UNIVERSIDADE ESTADUAL DE CAMPINAS
SISTEMA DE BIBLIOTECAS DA UNICAMP
REPOSITÓRIO DA PRODUÇÃO CIENTIFICA E INTELECTUAL DA UNICAMP

Versão do arquivo anexado / Version of attached file:

Versão do Editor / Published Version

Mais informações no site da editora / Further information on publisher's website:
http://www.davidpublisher.org/index.php/Home/Article/index?id=7739

DOI: 10.17265/2159-5275/2015.03.003

Direitos autorais / Publisher's copyright statement:
©2015 by David Publishing. All rights reserved.

DIRETORIA DE TRATAMENTO DA INFORMAÇÃO

Cidade Universitária Zeferino Vaz Barão Geraldo
CEP 13083-970 – Campinas SP
Fone: (19) 3521-6493
http://www.repositorio.unicamp.br

Journal of Mechanics Engineering and Automation 5 (2015) 153-160
doi: 10.17265/2159-5275/2015.03.003

Neural Networks and the Study of Time Series: An Application in Engineering Education

José Tarcísio Franco de Camargo[1], Estéfano Vizconde Veraszto[2], Gilmar Barreto[3] and Sérgio Ferreira do Amaral[4]

1. Department of Computer Engineering, Regional Universitary Center of E. S. do Pinhal, E. S. do Pinhal 13990-000, Brazil

2. Department of Natural Sciences, Mathematics and Education, Federal University of Sao Carlos, Araras 13604-900, Brazil

3. School of Electrical and Computing Engineering, State University of Campinas, Campinas 13083-852, Brazil

4. School of Education, University of Campinas, Campinas 13084-865, Brazil

Received: February 16, 2015 / Accepted: March 04, 2015 / Published: March 25, 2015.

Abstract: Time series are an important object of study in sciences, engineering and business, especially in cases where it is expected to know, predict and optimize behaviors. In this context, we intend to show the feasibility of using artificial neural networks in the study of several time series in an engineering course, especially those that have no overt behavior or are not able to be modeled mathematically in a simple way and have direct application in the education of future engineers.

Key words: Engineering education, time series, mathematical modeling.

1. Introduction

Knowing, predicting and optimizing behaviors are quite common objectives in the study of time series. Being able to analyze data sets, understand its behavior and approximate predict future values can become a crucial factor for a project success. Thus, the study of series with no easy modeling behaviors, such as those from data mining, process optimization or demands prediction is, therefore, an objective to be carefully treated in engineering courses. Future engineers must, therefore, be able to analyze and "extract knowledge" from non-trivial time series modeling, being able to, as far as possible, predict future situations. In this context, this paper presents a methodology, the use of neural networks for the study of time series, aiming to contribute to this objective.

According to our proposal, this paper is organized as follows: Section 2 presents the perspective of the study of time series in engineering education, introducing the use of ANNs (artificial neural networks) in non-trivial time series analysis; Section 3 presents the basic concepts on modeling and simulating ANN; Section 4 discusses the application of ANN in time series analysis; finally, Section 5 presents our conclusions about the use of ANN in time series analysis and its use in engineering education.

2. Time Series in Engineering Education

The study of time series can be achieved more frequently through the use of well-defined equations, when the series has a well-defined behavior, or through approximation or interpolation equations, when there is a behavior that can be identified, but not modeled with accuracy.

For example, Table 1 shows a time series that can be completely defined by the equation:

$$x_{k+1} = x_k + x_{k-1} \quad (1)$$

where, $x_0 = x_1 = 1$.

Table 2 shows a time series representing the Brazilian population evolution from 1900 to 1990; it

Corresponding author: José Tarcísio Franco de Camargo, Ph.D., research fields: computer graphics, automation and education. E-mail: jtfc@bol.com.br.

Table 1 Fibonacci series.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	X_9	...
1	1	2	3	5	8	13	21	34	55	...

Table 2 Brazilian population during the twentieth century [1].

