Key Learnings from Twenty Years of Neural Network Applications in the Chemical Industry

Aaron J. Owens, PhD
DuPont Company, Engineering Research and Technology,
POB 80356, Wilmington, DE, 19880-0356, U.S.A.
aaron.j.owens@usa.dupont.com

Abstract. This talk summarizes several points that have been learned about applying Artificial Neural Networks in the chemical industry. Artificial Neural Networks are one of the major tools of *Empirical Process Modeling*, but not the only one. To properly assess the appropriate model complexity, combine information about both the *Training* and the *Test* data sets. A neural network, or any other empirical model, is better at making predictions than the comparison between modeled and observed data shows. Finally, it is important to exploit synergies with other disciplines and practitioners to stimulate the use of Neural Networks in industry.

Keywords: Artificial Neural Networks, Neural Network Applications, Stopping Criteria, Empirical Model Validity

1 Introduction

DuPont has had a long history of supporting neural network applications in the chemical industry. Initial efforts were aimed at commercial applications in digital image processing during the 1980s. In 1990, the Neural Network Resource Center was established to catalyze the use of artificial neural network models, particularly the Back Propagation Network [1], [2], internally throughout the Company. The author has personally been involved in this these efforts ever since, except for a brief, misguided detour into management. This talk summarizes some of the "key learnings" about neural network applications resulting from those efforts.

2 Neural Networks as an Empirical Process Modeling Tool

Empirical Process Modeling is data driven, rather than emphasizing fundamental equations. It can be described colloquially as *Modeling a Spreadsheet*. The spreadsheet metaphor has been useful in helping our internal practitioners to decide what type of model to use. The *Process* being modeled can be of any kind: chemical, physical, biological, business, financial, marketing, etc. DuPont is a large, international Science company. Our *Empirical Process Modeling* efforts have spanned many products, functions, and regions.

DuPont's proprietary neural network software, called the *Neural WebKit*, uses a simple spreadsheet-based data format. Each *Exemplar* or case appears on a row. It consists of the following columns: a unique integer *Exemplar Number* followed by the numerical values of all of the process *Inputs X* and then of all the process *Outputs Y*. See Figure 1.

The empirical modeling tool, here illustrated by the Back Propagation Network (BPN), is applied to the data set to obtain the parameters for an Empirical Model. The goal of Empirical Modeling is to maximize accuracy of the prediction of the Ys from the Xs.

The software requires that the data set be pre-processed so that there are no missing entries and that all variables are real-valued. Categorical variables need to be unary encoded, with the Machines A, B, or C coded as the three variables [1 0 0], [0 1 0] and [0 0 1].

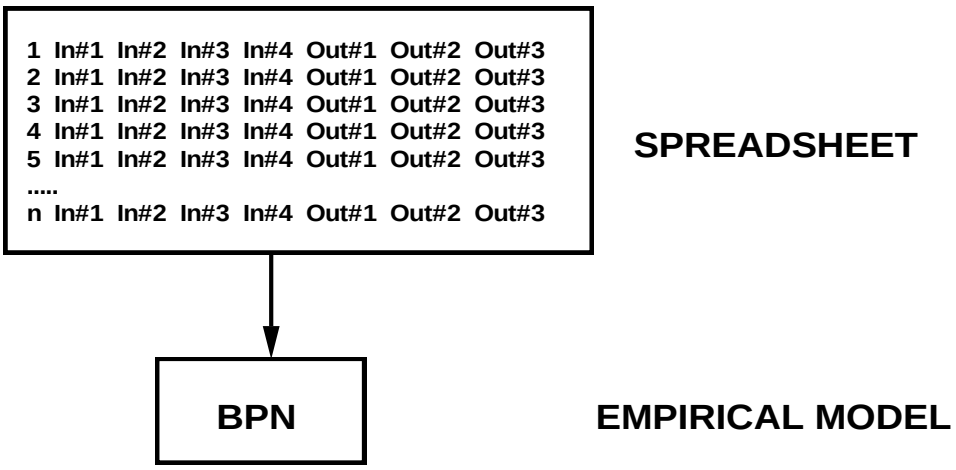


Fig. 1. The Spreadsheet Metaphor. In *Empirical Modeling,* data are arranged into a Spreadsheet with real-valued *Inputs* and *Outputs* from the process being modeled.

