the process inputs. The selected model is the one that is expected to have least prediction error in the future. The expected prediction error of the candidate models is usually computed by evaluating them on data available for testing the model. When a separate data set is not available for selecting the model, cross validation (Weiss and Kukilowski, 1991) can be used. Cross validation allows model selection by using the same sample for model development as well as to provide a reasonable estimate of the expected prediction error in the future. Using a single optimal model implicitly assumes that one ANN model can extract all the information available in the data set and that the other candidate models are redundant. In general, there is no assurance that any individual model has extracted all relevant information from the data set. It is well recognized in the

Correspondence concerning this article should be addressed to E. B. Bartlett.

forecasting literature that identifying and using a single model is suboptimal for prediction (Bates and Granger, 1969; Granger and Ramanathan, 1984). Clemen (1989) provides a comprehensive review on the use of multiple models for forecasting. The primary conclusion of Clemen's work is that combining multiple models usually leads to increased forecast accuracy. Combining ANN models is based on the premise that different neural networks capture different aspects of process behavior: Aggregating this information should reduce uncertainty and provide more accurate predictions. Hence stacking or combining different models can be beneficial for predictive modeling. Stacked neural networks (SNNs), introduced here, allows multiple neural networks to be selected and combined in an attempt to obtain a better predictive model. The methodology is inspired by the general method for combining models known as stacked generalization (Wolpert, 1992). In this work, any empirical modeling approach based on stacked generalization will be referred to as stacked modeling or stacking.

The remainder of this article is organized in the following manner. First the rationale for combining ANN models is presented, followed by a discussion of stacked generalization. Second, the methodology for implementation of SNNs is provided. The idea of stacked modeling is then illustrated using a simple example. Following this, the feasibility of SNNs is explored through application to two examples, including a dynamic process modeling problem. Results obtained demonstrate that SNNs can achieve highly improved performance as compared to selecting a single optimal network using crossvalidation.

Combining ANN Models

Combining models to improve prediction accuracy is an idea that appears to have originated with the work of Bates and Granger (1969). Bates and Granger combined two different models for forecasting a time series and reported improved predictions using the combined model. Since Bates and Granger's pioneering efforts, a large amount of work has been performed on combining models for forecasting. It is now widely accepted that combining of forecasts is a pragmatic and useful approach for producing better forecasts (Granger, 1989).

In the classic approach to modeling, one typically assumes that a certain ideal model is "true" and, based on available evidence, an actual model is rejected or accepted. If it is felt that the model is misspecified, a better model is sought. This approach is appropriate when there is theory that guides the specification and evaluation of the models (Diebold, 1989). However, a more pragmatic approach can be taken: models can be combined to improve predictions. Combining models is equivalent to admitting that a "true" single model is not easily attainable and that multiple models should be viewed as different pieces of information that can be integrated. Pooling models has a short-term perspective with an emphasis on real-world applications (Winkler, 1989).

The preceding arguments hold for neural-network-based empirical modeling of plant processes. As in time-series forecasting, the emphasis is on maximizing accuracy on future predictions. In ANN modeling, it is difficult to specify the optimum ANN architecture *a priori*. Therefore, neural networks can be viewed as inherently misspecified models since they are an approximation of the underlying process (White, 1989). Hence, there is no reason to believe a given network architecture represents the actual underlying structure of the data-generating process. Different neural networks are capable of approximating different classes of functions: a sufficiently large network can approximate any arbitrary function (Cybenko, 1989; Hornik et al., 1989). However, because of finite sample sizes, one tends to use architectures that are smaller than required. Smaller networks have lesser variance but show greater bias than the larger networks. This problem is well-known as the bias-variance dilemma (Geman et al., 1992). For a given problem, a number of network architectures are evaluated and a hopefully optimal architecture is selected, based on its performance on some test set. In this article we define an optimal network architecture as one that performs better than any other model over the entire input space. Optimality in this sense cannot be guaranteed by simply selecting the model that produces the least average prediction error over some test set. It is quite possible that different networks perform better in different regions of the input space. This situation can arise because

- The optimal architecture was not considered by the modeler or could not be found by the automatic network construction methods used. Methods like cascade correlation (Fahlman, 1990) and dynamic node architecture (Basu and Bartlett, 1994) can automatically develop good neural network models. However, optimality as defined earlier cannot be guaranteed, as all these methods typically focus on minimizing the average error over some training set, and not over the entire input space.
- As explained by Hansen and Salamon (1990), multiple networks are desirable because optimization of network parameters is a problem with many local minima. Even for a given architecture, final parameters can differ between one run of the algorithm to the next. For example, varying the initial starting conditions for network training can lead to a different solution for the network parameters. This tends to make the errors produced by the different networks uncorrelated. Therefore, the different networks might make independent errors on different subsets of the input space.
- Different activation functions and learning algorithms can lead to different generalization characteristics, and no one activation function or learning algorithm may be uniformly best (Hashem, 1993).
- The convergence criteria used for network training can lead to very different solutions for a given network architecture. Networks can be trained using convergent training, which means that the neural network is trained to convergence on the entire training data set. Alternately, stopped training can be used. Stopped training involves monitoring the performance of the network as it trains on a training set, and training is stopped when the error on the crossvalidation set reaches a minimum. Networks trained by stopped training could be very different from those trained using convergent training (Finnoff et al., 1993).

Therefore, it is possible that a combination of ANN models could outperform a single model. If better performance is achieved using multiple models, it implies a better single model can be found. However, finding such a model requires further time and effort, with no assurance that the better single model will be found. Instead of searching for a better model, combining existing neural networks can provide a practical approach to develop a better overall model for prediction by using the models already at hand.

Stacked Generalization

To maximize the benefits of combining models, effective schemes for combining them must be used. From the forecasting literature, the usual approach is to model the true output as a linear combination of the outputs of the individual models. This can be shown to have lesser mean square error than any of the individual models (Granger and Ramanathan, 1984). In the context of neural networks it has been seen that a linear combination of neural networks often improved predictions (Hashem, 1993; Perrone and Cooper, 1993). The primary limitation of these methods is that model selection is not considered in conjunction with model combination. When using nonparametric methods like neural networks, it is possible that overparametrized models may be included in the combination. It is often difficult to judge which ANN model is a good generalizer. Hence models with poor generalization could be included along with good models, causing performance of the combined model to be poor. For instance, Hashem (1993) uses a weighted average of the outputs of the candidate networks, estimated from the training data. To check the robustness of the combined model he proposes testing the individual models and the combined model on a crossvalidation set. While determining the combination, the reliability of the individual networks is not taken into account. Hashem later proposes algorithms that check if performance of the combined model is better than the individual models. In Hashem's work, the issue of whether the combined model is a good predictor is addressed only after a set of models has been selected and combined.