Year (x)	Population in millions (y)
1900	17.4
1910	23.4
1920	30.6
1930	35.8
1940	41.2
1950	51.9
1960	70.9
1970	94.5
1980	121.1
1990	146.9

can be approximated by an exponential function of the type:

$$y = a_0 \cdot e^{a_1 \cdot x} \tag{2}$$

where, a_0 and a_1 are constants that can be determined through the solution of the system:

$$[X]^T \bullet [X] \bullet [A] = [X]^T \bullet [Y] \tag{3}$$

While the above cases have a relatively predictable behavior, we should consider a problem to be outlined: a time series does not always have a well-defined behavior which can be accurately described by an equation. Also, they do not always have a behavior that allows an approximation or interpolation using a given equation. In this context we intent to present the use of neural networks in the study of time series: they can detect and learn behaviors that are not mathematically modeled in a trivial way.

As presented in Ref. [2], we can state that ANN is able to identify behaviors from prior learning of certain patterns. Thus, the series behaviors we presented before may be learned by a particular ANN, making it possible to infer values for this series, within certain limits. Furthermore, implicit behaviors, which could not be observed in a direct or trivial way, may be detected by an ANN. For example, the study of a series of prices of a particular share traded on a stock exchange can be accomplished through an ANN. In this way, Table 3 shows a time series with the AMA (arithmetic moving averages) of five days of EMBR3 share closing price (EMBRAER-ON) traded on the Sao Paulo Stock Exchange (BOVESPA).

The MLP (multilayer perceptron) is an ANN model very interesting for the study of time series. Through this model, implicit patterns in a time series are likely to be detected, allowing the network to identify certain behaviors. Thus, the time series analysis we proposed here is focused on the use of the MLP-ANNs, as an alternative to conventional methods for the series study.

3. Neural Networks and Engineering Education

An ANN is a system commonly abstract, which can be used in various applications such as recognition and classification of patterns, modeling of physical and/or mathematical problems, processes control and signal processing as well as several other applications such as shown in Refs. [3-8]. Therefore, before its use in these tasks, an ANN must be “trained” through behavior patterns examples. Thus, in its training phase, the network is presented to different input patterns and their desired output patterns, in order to “learn” the “formation law”, correlating each input pattern to its respective output pattern.

After this phase, if an input pattern that is not part of the training set is presented to the network, it is expected that the ANN could infer a likely output pattern. As an example, let us consider that we want to check a possible formation law for the series described in Table 3.

Table 3 Arithmetic moving average of EMBR3 share closing prices.

Date	Price (in Brazilian Reais)
14/03/13	17.22
15/03/13	17.25
18/03/13	17.27
19/03/13	17.28
20/03/13	17.28
21/03/13	17.33
22/03/13	17.40

Table 4 Training set for an ANN.

Input vector	Desired output
(x_1; x_2; x_3; x_4)	(y)
17.22; 17.25; 17.27; 17.28	17.28
17.25; 17.27 ; 17.28; 17.28	17.33
17.27; 17.28; 17.28; 17.33	17.40

In this case, during its training phase, we could give to the network the following input and output patterns, so it could understand the formation of this law series.

So after the training phase, if the following input pattern is presented to the network (17.27; 17.28; 17.28; 17.33), it is expected that it presents in its output an estimate for a moving average of 22/03/13 (i.e., 17.40, according to Tables 3 and 4).

An ANN model that fits within this proposal is the MLP proposed in Ref. [9], which is discussed below.

3.1 The Perceptron

A MLP-ANN is a layered organization of a computational element called "perceptron". A perceptron, in turn, is characterized by trying to express a symbolic (and mathematical) representation of a biological neuron. Fig. 1 shows a symbolic model of a perceptron. Its model is widely discussed in Refs. [2, 9, 10] and briefly presented below. Refs. [9-12] also present in details the "ackpropagation algorithm" which will be discussed later.