The appropriate modeling technique to use in building the empirical model is determined by the *shape* of the spreadsheet, as illustrated in Figure 2, along with the data's *Signal to Noise* ratio. For *Short and Fat* spreadsheets, there are fewer (often many fewer) exemplars than variables, so a sub-linear method is necessary. To identify the model coefficients, the number of model parameters needs to be less than the number of observations. Even simple *Multiple Linear Regression* (MLR) cannot be used for these cases.

For *Short and Fat* spreadsheets, the recommended *Chemometric* technique is *Partial Least Squares (PLS)* [3]. Classical statistical techniques (like MLR and polynomial regression) deal with cases in which the spreadsheet matrix is close to *Square*. In fact, classical statistical *Design of Experiments* (DOE) minimizes the number of experiments so that it is just a few more than the number of parameters to be estimated, so the spreadsheet is nearly *Square*.

Neural networks are more appropriate only in the relatively data-rich case of a *Tall and Skinny* data matrix. Then there are enough observations to allow the system to identify a

model with more coefficients than *Inputs* [4], [5].

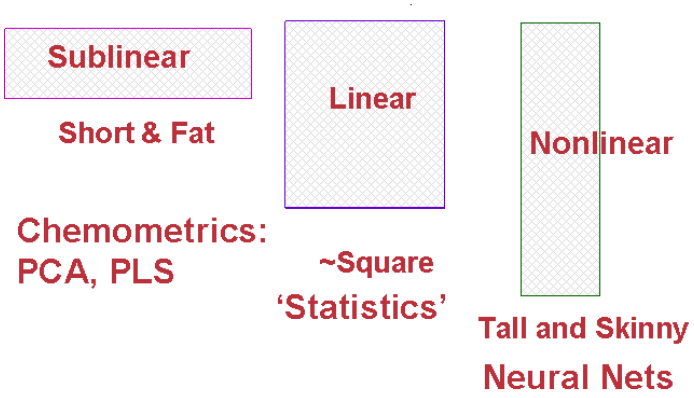


Fig. 2. What *shape* is your spreadsheet? *Chemometrics* techniques like *Partial Least Squares* [PLS] are best suited to *Short and Fat* matrices. Classical statistical techniques are appropriate with *Square* matrices. Neural network can be applied to *Tall and Skinny* matrices. For low *Signal to Noise* data sets, more *Exemplars* are required to accurately determine model parameters.

There are a couple of *caveats* with respect to selecting the appropriate model. First, while a matrix may appear *Tall and Skinny*, if the underlying process has a small *Signal to Noise* ratio (i.e., it is dominated by noise), then more -- perhaps many more -- observations are required to accurately identify the model parameters. So it may be appropriate to use PLS for a *Tall and Skinny* matrix with low *Signal to Noise*. Second, both PLS and neural networks can successfully model linear systems. So in *Empirical Modeling* the classical statistical tools are actually seldom needed, as long as the objective of the model is accurate *Prediction*. While PLS and neural networks have tools to assess variable importance, only classical statistical techniques allow you -- for example -- to point to a specific coefficient as the cross term between variables x(3) and x(4), as well as give its statistical uncertainty.

Most of the time needed to prepare an Empirical Process Model comes from collecting and checking the data. Once the data are in a *Spreadsheet*, the time to do an analysis is usually trivial, especially with PLS. So it is recommended to *always* run a PLS analysis before proceeding to use a neural network on the same data set. Note that you can transform a single hidden layer Back Propagation Network to the PLS structure simply by eliminating the squashing functions. In addition, commercial PLS software has good graphical tools for interpreting the models with multiple-input, multiple-output mappings.