Stacked generalization (Wolpert, 1992) has been proposed as a technique for combining models. Preliminary efforts by Sridhar et al. (1995) showed promise for improved predictive modeling using stacked generalization. The novel feature of stacked generalization is that it attempts to simultaneously solve the problem of model selection and estimation of model combinations to improve model predictions. This feature is inherent in the stacked generalization procedure, because it has been developed as an extension of nonparametric statistical methods used for model selection. The idea is that by analyzing the performance of the original models, using a statistical resampling scheme, one can estimate a combination of the original models that will maximize prediction accuracy in the future. Stacked generalization is based on the assumption that the resampling methods provide more information than simply specifying which model is the best: it can allow estimation of combinations of models that can provide more accurate predictions. The data used for fitting a stacked generalization model are generated using the same method as for cross validation. Stacked generalization can be viewed as a more general solution to estimating an optimum predictive model, while accounting for the bias-variance trade-off. Cross validation simply selects one model that minimizes the expected prediction error. Thus cross validation provides a solution to minimizing future prediction error subject to the assumption that the minimization needs to be achieved by a single model. In fact, cross validation can be viewed as a special case of stacked generalization. Stacking relaxes the assumption that one model needs to be selected and used for prediction. Multiple models can be selected and optimally combined to maximize prediction accuracy. The main goal is to maximize expected generalization in the future. If a combination of models reduces the expected prediction error, then it is preferable. Hence, stacking provides a method to estimate combinations of models that can in theory maximize generalization accuracy.

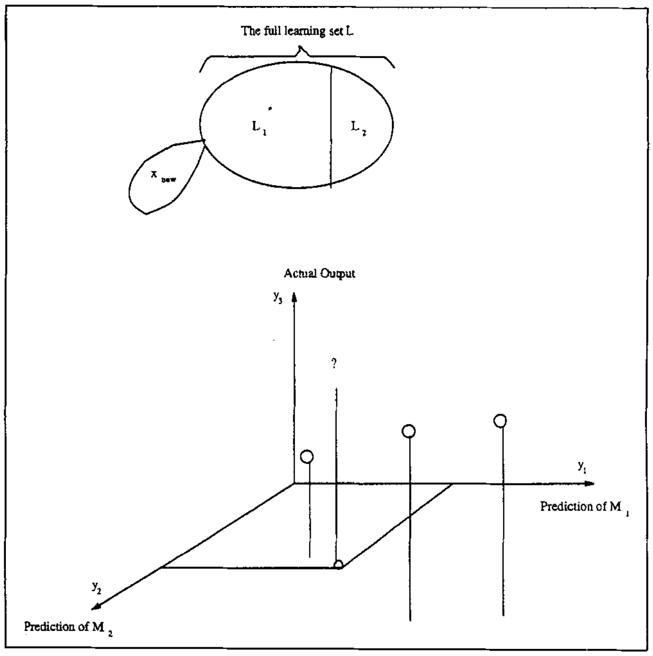


Figure 1. Stacked generalization.

The details of stacked generalization can be found in Wolpert's article: in this article, we provide a brief overview of this technique. The objective of stacked generalization is to fit a model for the true output. The inputs to the stacked generalization (SG) model are the outputs of the different models. The concept of stacking different models is shown in Figure 1. The models that are developed from the original training set are referred to as the level-0 models. Consider the simplest case when we have two level-0 models, M_1 and M_2. Let the training set be partitioned into L_2, containing one pattern (x, y) and L_1, containing the remaining patterns. Both M_1 and M_2 are trained on L_1 and then tested on L_2. Let the outputs of the two models be y_1 and y_2, while the true output is y. This information can be considered to be input-output information in a different space, with two inputs (the predictions of the two models) and one output (the actual output). Repeating the previous procedure with each training pattern in L_2, we obtain a data set L' containing the predictions of the two models and the true output. This data constitutes a new data set that can be used to train a new model M_3 to predict y given y_1 and y_2. M_3 is known as the level-1 model. For any novel input vector x_{new}, the predictions y_1 and y_2 are obtained and these predictions are combined by M_3 to produce the final prediction y_3. Extension of this approach, when more than two models are considered, is

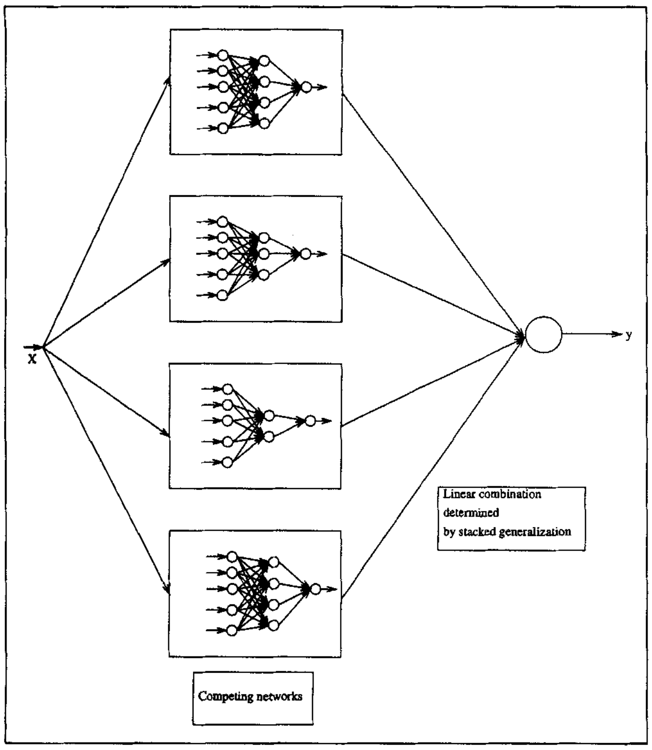


Figure 2. Architecture for stacked neural networks.

based on the same principles discussed earlier and is fairly straightforward.

Stacked Neural Networks

When the candidate level-0 models are ANNs and they are combined using stacked generalization, we define the resulting model as a SNN. In this work, attention is restricted to level-0 models that are single-layer backpropagation networks. Of course other level-0 models such as radial basis function networks, general regression networks, and backpropagation networks with more than one hidden layer can be used. The methodology developed here to stack level-0 neural networks can easily be extended to include any level-0 model. The level-1 models considered in this article are linear models. The proposed architecture for SNNs is shown in Figure 2.

Assume that a data set $D((y_n, x_n):1 \leq n \leq N)$ has been collected on some process of interest. Here, y denotes the output variable, x denotes the input vector, N is the number of training patterns, and n is the pattern index. It is assumed that M neural networks $(N_1, N_2, \ldots, N_M)$ are the candidate level-0 models. The level-0 data set will be denoted by D_{L0}. The data used to develop the level-1 model will be denoted by D_{L1}. The following algorithm is used for both stacking the ANN models and for selecting the best ANN model using k-fold cross validation. The approach can be easily extended for problems with multiple outputs.

1. Set the level-0 data set D_{L0} equal to the data set D. Train the M level-0 networks using D_{L0}. Denote the jth network trained on D_{L0} as $N_j(D_{L0})$, and denote the set of these level-0 networks as $N(D_{L0}) = \{N_j(D_{L0}):1 \leq j \leq M\}$. The $N(D_{L0})$ should be saved for use in the SNN model.

2. Divide the data set D into k disjoint subsets with an equal number of patterns, $D_1, D_2, \ldots, D_k$, as in k-fold crossvalidation. Define CV_i as $D - D_i$. CV_i contains all the patterns in the data set D except the patterns in D_i.