In this model there is a "node", representing the neuron's body, powered by various external stimuli (x) through "synaptic connections". The computation result of the external stimuli processed by the perceptron is exported via an output synaptic connection (y), which will transmit this signal to the input of other network perceptrons.

The external stimuli that a perceptron receives can be obtained from the output exported by other perceptrons or from the ANN input, or from a particular "polarization signal" (bias) of the neuron. According to Ref. [13], the polarization signal has great importance in controlling the noise of the data presented to the network. All signals that arrive at a neuron are considered by the connection "synaptic

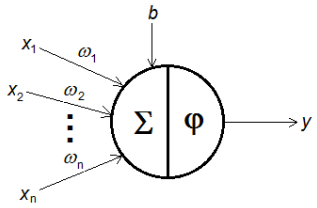

Fig. 1 Symbolic model of a perceptron.

weight" that takes this signal to the neuron.

The mathematical equation that forms this model takes into account that the input stimuli have their effects thoughtfully summed. That is:

$$v = \sum_{i=1}^{n} \omega_i \cdot x_i + b \tag{4}$$

where, "v" is the thoughtfully sum of the input stimuli, "$w_i.x_i$" is the product between an input signal "x" and the synaptic weight "w" of the connection that leads the signal to the neuron, "b" is the particular polarization signal of this neuron.

The perceptron response to incoming stimuli is a function of their considered sum. That is:

$$y = \varphi(v) \tag{5}$$

where, "y" is the neuron's response, "$\varphi(v)$" is the neuron's "activation function", correlating the output to input stimuli.

The perceptron activation function "φ (v)" must be non-linear in order to allow the entire ANN to represent non-linear functions, in addition to being continuous and "soft" in order to be differentiable in all its interval of consideration.

An example of a very popular activation function in MLP networks is the tangent-hyperbolic function:

$$\varphi(v) = \alpha.\tanh(\beta.v) \tag{6}$$

where "α" and "β" are constants.

For this activation function, Ref. [14] considers α = 1.7159 and β = 2/3 as appropriate values.

3.2 Layered Neural Networks

As we mentioned previously, an ANN-MLP is composed by layers association of perceptrons. Fig. 2 shows a model for a MLP.

In this architecture we have an "input layer", where

the network stimuli (signals) are applied, which are propagated layer by layer until the network output, from the "1st hidden layer" of neurons until the network "output layer", which will present the results of all network processing to the external environment.

It should also be noted that in a MLP the inputs from a given neuron are connected to the outputs of all neurons of the preceding layer, and the output of this neuron is connected to all inputs of the neurons of immediately posterior layer.

3.3 ANN Learning — The Backpropagation Algorithm

During the learning phase, so that the network can learn the desired behavior, a recursive algorithm such as the "error backpropagation algorithm" (or error backpropagation) is used.

In the backpropagation algorithm, during the network training phase, an "*x*" input pattern is presented to the network and propagated towards the output, generating an "*o*" output pattern, which is compared to the desired value "*d*" for the concerned input pattern.

The "*o*-*d*" difference between the desired pattern and the actually obtained in the network output is the error "*e*" of this learning phase for this pattern. If $e = 0$, then we conclude that the network has "learned" the "*x*" input pattern. If $e \neq 0$, then the error should be backpropagated (from the output layer toward the 1st hidden layer) in order to adjust the connections weights between the layers' neurons.

Formally, we have:

$$\text{error signal: } e_j = d_j - o_j$$

where, d_j is an element of the desired output vector for the presented input pattern, o_j is an output vector element obtained with the input vector propagation, and e_j is the error vector element obtained.

According to Ref. [13], to minimize ANN training difficulties, the error function should be non-linear. Thus, for the presented input pattern, we define the instantaneous sum of squared errors on the network output by:

$$\varepsilon = \frac{1}{2} \cdot \sum_{j=1}^{m} e_j^2 \tag{7}$$

where, "*m*" is the number of elements of the error vector (which equals the number of neurons in the output layer of the network).