3 Use Both Training and Test Data to Find Model Complexity

More than 90% of the neural network applications at DuPont have used the Back Propagation Network with a single hidden layer [6]. That methodology has remained basically unchanged for more than twenty years. There are some special circumstances in which more hidden layers are used, often a *Bottleneck* layer [7], but they are limited. No matter how many hidden layers are used, there is usually one called the *Mapping Layer*, which controls the complexity of the input-output mapping. A small number of nodes in the Mapping layer corresponds to low model complexity, and the representational ability of the neural network grows as that number is increased.

Because training a neural network can be time-consuming, it is usual to use a single *Train/Test* data split, rather than more sophisticated multiple cross-validation methods from the Chemometrics literature. For neural networks, the data are split, either randomly or by design, into a Training data set and a Test data set. Models are developed ('trained') on the *Training* data, and their expected performance for generalization is estimated from the *Test* data. If enough data are available, a separate Validation data set is sometimes used as a final check on the accuracy of the system.

3.1 Three Train/Test Methods: Scattergun, Train & Grow, and Early Stopping

There are at least three competing methods for developing neural network models of the appropriate complexity. In the *Scattergun* method, neural networks with 1 through M hidden nodes in the *Mapping* layer are randomly initialized. They are all trained to completion, and the choice is the one with the minimum *Test* error. Because it uses multiple random initializations, this method is not robust.

A method with wide application in DuPont is *Train and Grow* [6]. Start with one node in the Mapping Layer, and train the model on the Training data to completion. Note the performance on the Test data. Grow the trained network by adding a second mapping node, with small random values for the new connection weights, and then train it. Continue training and growing until the Test error increases with increasing nodes. This method gives a more robust and reproducible Train/Test path, because the models usually stay in the same basin of attraction in weight space.

Early Stopping is a viable alternative to *Train and Grow*, particularly if modeling similar systems has given you a reasonable starting point. In this method, a single, large number of mapping units is selected, more than are actually needed. The weights are randomly initialized and then training begins. Rather than training to completion, the Root Mean Square (RMS) Errors of the *Train* and *Test* data sets are monitored as a function of the number of training iterations. Select the *Number of Training Iterations* giving the minimum *RMS Test* error as the appropriate model complexity [8].

Note that with Early Stopping, training is stopped before the weights reach their final optimum values. This method runs much more quickly than Train and Grow, since only one model is developed. But *Early Stopping* is more sensitive to initial conditions, particularly for multiple-hidden-layer models.

For all of these methods, the important piece of information is the minimum RMS *Test* error, since that should predict performance on new data. As a final step, it is always recommended to re-train the model using *All* of the data and stopping when the appropriate *RMS Test* error limit is reached.

Historically at DuPont, the *Shotgun* method was first tried but quickly abandoned as too unreliable. *Train and Grow* continues to be used for most applications with a single hidden layer [6]. For multiple-hidden-layer *Bottleneck Neural Networks, Early Stopping* has been adopted for decreased training time [8].

3.2 A Complexity Criterion Using Both Train and Test Errors

To select the best model, the neural network community has heavily weighted one criterion – minimum *RMS Test* error – but ignored some additional relevant information. A good model would be one that has a clear minimum in the *RMS Test* error but *also* has *Train* and *Test* errors similar to each other. In other words, we should also consider the *RMS Training* error in selecting the appropriate model complexity. If it is significantly less than (or greater than) the *RMS Test* error at its minimum (as a function of *Number of*

Hidden Units for *Train and Grow* or *Number of Iterations* for *Early Stopping*), that indicates a modeling failure. The *Train* and *Test* data sets are, for some reason, not compatible.

We propose a new metric that takes into account both *RMS Train* and *Test* errors. It optimizes *both* the *Test* error and the difference between *Train* and *Test*. An equation that accomplished this is to minimize

$$\alpha*(\text{RMS Test error}) + (1-\alpha)*|\text{RMS Train error} - \text{RMS Test Error}| \; . \qquad (1)$$

If $\alpha = 1$, only the RMS Test error matters (as has been conventional). If $\alpha = 0$, the criterion assures that the *RMS Train* and *Test* errors are the same. After looking at dozens of cases, we propose $\alpha = 0.5$ as a good compromise. That is a 50/50 combination of the two parts of Equation (1).