3. Repeat the following for $1 \leq i \leq k$: train the M candidate ANN models using the data set CV_i. Denote the jth network trained on CV_i as $N_j(CV_i)$, and denote the set of these trained networks as $N(CV_i) = \{N_j(CV_i):1 \leq j \leq M\}$. Note that the $N(CV_i)$ have the same architecture as the M level-0 networks; however, they are trained on data set CV_i. The sole reason for developing the networks $N(CV_i)$ is to construct the level-1 data set D_{L1}. Recall $N(CV_i)$ on the data set D_i. Denote the prediction of the jth network for pattern n in data set D_i as yp_{nj}. For pattern n, the output of the candidate models is collected in an M-dimensional vector $\boldsymbol{yp}_n = \{yp_{nj}:1 \leq j \leq M\}$. The actual output y_n and the network outputs $\boldsymbol{yp}_n$ constitute the output and input, respectively, for the nth pattern in data set D_{L1}. Discard the networks $\{N_j(CV_i):1 \leq j \leq M\}$.

4. Step 3 produces the level-1 data set $D_{L1}(y_n, \boldsymbol{yp}_n)$. Data set D_{L1} contains the true output and the predictions of the M models, for all the N training patterns.

5. Compute the prediction sums of squares errors (PRESS) for each of the M networks in $N(D_{L0})$. PRESS for network j is calculated as

$$\mathrm{PRESS}_j = \sum_{n=1}^{N} (y_n - yp_{nj})^2. \tag{1}$$

The network with the minimum PRESS is considered to be the optimal single network.

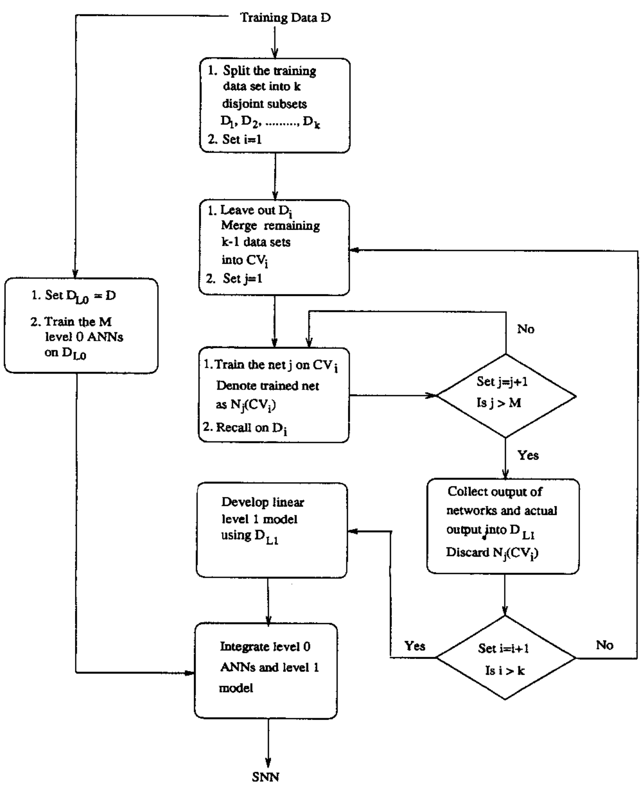
6. For developing the stacked model, the following level-1 model will be used to combine the M level-0 neural networks in $N(D_{L0})$.

$$y = \sum_{j=1}^{M} \alpha_j yp_j, \tag{2}$$

where all the model coefficients α_j are required to be positive. Similar combining rules have been used for combining models for forecasting (Bates and Granger, 1969; Granger and Ramanathan, 1984) and more recently for combining regression models (Brieman, 1992). D_{L1} is used to estimate the parameters of the level-1 model. The objective function to be minimized for the level-1 model is the output sums of squares errors over D_{L1}. More complex level-1 models can also be used if desired.

7. For prediction on a novel input pattern, the input is fed through all of the candidate models. The outputs of the level-0 neural networks are combined using the level-1 model to produce the final prediction.

The proposed method is shown in Figure 3. It should be noted that the method just outlined can be used not only with k-fold cross validation, but with any data-based model-selection technique. One commonly used approach is to divide the data set into two data sets: D_1 used for training the candidate models and D_2 for testing the developed models. Development of SNNs in this case is a straightforward variation of the method just discussed. D_1 constitutes the level-0 data set D_{L0} and testing on D_2 produces the level-1 data set D_{L1}.



Training Data D
1. Split the training data set into k disjoint subsets D1, D2, ..., Dk
2. Set i=1
1. Leave out Di Merge remaining k-1 data sets into CVi
2. Set j=1
1. Set DL0 = D
2. Train the M level 0 ANNs on DL0
1.Train the net j on CVi Denote trained net as Nj(CVi)
2. Recall on Di
Set j=j+1 Is j > M
No
Yes
Collect output of networks and actual output into DL1 Discard Nj(CVi)
Develop linear level 1 model using DL1
Integrate level 0 ANNs and level 1 model
Set i=i+1 Is i > k
Yes
No
SNN

Figure 3. Developing the stacked neural network.

The methodology for developing stacked models will first be illustrated using a simple example. Application of the stacked neural network (SNN) approach to two example problems, with a discussion of results, will then be presented.

Example 1. The first example involves identification of a simple function, using linear regression and nearest-neighbor models as the candidate models. Neural networks are not considered as candidate models in this example, since it is intended to illustrate the idea of stacked modeling in a simple manner. Assume that the true parent function generating the data is

$$w = x + y + z, \tag{3}$$

where x, y, z are independent variables assumed to be drawn from a uniform distribution on the interval $[0,1]$. A data set of eight patterns corresponding to the eight corners of the unit cube, defined by the independent variables, has been collected for identifying the function (Table 1). The following models w_1, w_2, w_3 (level-0 models) are considered as candidates for identifying the true function w:

1. $w_1 = x$
2. $w_2 = ay + bz$
3. Nearest-neighbor model using variables y and z. For the nearest-neighbor model w_3 = output for the nearest pattern in the data set. For example, for the pattern (0.05, 0.1, 0.1) the closest training pattern is (0, 0, 0). Therefore the output of the nearest-neighbor model is 0.

These functions are deliberately chosen to illustrate the central ideas behind stacked modeling. It is obvious that each of the models alone is incapable of representing the true function. Models 1 and 2 together contain all the information required to identify the true function. Model 3, which is the nearest-neighbor model, provides a perfect fit to the training data, unlike models 1 and 2. However, model 3 is not a good approximation to the true function and is in fact redundant, given models 1 and 2. Ideally, it is desired that stacking the three candidate models should result in models 1 and 2 being combined while model 3 should be rejected. Table 1 also shows the outputs of the models obtained using leave-one-out cross validation. w_{ij}^{cv} is the prediction of model i on the jth pattern when that pattern is left out and the model is developed using the rest of the data. Table 2 shows the expected prediction error, E(MSE), for all three models, computed using cross validation. The performance of the three models, evaluated on 10,000 test patterns is also shown. The performance measures used are the mean square error (MSE) and the linear correlation coefficient (r), between the desired output and the model's actual output. Model 2 is identified by cross validation as the best single predictor, as it has the least expected prediction error. Model 2 is indeed the best individual predictor, as it has the least error on the test data set. However cross validation has ignored the useful information in model 1. Clearly using a single model is suboptimal in this case. Stacked modeling, on the other hand, attempts to identify a combination of the three models. Fitting the combining rule defined by Eq. 2 to the level-1 data shown in Table 1, results in the following level-1 model:

$$w_s = 1.22 w_1 + 0.728 w_2 . \tag{4}$$

Substituting for the functions w_1 and w_2 in terms of the independent variables x, y, and z gives the final model:

$$w = 1.22x + 0.92y + 0.92z . \tag{5}$$

The expected prediction error and the error on the test data set for the stacked model are both less than that of any of the other single models. Combining the three models results in a stacked model that is quite close to the actual func-