If we consider all input patterns to be presented in the training phase, we can determine the "mean square error" for these patterns during an "epoch" ("epoch" is every propagation/backpropagation held for all input patterns presented to the network in its training phase). Therefore,

$$\varepsilon_{QM} = \frac{1}{N} \cdot \sum_{n=1}^{N} \varepsilon(n) \tag{8}$$

where, N is the number of training patterns presented

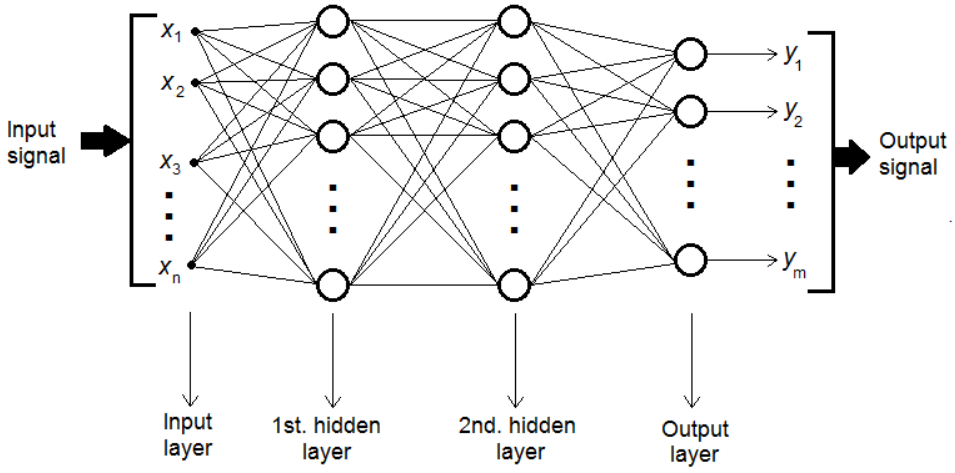

Fig. 2 Basic model for a MLP.

to the network and $\varepsilon(n)$ is the obtained error while presenting the pattern n.

Therefore, the less will be the mean square error value (ε_{QM}) the better the network input patterns learning in a given epoch. Thus, if $\varepsilon_{QM} = 0$, the network learned with absolute precision all patterns presented.

Thus, a proposal to a MLP training can be:

• While ε_{QM} is not low enough:

(a) For each training pattern:

- propagate this pattern, toward the output, layer by layer;

- calculate the error "*e*" between the desired output and the output obtained;

- Retropropagate the output error towards the input, correcting the synaptic weights of each connection.

- End

(b) Update the value of ε_{QM}.

• End

In the algorithm presented above, the error correction to adjust the synaptic weights can be achieved by the "delta rule". This synaptic weights adjustment is primordial to the ANN learning, since the knowledge acquired by the network is condensed in these. Considering the connection between two neurons of consecutive layers, updating the synaptic weight "ω_{ji}" between these neurons, from iteration to another, is given by:

$$\omega_{ji}(t+1) = \omega_{ji}(t) + \Delta\omega_{ji} \quad (9)$$

where, $\omega_{ji}(t)$ is the synaptic weight current value, $\Delta\omega_{ji}$ is the synaptic weight correction to be applied, $\omega_{ji}(t + 1)$ is the updated value (next value) of synaptic weight.

The correction value calculation of the synaptic weight (ω_{ji}) should come in the opposite gradient direction of the accumulated error in the propagation of an epoch of input samples (training set). Thus, according to the delta rule, the synaptic weight correction can be given by:

$$\Delta\omega_{ji} = \eta \cdot \delta_j \cdot y_i \quad (10)$$

where, η is the neuron's "learning rate" ($0< \eta <1$), which value, arbitrated by the network user, defines how quickly the network will converge to the minimum point; δ_j is the "local gradient" of neuron j for error correction, calculated by the network according to the backpropagated error; y_i is the neuron "i" output value.