The process is to enter the RMS Train and RMS Test errors, along with the corresponding *Number of Hidden Units* or *Iteration Number*, into a spreadsheet. The spreadsheet locates the minimum of the function of Equation (1) and records the appropriate *Model Complexity* at that point.

See the example in Figure 3 below. The appropriate model complexity (*Number of Iterations*) is selected by minimizing the RMS *Test* error (plus symbols) and the absolute difference between RMS *Train* error (filled circles) and RMS *Test* error. A table with the plotted values, for this hypothetical example, appears on the right, where *Train* and *Test* refer to scaled RMS errors and *Crit.* is the new criterion of Equation (1) with $\alpha = 0.5$. The solid curve shows this new criterion.

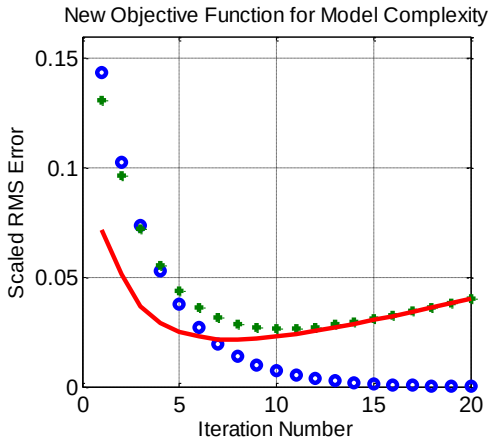
The usual procedure, considering only the minimum RMS Test error, gives an optimum at Iteration 10 (underlined), where the Train and Test errors are 0.0071 and 0.0264, respectively (differing by a factor of 3.7!). The New Criterion selects 8 Iterations (bold and underlined), where the train and Test errors are 0.0139 and 0.0285 (a smaller factor of 2.1).

Adopting this *New Criterion* should give a more balanced selection of the appropriate model complexity. Incidentally, it works well for PLS models too, where the *Train* and *Test* errors are replaced by *Calibration* and *Cross-Validation* errors, and the complexity parameter is the *Number of Latent Variables* selected.

An important final step is to re-train the neural network model using *All* of the data (*Training* + *Test* data sets), stopping at the appropriate *Model Complexity*: the selected optimum *Number of Hidden Units* for *Train and Grow* or the *Number of Iterations* for *Early Stopping*.

4 An Empirical Model is Better Than the Data Show

The discussion in this section may be well known to statisticians, but many people in the neural network and chemometrics communities may not be aware of it. Quite simply, any empirical model fit by least squares has its model coefficients and hence its prediction accuracy improved with additional data from the same process. However, by looking at model predictions compared with just the (noisy) data from that process itself, you cannot prove this.



Iters	Train	Test	Crit.
1	0.1433	0.1310	0.0717
2	0.1027	0.0964	0.0513
3	0.0736	0.0722	0.0368
4	0.0527	0.0554	0.0291
5	0.0378	0.0440	0.0251
6	0.0271	0.0364	0.0228
7	0.0194	0.0315	0.0218
8	**0.0139**	**0.0285**	**0.0216**
9	0.0100	0.0270	0.0220
10	0.0071	0.0264	0.0229
11	0.0051	0.0266	0.0240
12	0.0037	0.0273	0.0255
13	0.0026	0.0284	0.0270
14	0.0019	0.0297	0.0288
15	0.0013	0.0312	0.0305
16	0.0010	0.0329	0.0324
17	0.0007	0.0346	0.0343
18	0.0005	0.0364	0.0362
19	0.0004	0.0383	0.0381
20	0.0003	0.0402	0.0401

Fig. 3. New Objective Function for Selecting Model Complexity.