Table 1. Training Data and Cross Validation Results for Example 1

	Training Data				Cross Validation		
Pattern	x	y	z	w	w_{1j}^{cv}	w_{2j}^{cv}	w_{3j}^{cv}
1	0	0	0	0	0.0	0.0	1.0
2	0	0	1	1	0.0	1.5	2.0
3	0	1	0	1	0.0	1.5	2.0
4	0	1	1	2	0.0	3.0	3.0
5	1	0	0	1	1.0	0.0	0.0
6	1	0	1	2	1.0	1.0	1.0
7	1	1	0	2	1.0	1.0	1.0
8	1	1	1	3	1.0	2.5	2.0

Table 2. Comparisons of Models for Example 1

Model	E MSE	MSE Test Data	r Test Data
$w = x$	1.50	1.33	0.57
$w = 1.33y + 1.33z$	0.59	0.36	0.81
Nearest neighbor	1.00	0.50	0.70
Stacked model	0.02	0.08	0.99

tion. The stacked model is better than any of the individual models investigated. It can be seen from Eq. 4 that model 3 is completely rejected, and models 1 and 2 are combined to effectively approximate the true function. This example illustrates the usefulness of combining when the candidate models are misspecified. If we had specified one of the candidate models more accurately, we could have exactly identified the true function. In this example, if we had attempted to fit

$$w = ax + by + cz \tag{6}$$

the true function would have been identified exactly. If we can accurately specify models, combining would not be required. In the present example, it is possible to determine the true model. However, in complex real-world problems model specification, as well as determining the optimal model, can be quite difficult. In such situations we would expect that stacked modeling will provide us with better predictions by integrating all the useful information that the various candidate models may have acquired. It should also be noted that in the preceding example none of the models used all the variables. It is therefore obvious that the models have information that is independent of each other. The problem was designed to illustrate the benefits of stacked modeling in a simple manner. The use of Eq. 2 for stacking is very effective for this example. In general, it is not known what level-1 model is most effective for stacking the candidate models. However, for the sake of simplicity, we have restricted our attention in this article to the linear level-1 model.

Example 2. It is often possible that all the candidate models considered for approximating a given process use all of the independent variables. The independence of information contained in the outputs of these models arises from the inherent differences in the approximation abilities of the candidate models. Of course, some correlation between the candidate models is inevitable, since they all make use of information from the same data. However, the information that is captured could be different, and this is what we seek to take advantage of and integrate through stacked modeling. For this example, the objective is to estimate the six-dimensional additive function,

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + 0x_6. \tag{7}$$

The x_i were generated from a uniform distribution in the six-dimensional hypercube. This function has been used as a benchmark for MARS (Friedman, 1991), and more recently for comparing neural network and statistical methods (Cherkassky et al., 1995). We follow an approach similar to Cherkassky et al. for this study. The training data set S_1 consisted of 100 samples, and constitutes the level-0 data. The added noise was Gaussian with a signal-to-noise ratio (SNR) of 4.0. The SNR is defined as

$$\text{SNR} = \sigma_y / \sigma_N, \tag{8}$$

where, σ_y is the standard deviation of y and σ_N is the standard deviation of the noise. In addition we also generated a data set S_2 consisting of 50 patterns. Data set S_2 was used for model selection and model integration. A test data set S_3 of 10,000 patterns without any noise was generated to measure the performance of the different models. The performance metric used was the normalized root-mean-square error (NRMS), which is the average root-mean-square error on the test data set normalized by the standard deviation of the data set. NRMS represents the fraction of unexplained standard deviation. Four different single hidden layer backpropagation networks N_1, N_2, N_3, and N_4 with 5, 10, 20 and 40 hidden nodes, respectively, were developed. Networks were trained using the scaled conjugate gradient algorithm (Moller, 1993). N_1 was trained on all 100 patterns. For the remaining networks, stopped training was used. These networks were trained on 70 randomly selected patterns and their performance was monitored on the remaining 30 patterns. The weights that gave the least error on the 30 patterns were considered to be the optimal weights for the networks. Performance of the candidate networks on the data set S_3 was evaluated by computing the NRMS. The different models were then stacked to obtain the SNN model. The NRMS for the SNN was also computed. The performance comparisons are shown in Table 3. Using cross validation, N_2 is selected as the best model. N_2 has an NRMS of 0.34 on the test data. N_1 also performs similar to N_2. The larger networks N_3 and N_4 perform poorly and appear to be overparameterized. Testing on S_3 shows that these models are indeed poor generalizers. The competing networks were then combined using SNN and resulted in the following level-1 model:

Table 3. Comparisons of Models for Example 2

Network	RMSE Data Set S_2	RMSE Data Set S_3	NRMS on S_3
6×5×1	6.24	5.13	0.32
6×10×1	6.06	5.33	0.34
6×20×1	14.15	12.60	0.80
6×40×1	12.54	11.50	0.73
SNN	5.40	3.94	0.25

$$y_s = 0.469 * y_1 + 0.531 * y_2, \tag{9}$$

where y_s is the output of the SNN and the y_i are the outputs of the ith network. The RMSE of the SNN on the test data was found to be 0.25. Thus, a reduction of 26% in the prediction error was obtained using SNN over the cross-validation-selected network. It is interesting to note that Cherkassky et al. (1995) reported an NRMS of 0.319 for this problem. Their results are slightly better than those obtained in this work, with N_1 (NRMS = 0.32) and N_2 (NRMS = 0.34). However, SNN helped us significantly improve the model performance. SNN had NRMS of about 22% less than that reported in Cherkassky's work. N_3 and N_4 were rejected by the SNN model, while N_1 and N_2 were combined to provide an improved model. SNN was able to successfully integrate the knowledge acquired by the candidate networks. In terms of overall performance, both N_1 and N_2 are equally good (NRMS within 6% of each other). Improved performance by combining the two models means that the models make errors of different types. Indeed the linear correlation coefficient between the errors made by the two models is very small ($r = 0.11$). As pointed out earlier, a number of reasons can cause networks to perform very differently. In this case, the network architecture, the use of both convergent and stopped training, and the different weight initializations led to the significant differences between models N_1 and N_2. The error independence of the two models allowed SNN to integrate the two models and develop a better model. It can be argued here that a single network could have been found that has the same performance as the SNN. Indeed this is theoretically possible. Obtaining this optimal model would, however, require searching for such a model. The SNN method avoided a further search and provided a direct approach toward obtaining a better model.

Dynamic Modeling Example

This example is intended to demonstrate the performance of SNNs for a more practical modeling problem. In this example, the performance of the SNNs was examined for training sets of different sizes and various noise levels.

