Neural Network Modeling

Course Notes

Neural Network Modeling Course Notes was developed by David Yeo, Ph.D., based on the course originally written by William J. E. Potts. Additional contributions were made by Robert Blanchard. Editing and production support was provided by the Curriculum Development and Support Department.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

Neural Network Modeling Course Notes

Copyright © 2016 SAS Institute Inc. Cary, NC, USA. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

Book code E70569, course code LWDMNN41/DMNN41, prepared date 26Aug2016. LWDMNN41_001

ISBN 978-1-62960-187-8

Table of Contents

Course Description vi

Prerequisites vii

Chapter 1 Introduction to Neural Networks 1-1

1.1 A Brief History of Neural Networks 1-3

1.2 Traditional Approaches to Nonlinear Modeling 1-15

Demonstration: Nonlinear Regression 1-17

Demonstration: Nonparametric Regression 1-30

1.3 Advantages and Disadvantages of Neural Networks 1-34

Demonstration: Two Spirals Problem 1-39

1.4 Chapter Summary 1-46

1.5 Solutions 1-48

Chapter 2 Network Architecture 2-1

2.1 Linear Perceptrons 2-3

Demonstration: Modeling Assorted Target Distributions 2-8

2.2 Multilayer Perceptrons 2-19

Demonstration: Compositional Data 2-26

Exercises 2-38

2.3 Radial Basis Functions 2-40

Demonstration: Plotting Basis Functions 2-49

Exercises 2-58

2.4 Chapter Summary 2-62

2.5 Solutions 2-63

Solutions to Exercises 2-63

Chapter 3 Learning 3-1

3.1 Avoiding Overfitting and Inferior Local Minima 3-3

Demonstration: Early Stopping 3-10

Exercises 3-19

3.2 Parameter Estimation 3-21
Demonstration: Robust Networks 3-28
Exercises 3-32
3.3 Numerical Optimization Methods 3-35
Demonstration: Training Tracks 3-46
3.4 Chapter Summary 3-53
3.5 Solutions 3-54
Solutions to Exercises 3-54

Chapter 4 The NEURAL Procedure 4-1

4.1 Overview of PROC NEURAL 4-3
Demonstration: Linear Perceptrons Using PROC NEURAL 4-9
4.2 Input Selection Using PROC NEURAL 4-15
Demonstration: Variable Importance 4-18
Demonstration: Sensitivity-Based Pruning 4-26
Exercises 4-32
4.3 Sequential Network Construction 4-33
Demonstration: Sequential Network Construction 4-35
Exercises 4-46
4.4 Generalized Additive Neural Networks (Self-Study) 4-47
Demonstration: Generalized Additive Neural Network 4-54
4.5 Stochastic Gradient Descent 4-71
Demonstration: Mini-Batch Gradient Descent 4-75
4.6 Chapter Summary 4-84
4.7 Solutions 4-85
Solutions to Exercises 4-85

Chapter 5 Augmented Networks 5-1

5.1 Time Delay Neural Networks 5-3
Demonstration: Implementing a Time Delay Neural Network 5-13
5.2 Surrogate Models 5-22

Demonstration: Interpreting a Neural Network with a Continuous Target 5-24

Demonstration: Interpreting a Neural Network with a Categorical Target (Self-Study) 5-27

5.3 Chapter Summary 5-31

5.4 Solutions 5-32

Chapter 6 The HP Neural Node 6-1

6.1 Introduction to High-Performance Analytics 6-3

6.2 Overview of the HP Neural Node 6-9

Demonstration: Comparing One- and Two-Layer Networks 6-16

6.3 Chapter Summary 6-28

6.4 Solutions 6-29

Chapter 7 The AutoNeural Node (Self-Study) 7-1

7.1 Introduction 7-3

7.2 Architectures 7-12

Demonstration: Comparing AutoNeural Architectures 7-18

7.3 Chapter Summary 7-23

7.4 Solutions 7-24

Appendix A PROC DMDB and PROC NEURAL User's Guide A-1

A.1 The DMDB Procedure A-3

A.2 The NEURAL Procedure A-14

Appendix B Empirical Partial Residuals B-1

B.1 Empirical Partial Residuals B-3

Demonstration: Illustrating Empirical Partial Residuals B-4

Appendix C References C-1

C.1 References C-3

Course Description

This course helps you understand and apply two popular artificial neural network algorithms, multi-layer perceptrons and radial basis functions. Both the theoretical and practical issues of fitting neural networks are covered. Specifically, this course teaches you how to choose an appropriate neural network architecture, how to determine the relevant training method, how to implement neural network models in a distributed computing environment, and how to construct custom neural networks using the NEURAL procedure.