Suppose that there is a Process with multivariate Inputs X and Outputs Y. There will also be noise (ε) in the process. A model for this kind of process is

$$Y_k = f(X_k) + \varepsilon_k \, . \tag{2}$$

Here the subscript k refers to the kth *Exemplar*.

The function $f(X_k)$ is the *True Relationship* between *X* and *Y*, but each observation or *Exemplar* has an associated noise (ε). As is customary, the random errors ε for the exemplars are assumed to be *Independent and Identically Distributed* (IID), sampled from a *Normal* distribution with mean 0 and standard deviation σ.

In a neural network, the True Relationship is approximated by a neural network model $f^{NN}(X)$ with many parameters. After the training of neural network to get the optimum values of the model parameters, we assume that the neural network model approximates the truth, or:

$$f^{NN}(X) \sim f(X) \, . \tag{3}$$

Note that even if Equation (3) represents full equality, when we use $f^{NN}(X)$ to estimate the (noisy) observed value of *Y*, we see from Equation (2) that the difference between predicted and observed is equal to the noise (with standard deviation σ). While the model can get very accurate, as more and more data points are added, when we compare it with the noisy real data we still get a *Root Mean Square* error of σ.

4.1 Simple Linear Example: Numerical Simulations

The effect can be illustrated with a very simple example, with univariate x and y related by a linear relationship. If Equation (3) is applicable (i.e., the neural network, when trained,

approaches the true X-Y noise-free relationship), the same kind of results would be obtained with a Neural Network model rather than this linear model. The linear results are easier to explain.

Equation (2) with this underlying model becomes:

$$y_k = c^* x_k + \varepsilon_k \ . \qquad (4)$$

Equation (4) is used to generate the *Observed* data. For specificity, take c = 6 and $\sigma = 1$.

Numerical simulations were performed using a simple Matlab script. Each of N *Exemplars* has *x* values between uniformly sampled between 0 and 1, then added random noise with $\sigma = 1$ to get *Observed y* values from Equation (4).

To simulate the model-building phase, a simple linear regression model (Equation 5) was fitted to the N data pairs (*x,y*).

$$y = a^* x \ . \qquad (5)$$

The slope *a* and the RMS error between the *Observed* and *Predicted* values of *y* were recorded for each realization. Calculations were made with the number of observations N = [10 30 100 300 1000 3000 10000]. The simulation was repeated with each value of N for a total of 1000 iterations and averaged over all 1000 of them to reduce sampling noise.

See the results in Figure 4. The RMS error between the model and the (noisy) Observed data is shown with the symbol 'x'. The RMS error between model and the underlying True (noise-free) data is shown with the symbol 'o'. The RMS error between the true slope (c = 6) and the model-estimated slope (a) is shown with the symbol '*'. The line is drawn to guide the eye; its equation is 1/sqrt (N), where N is the Number of Samples.

Note that the RMS error between *Predicted* and *Observed* y is close to $\sigma = 1$, as discussed above. (It is labeled 'Observed' in the figure.) Increasing N does not change this estimate, which is roughly constant with N. With added data, the RMS error of the model prediction compared with the (noisy) *Observed* data does not decrease.

Next consider the value of the modeled slope, *a*. It is close to the true value of c = 6 for N = 10, but it gets increasingly closer, with error varying as 1/sqrt (N), as N is increased. While standard statistical packages would estimate the error limits on the slope, there is no direct way in most PLS or neural network calculations to extract that information. The modeler may not realize that the model is getting more accurate with additional data, especially if RMS error of model predictions versus the observed noisy data is the only evaluation factor.

Finally, since this is a *Simulation*, one can 'play God' and look at how the model predicted the *underlying actual relationship*, y = 6*x, ignoring the noise in Equation (4). The points labeled 'True' in Figure 4 shows that the model prediction accuracy, compared with the underlying noise-free generating model, does in fact get better as 1/sort(N). In this simulation, the RMS prediction error from the noisy observed data remains close to $\sigma = 1$, while the model's prediction of the true underlying relationship has an RMS error of about 0.01 with 10,000 samples.