The process considered is an ideal, adiabatic continuous first-order exothermic reaction in a continuous stirred-tank reactor (CSTR), shown in Figure 4. This process has been used as a benchmark for internal model control (Economou, 1986), radial basic-function modeling (Leonard et al., 1992), and comparison of backpropagation networks and multivariate adaptive regression splines (DeVeaux et al., 1993). The feed to the CSTR is pure A and the desired product is R. The system is simulated by the following coupled ordinary differential equations.

$$\frac{dA_o}{dt} = \frac{A_i - A_o}{\tau} - k_1 A_o + k_{-1} R_o$$

$$\frac{dR_o}{dt} = \frac{R_i - R_o}{\tau} + k_1 A_o - k_{-1} R_o$$

$$\frac{dT_o}{dt} = \frac{-\Delta H_R}{\rho C_p}(k_1 A_o - k_{-1} R_o) + \frac{T - T_o}{\tau}, \qquad (10)$$

where

$$k_1 = c_1 \exp\left(\frac{-Q_1}{RT_o}\right)$$

$$k_{-1} = c_{-1} \exp\left(\frac{-Q_{-1}}{RT_o}\right).$$

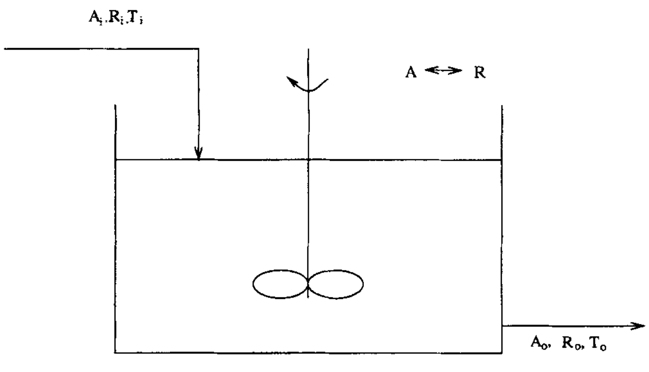


Figure 4. CSTR.

Table 4. Constants and Steady-State Operating Conditions for CSTR

τ	60s
c_1	5×10^3 s^{-1}
c_{-1}	1×10^6 s^{-1}
Q_1	10,000 cal mol^{-1}
Q_{-1}	15,000 cal mol^{-1}
R	1.987 cal mol$^{-1}\cdot$K^{-1}
ΔH_R	5,000 cal mol^{-1}
ρ	1 kg/L
C_p	1,000 cal$\cdot$kg$^{-1}\cdot$K^{-1}
A_i	1.0 mol/L
R_i	0.0 mol/L
A_o	0.492 mol/L
R_o	0.508 mol/L
T_i	427 K
T_o	430 K

The variables A_o, R_o, and T_o represent the state variables of the system and are the feed species concentration, the output product species concentration, and the reactor's temperature, respectively. The goal of the modeling is to learn the open-loop relationship between the controlled variable, which is the concentration of species R, and the manipulated variable, which is the temperature of the feed stream, T_i. It is desired to predict the concentration of species R at the next time step, given the state variables A_o, R_o, T_o and the manipulated input T_i. The input and output dimensionality of the system are 4 and 1, respectively. The steady-state operating point and the process parameters are as given in the article by Economou et al. (1986), and are shown in Table 4. A method similar to that in DeVeaux et al. (1993) was used to generate the training data for the neural network modeling. The preceding set of differential equations was integrated, with T varying randomly with a uniform distribution within 15% of the steady-state operating point. Sampling time for the process was 30 seconds. During the simulation, the system's state variable as well as manipulated input were recorded. At the end of the sampling instant, the system's response was also recorded. Measurement noise was added to the noise-free simulation. The noise was assumed to be Gaussian. A time-series plot of one realization of the data is shown in Figure 5. Training data sets S_1 of sizes 50, 70, 100, and 120 were generated at five different noise levels with standard deviations of 2.5%, 5.0%, 7.5%, 10%, 12.5% of the range of the output, respectively. Thus a total of 20 data sets were considered in this study. A noise-free test data set S_2 of 10,000 patterns was also generated in order to test and compare all the models developed.

In this analysis, the following terminology will be used:

1. Cross validation selected network (CVSN): This is the network that is selected using the cross validation scheme.

2. Stacked neural networks (SNN): SNN represents the stacked model, where the competing ANN models are stacked using the technique described earlier in this work.

3. Best *a posteriori* network (BAPN): After the CVSN and SNN have been developed and tested, all of the candidate models are tested on the test data S_2. The network with the least error on the test data set S_2 will be referred to as the best *a posteriori* network.

The CVSN and SNNs will be compared to the BAPN in

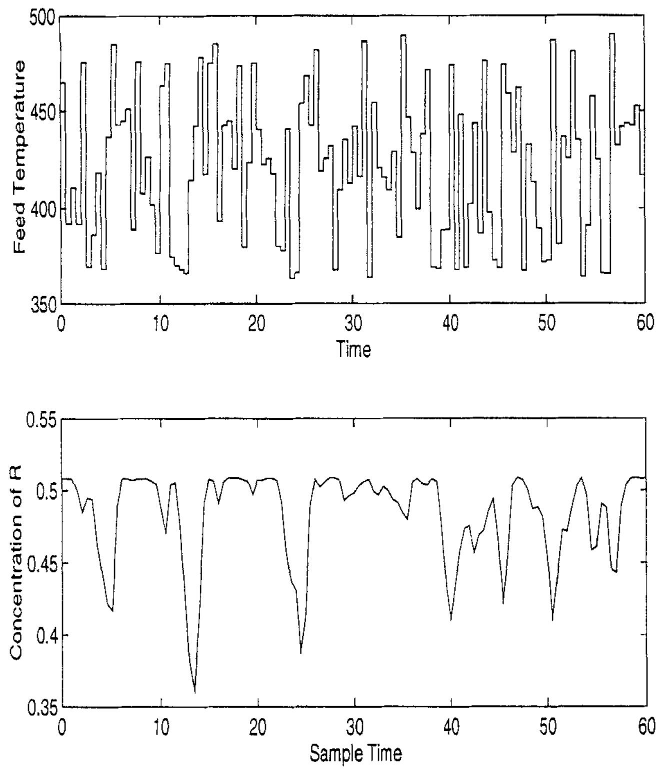


Figure 5. Feed temperature sequence and concentration response of CSTR.

the following analysis. The rationale for using the BAPN as a baseline for comparison is that BAPN represents the best performance that could have been obtained given the candidate networks and assuming we wanted to select and use only a single network as our model. The idea behind using cross validation is to identify the BAPN beforehand. However, the BAPN may or may not be successfully identified by cross validation. In either case, the CVSN can never be better than the BAPN. SNNs, on the other hand, have the potential to identify stacked models that may be better than the BAPN.

Eight neural network models $[N_i{:}1 \le i \le 8]$ with 1, 2, 3, 4, 8, 12, 16, and 20 hidden nodes, respectively, were considered as the level-0 models for modeling the process. For the training algorithm, the scaled conjugate gradient algorithm (SCGA) was used (Moller, 1993). The methodology described earlier in this work was used to stack the eight networks, and k-fold cross validation was used to identify the CVSN. The neural nets considered in this example had one hidden layer and used the hyperbolic tangent function as the activation function. Weight initialization for networks N_1, N_2, N_3, N_4 was performed using the method based on principal-component regression, proposed by Piovoso and Owens (1991). The remaining networks were initialized with random weights distributed uniformly on $[-0.1, 0.1]$.