The local gradient "δ_j" value for the neuron "j" can be calculated as follows:

$\delta_j = \varphi' \cdot e_j$, if neuron j belongs to the output layer; or

$\delta_j = \varphi' \cdot \sum_k \delta_k \cdot \omega_{kj}$, if neuron j belongs to a hidden layer.

In order to improve the delta rule convergence, we may use a "time factor" (μ) which will consider $\Delta\omega_{ji}$ value of the previous iteration for calculating the next value:

$$\Delta\omega_{ji}(t) = \mu \cdot \Delta\omega_{ji}(t-1) + \eta \cdot \delta_j(t) \cdot y_i(t) \quad (11)$$

being "μ" arbitrated between 0 and 1.

Simply, the ANN design and training process can be described as:

Network design:

(1) Define the size of the input vector of the network, i.e., the number of nodes of the input layer of the network;

(2) Define the size of the output vector of the network, i.e., the number of neurons in the output layer of the network;

(3) Define the number of hidden layers of the network;

(4) Define the number of neurons in each hidden layer of the network;

(5) Assign random values (between -1 and 1) for the bias signal of each neuron;

(6) Assign random values (between -1 and 1) for the synaptic weights of each connection of the ANN;

(7) Define the activation function of the neurons and its derived;

(8) End.

Network training:

(1) Define values for η and μ;

(2) Define the threshold value for ε_{QM};

(3) While ε_{QM} is not low enough:

a. For each training pattern:

- Apply the pattern to the input x of the ANN;
- Calculate the output y of each neuron in the first hidden layer, applying these outputs to the inputs of the next layer, also calculating the outputs of this layer until the output layer (propagation of the input signal);
- Calculate the error (e) for this pattern, comparing the desired value for the output (d) and the value obtained at the output (o) for the applied input;
- From the output layer, toward the input layer of the network (error backpropagation):
  - Calculate the local gradient (δ) of each neuron of each layer.
- From the output layer toward the input layer of the network:
  - Calculate $\Delta\omega$ of each synaptic connection;
  - Update ω of each synaptic connection.

End.

b. Update the value of ε_{QM} for this epoch;

c. End.

(4) End.

After the network training, its use is accomplished simply by presenting any input pattern, which is propagated towards the network output, where the network response can be observed to the input stimulus presented.

4. Implementation on Case Studies

A MLP-ANN implementation and simulation can be accomplished through computer programs development in various languages. These authors implemented ANNs in the programming language of "SciLab" (http://www.scilab.org), obtaining the following results for the simulations.

According to Refs. [16, 17], an ANN with a single intermediate layer is sufficient to approximate any continuous function. In turn, Cybenko [18] states that any mathematical function can be modeled by an ANN with no more than two intermediate layers. Funuhashi [19] also investigates the number of intermediate layers needed for class function implementation in an ANN. Thus, as the series we desire to simulate can be approximated by continuous functions, all simulations were performed with ANNs that had a single hidden layer. Indeed, in the simulations shown below we used an ANN with four neurons in the input layer, one hidden layer with five neurons and an output layer with a single neuron. In all cases we also used the same training parameters: learning rate (η) equal to 0.3, time factor (μ) equal to 0.5 and the hyperbolic tangent as the activation function of the neurons.

Table 5 shows the results for the Fibonacci series simulation, initially presented in Table 1.

In Fibonacci series, the ratio x_{k+1}/x_k converges to 1.618. In our simulation, there is a convergence to 1.612 at the far right of Table 5. On average, we found that this ratio, for data simulation, is around 1.641, indicating an ANN reasonable learning of this series.

The resulting simulation for the Brazilian population evolution in the last century is presented in Table 6.

We verify that the simulated value is very close to the actual values presented by IBGE (The Brazilian Institute of Geography and Statistics) in Ref. [1], an error at most 2.5% was observed. When required to infer information for 2000 and 2010, the network

Table 5 Simulation of Fibonacci series on an ANN.