So the empirical model predictions actually are much more accurate than a comparison between the predicted and observed data show. The law of large numbers applies here, so the model prediction error of Neural Networks and PLS models really does decrease as 1/sqrt(N), as in standard statistical model estimation.

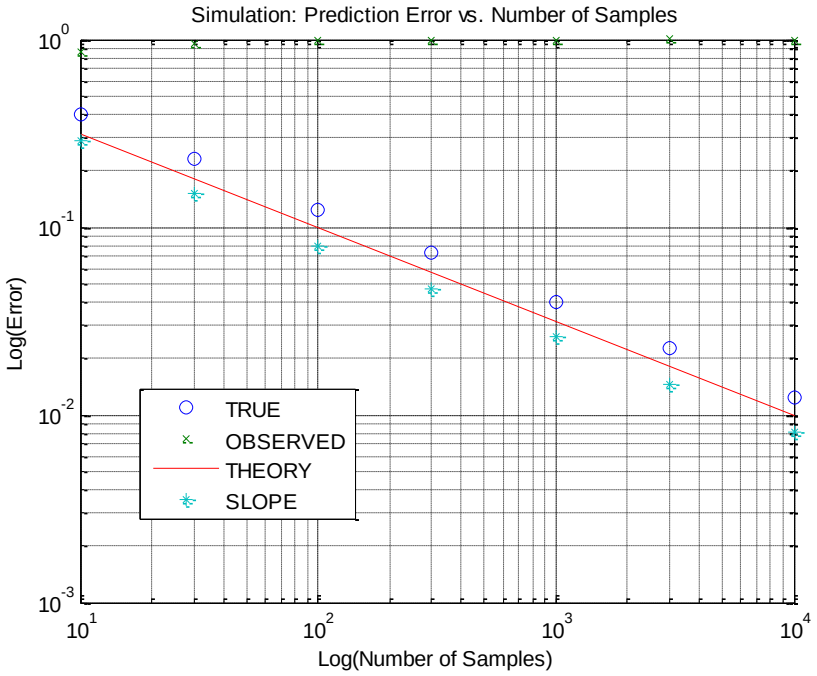


Fig. 4. Summary of Numerical Simulations of Model Accuracy, on a Log-Log Plot.

5 Synergies to Increase the Use of Neural Networks in Industry

As discussed in Section 2 above, the experience at DuPont has shown Neural Networks are best used as an Empirical Process Modeling tool, in conjunction with traditional statistical analysis and methods from Chemometrics. In light of this, the Neural Network practitioner in industry (as well as in research) should not be isolated from their colleagues who are applied statisticians and chemometricians.

Whatever success achieved in introducing Neural Networks at DuPont [9] was catalyzed by neural net modelers interacting with these other groups. As early as 1990, when few statisticians would deign to consider Neural Networks, a close working relationship was established with the Applied Statistics group. Industrial statisticians are very appropriate Neural Network practitioners, because of their understanding of nonlinear function fitting and their appreciation for the effects of uncertainty. The neural network and statistics groups had cross-training sessions about Neural Networks, Exploratory Data Analysis, and Design of Experiments. The internally developed Neural Network software was enhanced by adding Analysis of Variance tables, incorporating Degrees of Freedom thinking into the r^2 calculations, and surface and contour plots of results. Our group presented one of the first papers on Neural Networks (for process modeling and control) at an American Statistical Association in 1992 [10]. Two statisticians reported on different kinds of cross-validation for Neural Networks at the same conference the following year [11].

About the same time, Mike Piovoso introduced us to chemometrics and began our interactions with the process control community. We learned how to interpret the Hidden Layer outputs using Principal Component Analysis and recognized the similarity between

the structure of a Back Propagation Network (with one hidden layer) and Partial Least Squares (PLS). Effectively, PLS is a BPN *without the squashing function*. PLS software packages showed us the utility of the Modeled versus Observed plot (Goodness of Fit), and the graphical presentation in PLS's Variable Importance Plot guided our Sensitivity Analysis plot. Interactions with process control engineers prompted the early adoption of Neural Networks as *Virtual Sensors* for model-based control [12]. Neural Networks were subsequently integrated by the major process control vendors into their software.