After training, the performance of the SNN and the CVSN was tested on the 10,000 patterns. Finally, all eight networks were evaluated on the test data set to identify the BAPN. The mean absolute prediction error (MAPE) (Makridakis et al., 1983) was used for making the comparisons. Table 5 shows the CVSN, the MAPE for the CVSN, the MAPE for the SNN,

Table 5. Comparison of the Models for Continuous Stirred-Tank Reactor Data

Sample Size	% Noise	CVSN	MAPE CVSN	MAPE SNN	% Reduction in MAPE Using SNN
50	2.5	N_2	77.7	48.7	37
50	5.0	N_3	92.9	67.5	27
50	7.5	N_2	125.2	89.4	29
50	10.0	N_2	130.1	77.2	41
50	12.5	N_3	90.1	63.4	29
70	2.5	N_4	74.7	59.6	20
70	5.0	N_3	67.0	54.4	19
70	7.5	N_3	67.8	53.8	20
70	10.0	N_3	73.5	48.8	34
70	12.5	N_4	67.9	60.0	12
100	2.5	N_3	92.9	78.6	15
100	5.0	N_4	103.5	81.0	22
100	7.5	N_2	83.3	77.6	7
100	10.0	N_2	80.9	69.4	14
100	12.5	N_2	76.3	65.8	14
120	2.5	N_5	94.6	72.1	24
120	5.0	N_4	95.0	68.7	28
120	7.5	N_2	95.3	73.6	23
120	10.0	N_3	73.4	58.0	21
120	12.5	N_2	63.8	58.7	8

and the percent reduction in MAPE using SNN as compared to using the CVSN, for all 20 cases. For convenience, the numbers have been multiplied by 10,000. As can be seen from Table 5, SNNs outperformed the CVSN in all cases. Reduction in prediction errors achieved by using the SNN, as compared to using the CVSN, ranged from 7% to 41%.

Graphical comparisons of the SNN with the CVSN and the BAPN for the different cases are shown in Figure 6. For clarity, in all the figures, the predicted errors have been normalized with respect to the errors for the BAPN. When normalized in this manner, BAPN always has an average error of 1.0. The results shown in Figure 6 indicate that the behavior of k-fold cross validation tends to be erratic. Difference between the MAPE of CVSN and BAPN ranged from 0% to as high as 130%. There was also no apparent correlation between the performance of CVSN and the SNN with respect to sample size and noise levels. The reason for this behavior is that there are a number of other factors that can influence the level-1 data generated. The random partitioning of the data set into k data sets for cross validation and the random weight initialization used in neural network training strongly influence the level-1 data. Moreover, it is known cross validation estimate of prediction error can show high variance (Weiss and Kukilowski, 1991). These factors make it difficult to discern any systematic behavior in the performance of the CVSN and the SNN with respect to sample size and noise. However, these factors equally affect both SNN and CVSN, since they both make use of the level-1 data. Therefore, it is easier to compare the relative performance of SNNs with respect to CVSNs. In the following discussion we focus attention on the relative performance of SNN with respect to the CVSN and how it is affected by noise and sample size.

Figure 6a shows the performance comparisons of CVSN and SNN to BAPN for a sample size of 50, at different noise levels. The performance of CVSN for this sample size is good for low noise levels; however, it performed very poorly at high noise. For a noise level of 2.5%, a reduction of 32% in error

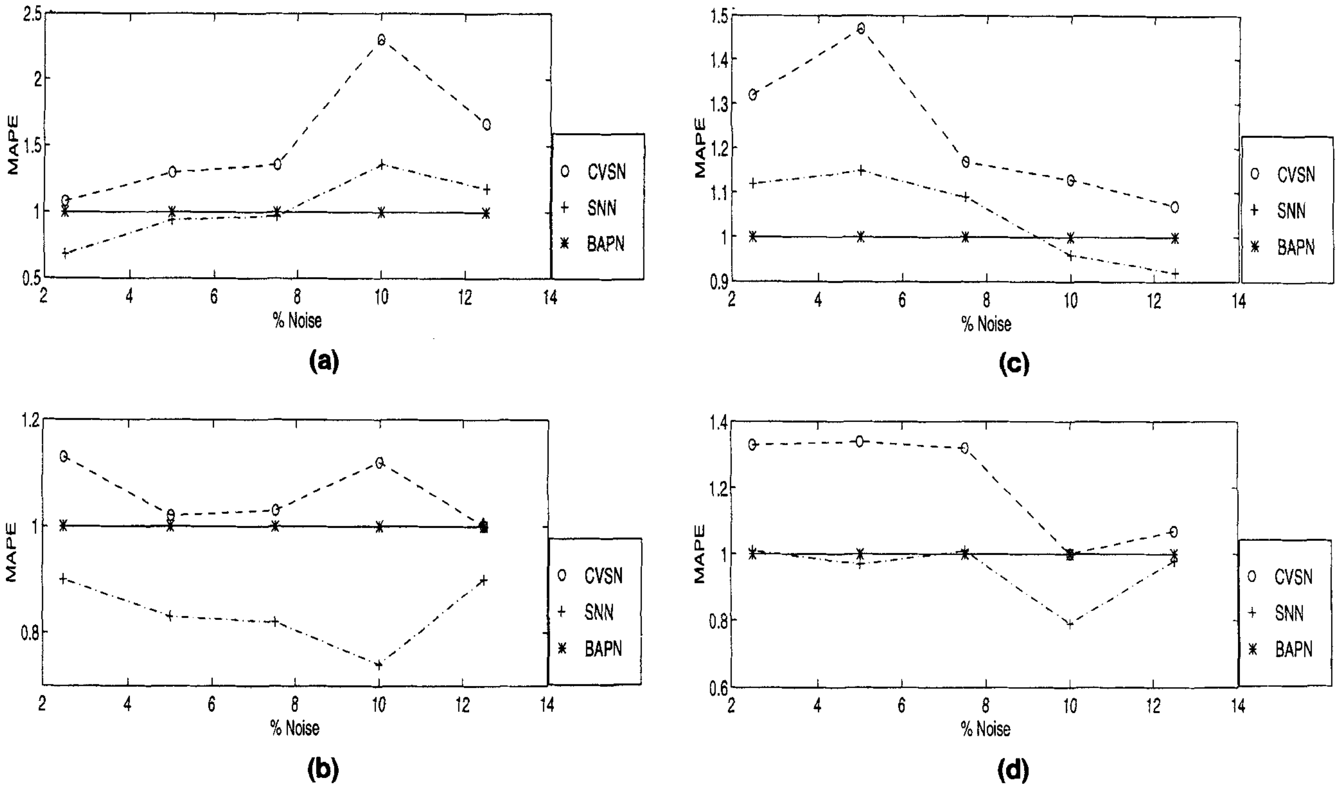


Figure 6. (a) SNN vs. CVSN for sample size = 50; (b) SNN vs. CVSN for sample size = 70; (c) SNN vs. CVSN for sample size = 100; (d) comparison of SNN and CVSN for sample size = 120.

was obtained using SNN, as compared to using the BAPN. At higher noise levels SNN produced errors greater than BAPN. However, crossvalidation performed very poorly for these cases. For 10% and 12.5% noise, the CVSN produced 130% and 67% greater error than BAPN. Poor performance of *k*-fold cross validation implies that the level-1 data does not provide a reliable indication of model accuracy. Inaccurate level-1 data also affects the SNN adversely. SNN produced 36% and 18% greater error than BAPN. Nonetheless, SNN was significantly better than the CVSN: this shows that even when level-1 data are inaccurate, SNN performed better than the CVSN. When level-1 data are unreliable, cross validation is inaccurate and it is difficult to decide which model is appropriate. Under these conditions it seems unreasonable to simply select one model and reject the others. SNN weights the other models instead of simply using the wrong model. Averaging, as accomplished by stacking, seems to be beneficial under these circumstances. The results here demonstrate another important point: performance of SNN and CVSN will be correlated. Correlation between the two methods is to be expected since SNN is using the same level-1 data as CVSN, although in a different manner. The performance of both techniques depends on the accuracy of the level-1 data.