Real value	Simulated value
5	4.3
8	7.0
13	11.8
21	19.4
34	31.9
55	52.1
89	85.3
144	139.9
233	229.9
377	377.2
610	608.2

presented the values shown in Table 7, which are compared to the data presented by IBGE in Ref. [1] again.

We observe that the inferred value for 2000 has a reasonable approximation to the actual value (around 6.5% of the observed value in fact). In turn, the result inferred for 2010 can be partly explained by a reduction in the population growth trend, verified since the late twentieth century, which changed the studied series behavior.

Table 8 shows a comparison between AMA effectively verified and simulated by an ANN for EMBR3 share.

In Table 8, there is a difference of less than 1% between simulated and verified values. When asked to infer AMA for the days between 25/03 and 27/03/13 the network presented the following results, which are compared with the values verified in Table 9.

Table 6 Brazilian population evolution simulation in the twentieth century.

Year	Official IBGE data (in millions)	Simulated value (in millions)
1940	41.2	40.9
1950	51.9	50.8
1960	70.9	70.0
1970	94.5	94.0
1980	121.1	118.0
1990	146.9	146.6

Table 7 A Brazilian evolution simulation inferred by the network.

Year	Official IBGE data (in millions)	Simulated value (in millions)
2000	169.6	181.3
2010	190.8	227.4

Table 8 Arithmetic moving average of closing prices of EMBR3 share.

Date	AMA verified (Brazil Reais)	AMA simulated (Brazil Reais)
14/03/13	17.22	17.20
15/03/13	17.25	17.34
18/03/13	17.27	17.41
19/03/13	17.28	17.43
20/03/13	17.28	17.45
21/03/13	17.33	17.44
22/03/13	17.40	17.45

Table 9 Comparison between inferred and verified values by the network for the period between 25/03/13 and 27/03/13.

Date	AMA verified (Brazil Reais)	AMA simulated (Brazil Reais)
25/03/13	17.37	17.54
26/03/13	17.53	17.58
27/03/13	17.58	17.63

Observing the values we may note that even in this case the error between verified and inferred values is also less than 1%.

5. Final Conclusions

In this study, we presented an alternative way to the study and teaching of time series through ANNs. Throughout the text, the MLP-ANN model was discussed and we presented application examples.

Through this study, we can state that ANN application on time series that have a behavior perfectly defined by an equation, as in the Fibonacci Series, does not produce practical effect, as ANN simulation are always approximations of values that can be fully determined. In turn, with respect to series whose equations can be only approximate, as in the case of the series representing the Brazilian population evolution, the use of ANNs can be a very interesting analysis tool by which implicit behaviors are likely to be detected. Regarding the more complex nature series, as in the case of time series of share prices, the use of ANNs may constitute a fundamental tool of analysis that demonstrates behaviors that could hardly likely to be detected by other methods.

The major inconvenient in using ANNs for time series analysis lies in the difficulty of finding the most suitable network topology for each case to be simulated. Using a large number of intermediate layers is not recommended because each time the measured error is propagated during training to the previous layer it can turn out to be higher. Even respecting the modeling conditions presented in Refs. [16-18], they suggest the use of no more than two intermediate layers to any mathematical function, setting the number of neurons in each hidden layer is

not a simple task.

The number of nodes in an ANN hidden layer strongly depends on training patterns distribution and network validation. The use of many units may facilitate memorizing training patterns. However, this will limit the ability to extract general features that allow generalization or recognition of patterns not seen during training (overfitting). Moreover, a very small number of nodes may force the network to spend too much time trying to find a good representation. If the number of examples is much larger than the number of connections between the nodes, overfitting is improbable, but underfitting can occur (network does not converge during its training).

Finally, this proposal applicability in the classroom is justified by the need to provide engineering courses students' new tools for the study of cases that are not necessarily covered by the traditional teaching/learning tools (forecasting, for example). In this sense, the use of neural networks in time series teaching also seeks to instigate students and researchers to develop new and better tools for their everyday problems, considering thus, the indivisibility of teaching and research in an undergraduate course.