More importantly than just improving our Neural Network methodology, the contacts with the statisticians and process control engineers gave a wide variety of novel and important applications.

6 Summary

The primary "value add" of Neural Networks is that they are extremely good at extracting complex, nonlinear interactions between process input and outputs, particularly when there are enough observations to fit a model with high signal to noise. There are many and varied applications for that kind of tool in industry [13], [14], [15], [16].

Acknowledgments. DuPont colleagues who have contributed significantly to our neural network technology and applications include Tom Lynch, Alicia Walsh, Dewey Donovan, John van Stekelenborg, Vivek Bhide, and Tony Liu. Supporters in the businesses include Dave Alman, Allan Rodrigues, Mahnaz Mohammadi, Tariq Andrea, John Kinney, Arun Aneja, and John Locke. I learned about Chemometrics from Mike Piovoso, Barry Wise (Eigenvector), and John MacGregor (Prosensus).

References

1. Rumelhart, D. E., G. E. Hinton, and R. J. Williams: Learning Internal Representations by Error Propagation. In: Rumelhart and McClelland, eds, Parallel Distributed Processing, Vol. 1, Foundations. MIT Press, Cambridge, 318--335 (1986)
2. Hertz, J., A. Krough and R. G. Palmer: Introduction to the Theory of Neural Computation. Addison-Wesley, New York (1991)
3. Software User Guide, PLS Toolbox for Matlab: Eigenvector Research, Inc., Wenatchee, Washington, http://wiki.eigenvector.com (2011)
4. Ljung, L. and J. Sjoberg: A System Identification Perspective on Neural Nets. In: Kung, et, al., eds, Neural Networks for Signal Processing II. IEEE, Piscataway, NJ, 423--435 (1992)
5. Bishop, C. M.: Neural Networks for Pattern Recognition. Clarendon Press, Oxford (1995)
6. Piovoso, M. J. and A. J. Owens, Sensor Data Analysis Using Artificial Neural Networks. In: Arkun and Ray, eds., Chemical Process Control CPC IV. AIChE, New York, 101--118 (1991)
7. Owens, A. J.: Using Fundamental Knowledge to Guide Empirical Model Structure. In: International Conference on Engineering Applications of Neural Networks (EANN'01). Cagliari, (2001)
8. Owens, A. J.: Neural Network Based Product Formulation Models: Early Stopping Saves Training Time. In: International Conference on Engineering Applications of Neural Networks (EANN'07). Stockholm (2007)
9. Owens, A. J.: Neural Network Based Empirical Modeling in the Chemical Industry. In: International Conference on Engineering Applications of Neural Networks (EANN'98). Gibraltar (1998)

10. Owens, A. J.: Artificial Neural Networks for Process Modeling and Control. In: Proceedings of the Annual Joint Statistical Meetings of the American Statistical Association. Boston (1992).
11. Bhide, V., and A. Banerjee: Bootstrap Estimates in Neural Network Models. In: Proceedings of the Annual Joint Statistical Meetings of the American Statistical Association. San Francisco (1993).
12. Schnelle, P. D., Jr., and J. A. Fletcher: Using Neural Network Based Process Modeling in Process Control. In: Proceedings of the I.S.A./90 International Conference and Exhibition. (1990)
13. Hu, Yu Hen and Jeng-Neng Hwang, eds: Handbook of Neural Network Signal Processing. CRC Press, Boca Raton (2001)
14. Badiru, Adedeji B. and John Y. Cheung: Fuzzy Engineering Expert Systems with Neural Network Applications. Wiley-Interscience, New York (2002)
15. MacKay, David J. C.: Information Theory, Inference and Learning Algorithms. Cambridge University Press, Cambridge (2003)
16. Suzuki, Kendi, Ed.: Artificial Neural Networks - Methodological Advances and Biomedical Applications. InTech, Rijeka, Croatia (2011)