Figure 6b shows the results for a sample size of 70. For this sample size, SNN outperformed the BAPN, for all noise levels. Clearly, no single model is optimal, and model integration accomplished by SNN finds a better model. Again, the correlation between CVSN and SNN is noticeable. CVSN performs very well in all cases shown. At the most, it produces 13% greater error than BAPN, indicating that the level-1 data are accurate. SNNs make better use of the level-1 data to construct a stacked model that produced average errors that were 10% to 26% lesser than BAPN. Accurate level-1 data allowed the SNNs to successfully integrate the independent information provided by the various candidate models. Hence it appears that SNNs can outperform the BAPN when a number of competing models are good predictors, there is low correlation between the errors made by the models, and the level-1 data are accurate. Figure 6c shows the results for sample size of 100. Cross validation performs poorly at the low noise levels. It produces errors of up to 40% greater than BAPN. Maximum error for the stacked model is 15% greater than BAPN. Figure 6d shows the results for sample size of 120. Cross validation produces models with up to 34% greater error than BAPN. SNNS, on the other hand, perform as well as or better than BAPN, for all the cases. Figure 6c and 6d also show SNN and CVSN performance similar to Figure 6a and 6b. Correlation between SNN and CVSN performance is clearly evident. As discussed earlier, such a correlation is to be expected. The interesting point is that the combination determined by the level-1 model consistently outperforms the single model selected by *k*-fold cross validation.

To examine the robustness of SNNs with respect to varying noise levels, we plot the percent reduction in prediction error using stacking for different noise levels at several sample sizes (Figure 7). Stacking performance does not appear to be sensitive to noise since substantial improvements over CVSN could be obtained at all the noise levels used in this study. It can also be seen that the maximum improvement using SNNs was produced at the smallest sample size considered in this example (sample size of 50). This can be expected because

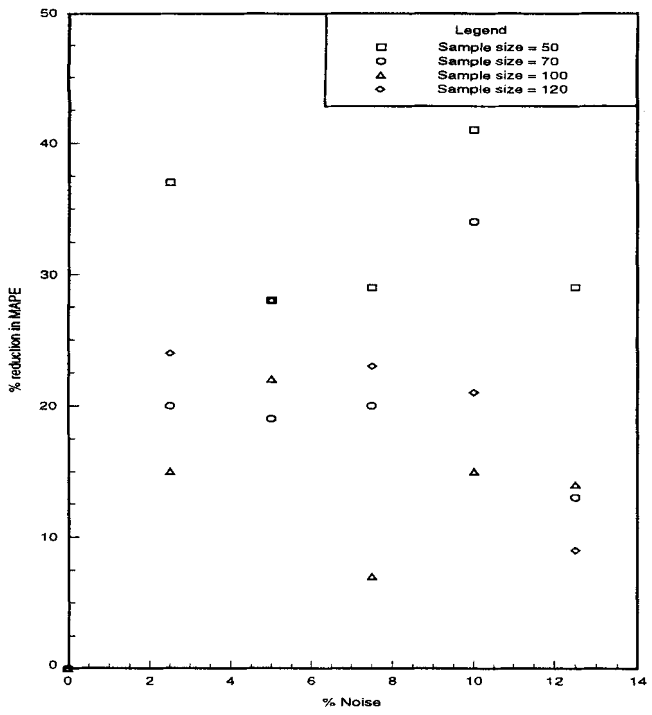


Figure 7. Reduction in MAPE using SNN vs. using CVSN for different noise levels.

variation between the models tends to be larger and errors made by different models showed lesser correlation for small data sets. The linear correlation between the errors made by the models ranged from 0.2 to 0.6 for a sample size of 50. For the larger sample sizes, the corresponding correlation coefficient ranged from 0.4 to 0.8. Hence combining was most beneficial at the small sample size. Smaller data sets also cause greater uncertainty regarding model selection, hence stacking can be a safer strategy as opposed to selecting a single model.

The preceding results suggest that the SNNs can consistently outperform the CVSN irrespective of the accuracy of the level-1 data. It appears that the SNNs make better use of the level-1 data, as compared to k-fold cross validation. Of course, situations can arise when cross validation can be better than the stacked modeling approach. For example, a poor level-1 generalizer can undermine the benefits of stacking. This has not been encountered by us, indicating the efficacy of the level-1 model used in this work. Since the level-1 model used in this example is linear, models with higher linear correlation to the actual output tend to be more heavily weighted. It is, however, possible that models whose outputs have low linear correlation could be important in a nonlinear sense. Such models cannot be effectively integrated by the linear level-1 model. It is important to identify and develop level-1 models that can develop nonlinear combinations of the candidate models. Identification of such general nonlinear level-1 models should further enhance the advantages of stacked modeling, and deserves further investigation.

Conclusion

In this work, a novel approach to empirical neural-network-based modeling of chemical processes has been proposed. Conventional approaches identify and use a single model for prediction. We have presented a novel architecture, stacked neural networks (SNNs), that effectively integrates the knowledge acquired by different networks to obtain a better predictive model. A technique for stacking neural networks has been developed and implemented. A linear model was used to stack the candidate networks. However, any nonlinear model can also be used. Stacked modeling was illustrated using a simple example and the performance of SNN was studied for two examples including the dynamic modeling of a chemical process. The examples demonstrate the feasibility of using stacked neural networks for improved modeling of chemical processes.

The main results of this work are summarized below:

1. Stacking consistently outperformed cross validation for all the cases studied. The SNN approach described in this work appears to be a promising technique for empirical ANN-based plant-process modeling. The only disadvantage of this method appears to be the increased computation time associated with developing the combination.

2. The linear level-1 model used in this work was seen to be an effective method for combining the level-0 neural networks.

3. For the dynamic modeling example, SNN performance did not degrade with increasing noise. SNN appears to be beneficial over a wide range of noise levels. Furthermore, the largest improvement using the SNN was obtained at the smallest sample size considered in the example.

The concept of stacked modeling can be easily extended to models other than linear combinations of backpropagation neural networks. Other empirical modeling methods, like radial basis function modeling, polynomial regression, and partial least squares, can also be combined using the approach described in this work. A diversity of models used as the level-0 models should further enhance the performance of the present technique, since the errors produced by a diversity of models are less likely to be correlated. It is important to determine what other models can be combined with neural networks. Achieving optimal performance would also require effective combining rules. Identification of such rules is an important issue for further research, and should be examined.

Acknowledgments

This work was made possible by the generous support of the Minnesota Mining and Manufacturing Company. Their support is greatly appreciated; however, it does not constitute an endorsement of the views expressed in this article.