References

[1] IBGE (Brazilian Institute for Geography and Statistics). 2010 Census Synopsis. Rio de Janeiro: IBGE, 2011. Accessed February 15, 2013. http://www.ibge.gov.br/home/estatistica/populacao/censo2010/tabelas_pdf/Brasil_tab_1_4.pdf.

[2] Lippmann, R. P. 1987. "An Introduction to Computing with Neural Nets." *IEEE ASSP Magazine* 4: 4-22.

[3] Burke, H., Rosen, D., and Goodman, P. 1995. "Comparing the Prediction Accuracy of Artificial Neural Networks and Other Statistical Models for Breast Cancer Survival." In *Neural Information Processing Systems 7,* edited by Tesauro, G., Touretzky, D. S., Leen, T. K. Cambridge: MIT Press.

[4] Mighell, D. A., Wikinson, T. S., and Goodman, J. W. 1988. "Back Propagations and Its Application to Handwritten Signature Verification." In *Advances in Neural Information Processing Systems 2,* edited by Lippmann, R. P., Moddy, J. E., and Touretzky, D. S. Massachusetts: Morgan Kaufmann.

[5] Philip, A. A., Taofiki, A. A., and Bidemi, A. A. 2011. "Artificial Neural Network Model for Forecasting Foreign Exchange Rate." *WCSIT (World of Computer Science and Information Technology Journal)* 1 (3): 110-8.

[6] Reategui, E., and Campbell, J. A. 1994. "A Classification System for Credit Card Transactions." In *Proceedings of the Second European Workshop on Case-Based Reasoning,* 167-74.

[7] Tarsauliya, A., Kant, S., Kala, R., Tiwari, R., and Shukla, A. 2010. "Analysis of Artificial Neural Network for Financial Time Series Forecasting." *International Journal of Computer Applications* 9 (5): 16-22.

[8] Yoda, M. 1994. *Predicting the Tokyo Stock Market.* New Jersey: John Wiley & Sons.

[9] Haykin, S. 2001. *Neural Networks—Principles and Practice.* 2nd edition. New York: Bookman.

[10] Hush, D. R., and Horne, B. G. 1993. "Progress in Supervised Neural Networks—What's New Since Lippmann?" *IEEE Signal Processing Magazine* 1: 8-39.

[11] Fahlman, S. E. 1988. *An Empirical Study of Learning Speed in Backpropagation Networks.* Technical report, Carnegie Mellow University.

[12] Günther, F., and Fritsch, S. 2010. "Neuralnet: Training of Neural Networks." *The R Journal* 2 (1): 30-8.

[13] German, S., Bienestock, E., and Doursat, R. 1992. "Neural Networks and the Bias-Variance Dilemma." *Neural Computation* 4: 1-58.

[14] LeCun, Y. 1989. *Generalization and Network Design Strategies.* Technical report CRG-TR-89-4, Department of Computer Science, University of Toronto, Canada.

[15] Andrews, R., and Geva, S. 1994. "Rule Extraction from a Constrained Error Backpropagation MLP." In *Proceedings of the 5th Australian Conference on Neural Networks,* 9-12.

[16] Hertz, J., Krogh, A., and Palmer, R. G. 1991. *Introduction to the Theory of Neural Computation, volume Lecture Notes. Vol. 1 of Santa Fe Institute Studies in the Science of Complexity.* Massachusetts: Addison-Wesley.

[17] Cybenko, G. 1989. "Approximation by Superpositions of a Sigmoid Function." *Mathematic of Control, Signals and Systems* 2: 303-14.

[18] Cybenko, G. 1988. *Continuos Valued Neural Networks with Two Hidden Layers Are Sufficient.* Technical report, Department of Computer Science, Tufts University.

[19] Funuhashi, K. I. 1989. "On the Approximate Realization of Continuos Mappings by Neural Networks." *Neural Networks* 2: 183-92.