Literature Cited

Basu, A., and E. B. Bartlett, "Detecting Faults in a Nuclear Power Plant by Using Dynamic Node Architecture Artificial Neural Networks," *Nucl. Sci. Eng.*, **116**, 313 (1994).

Bates, J. M., and C. W. J. Granger, "The Combination of Forecasts," *Oper. Res. Q.*, **20**(4), 451 (1969).

Bhat, N., and T. J. McAvoy, "Use of Neural Nets for Dynamic Modeling and Control of Chemical Process System," *Comput. Chem. Eng.*, **14**(4/5), 573 (1990).

Brieman, L., "Stacked Regressions," Tech. Rep. No. 367, Statistics Dept., Univ. of California at Berkeley (1992).

Cherkassky, V., Gehring, D., and Mulier, F., "Pragmatic Comparison of Statistical and Neural Network Methods for Function Estimation," *Proc. World Cong. on Neural Networks Conf.*, Vol. 2, Washington, DC, p. 917 (1995).

Clemen, R. T., "Combining Forecasts: A Review and Annotated Bibliography," *Int. J. Forecasting*, 5(4), 559 (1989).

Cybenko, G., "Approximation by Superpositions of a Sigmoidal Function," *Math. Contr. Signals Syst.*, **2**, 303 (1989).

De Veaux, R. D., D. C. Psichogios, and L. H. Ungar, "A Comparison of Two Nonparametric Estimation Schemes: MARS and Neural Networks," *Comput. Chem. Eng.*, **17**, 819 (1993).

Diebold, F. X., "Forecast Combining and Encompassing: Reconciling Two Divergent Literatures," *Int. J. Forecasting*, **5**, 589 (1989).

DiMassimo, C., G. Montague, M. J. Willis, M. T. Tham, and A. J. Morris, "Towards Improved Pencillin Fermentation via Artificial Neural Networks," *Comput. Chem. Eng.*, **16**, 382 (1992).

Economou, C. G., M. Morari, and B. O. Palsson, "Internal Model Control. 5. Extension to Nonlinear Systems," *Ind. Eng. Chem. Process Des. Dev.*, **25**, 403 (1986).

Fahlman, S. E., and C. Lebiere, "The Cascade-Correlation Learning Architecture," *Adv. Neural Inf. Process. Syst.*, D. S. Touretzky, ed., **2**, 524 (1990).

Finnoff, W., H. Hergert, and H. G. Zimmerman, "Improving Model Selection by Nonconvergent Methods," *Neural Networks*, **6**, 771 (1993).

Friedman, J. H., "Multivariate Adaptive Regression Splines," *Ann. Statist.*, **19**, 1 (1991).

Geman, S., E. Bienenstock, and R. Doursat, "The Bias-Variance Dilemma," *Neural Comput.*, **4**, 158 (1992).

Granger, C. W. J., and R. Ramanathan, "Improved Methods of Forecasting," *J. Forecasting*, **3**, 197 (1984).

Granger, C. W. J., "Combining Forecasts: Twenty Years Later," *J. Forecasting*, **8**, 167 (1989).

Hansen, L. K., and P. Salamon, "Neural Network Ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, **12**(10), 993 (1990).

Hashem, S., "Optimal Linear Combinations of Neural Networks," PhD Thesis, School of Industrial Engineering, Purdue Univ., West Lafayette, IN (1993).

Hornik, K., M. Stichcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Network*, **2**, 359 (1989).

Hoskins, J. C., and D. M. Himmelblau, "Artificial Neural Network Models of Knowledge Representation in Chemical Engineering," *Comput. Chem. Eng.*, **12**, 881 (1988).

Joseph, B., F. H. Wang, and D. S. Shieh, "Exploratory Data Analysis: A Comparison of Statistical Methods with Artificial Neural Networks," *Comput. Chem. Eng.*, **16**, 413 (1992).

Kramer, M. A., "Autoassociative Neural Networks," *Comput. Chem. Eng.*, **16**, 313 (1992).

Leonard, J. A., M. A. Kramer, and L. H. Ungar, "A Neural Network Architecture that Computes its Own Reliability," *Comput. Chem. Eng.*, **16**, 819 (1992).

MacGregor, J. F., T. E. Marlin, and J. V. Kresta, "Some Comments on Neural Networks and Other Empirical Modelling Methods," *Proc. CPC-IV Int. Conf. on Chemical Process Control*, Padre Island, TX, p. 665 (1991).

Mah, R. S. H., and V. Chakravarty, "Pattern Recognition Using Neural Networks," *Comput. Chem. Eng.*, **16**(4), 371 (1992).

Makridakis, S., C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord, and L. F. Simmons, "The M2 Competition: A Real Time Judgementally Based Forecasting Study," *Int. J. Forecasting*, **9**, 5 (1983).

McAvoy, T. J., N. S. Wang, and N. Bhat, "Interpreting Biosensor Data via Backpropagation," 1989 American Control Conference, Pittsburgh (1989).

Moller, M. F., "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning," *Neural Networks*, **6**(4), 525 (1993).

Perrone, M. P., and L. N. Cooper, "When Networks Disagree: Ensemble Methods for Hybrid Neural Networks," *Neural Networks for Speech and Image Processing*, R. J. Mammone, ed., Chapman & Hall, London (1993).

Piovoso, M. J., and A. J. Owens, "Sensor Data Analysis Using Neural Networks," *Proc. CPC-IV Int. Conf. on Chemical Process Control*, Padre Island, TX, p. 101 (1991).

Pollard, J. F., M. R. Broussard, and D. B. Garrison, "Process Identification Using Neural Networks," *Comput. Chem. Eng.*, **16**, 253 (1992).

Psichogios, D. C., and L. H. Ungar, "Direct and Indirect Model Based Control Using Artificial Neural Networks," *Ind. Eng. Chem. Res.*, **30**, 2564 (1991).

Sridhar, D. V., R. C. Seagrave, and E. B. Bartlett, "Improving Neural Network Based Predictive Modeling Using Stacked Generalization," *Intelligent Engineering Systems through Artificial Neural Networks*, Vol. 5, C. H. Dagli, M. Akay, C. L. P. Chen, and B. Fernandez, eds., American Society of Mechanical Engineers, St. Louis, MO (1995).

Venkatasubramanian, V., and K. Chan, "A Neural Network Methodology for Process Fault Diagnosis," *AIChE J.*, **35**, 1993 (1989).

Weiss, S. M., and C. A. Kukilowski, *Computer Systems that Learn*, Morgan Kaufman, San Mateo, CA (1991).

White, H., "Some Asymptotic Results for Learning in Single Hidden Layer Feedforward Network Models," *J. Amer. Stat. Assoc.*, **84**, 1008 (1989).

Winkler, R. L., "Combining Forecasts: A Philosophical Basis and Some Current Issues," *Int. J. Forecasting*, **5**, 605 (1989).

Wolpert, D. H., "Stacked Generalization," *Neural Networks*, **5**(2), 241 (1992).

Ydstie, B. E., "Forecasting and Control Using Adaptive Connectionist Networks," *Comput. Chem. Eng.*, **14**, 583 (1990).

Manuscript received July 3, 1995, and revision received Nov. 13, 1995.