To learn more…

For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at http://support.sas.com/training/ as well as in the Training Course Catalog.

For a list of other SAS books that relate to the topics covered in this course notes, USA customers can contact the SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the SAS Bookstore on the web at http://support.sas.com/publishing/ for a complete list of books and a convenient order form.

Prerequisites

Before attending this course, you should

- have an understanding of basic statistical concepts, which you can gain from the Statistics 1: Introduction to ANOVA, Regression, and Logistic Regression course.
- have completed the SAS® Programming 1: Essentials course or have equivalent knowledge.
- be familiar with SAS Enterprise Miner software. You can gain this knowledge from the Applied Analytics Using SAS® Enterprise Miner™.
- have completed a college-level calculus course.

Chapter 1 Introduction to Neural Networks

1.1 A Brief History of Neural Networks.....1-3

1.2 Traditional Approaches to Nonlinear Modeling 1-15

Demonstration: Nonlinear Regression.....1-17

Demonstration: Nonparametric Regression.....1-30

1.3 Advantages and Disadvantages of Neural Networks 1-34

Demonstration: Two Spirals Problem.....1-39

1.4 Chapter Summary.....1-46

1.5 Solutions 1-48

Solutions to Student Activities (Polls and Quizzes) 1-48

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

1.1 A Brief History of Neural Networks

Objectives

- Provide a brief history of neural networks.
- Describe key concepts underlying neural networks.

3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Adaptive Systems

- Artificial neural networks mimic key aspects of the brain, in particular its ability to learn from experience.
- This course focuses on two important variants:
  - multilayer perceptrons
  - radial basis functions

4

A biological brain has many features that would be desirable in artificial systems:

- It can easily adjust to a new environment (learn).
- It is highly parallel, and therefore, very fast.
- It can deal with information that is fuzzy, probabilistic, noisy, or inconsistent.
- It is robust and fault tolerant.
- It is highly energy efficient.
- It is relatively small.

Neural networks attempt to mimic key aspects of a brain, in particular its ability to learn from experience. Although inspired by cognitive science, in particular neurophysiology, neural networks largely draw their methods from statistical physics (Hertz et al. 1991). There are dozens, if not hundreds, of neural network algorithms. But despite this plethora of models, all neural networks fall into one of two broad classes:

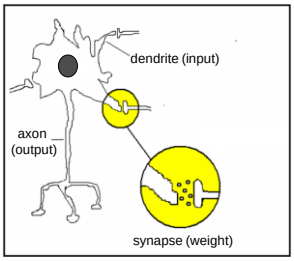
- *supervised learning*
- *unsupervised learning*

In supervised learning, one or more target variables guide learning. In contrast, in unsupervised learning there is no target; the algorithm must look for regularities in the data.

This course focuses exclusively on supervised learning, in particular, on the *multilayer perceptron* and *radial basis function* algorithms, both of which are found in the Neural Network node. It should be noted, however, that SAS Enterprise Miner offers a number of other neural network algorithms. For example, the Kohonen/SOM (self-organizing map) node implements unsupervised learning, and the DMNEURAL node is an additive nonlinear supervised learning model that uses principal components as inputs.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The Neuron



- Inputs arriving through *dendrites* are first weighted by adaptable *synapses* before being summed.
- If the sum is greater than an adaptable *bias*, the neuron sends a signal down its *axon* to other neurons.

5

Neurons are the fundamental units of cognition. In order to imitate neurons, it was first necessary to understand their mechanisms. There is still much to be learned, but the key functional aspects of neurons, and even small systems (networks) of neurons, are now known.

For example, the inputs to a neuron arrive through structures known as *dendrites*. The dendrites do not actually contact the receiving neuron but rather they transmit their information to the neuron by sending chemicals (neurotransmitters) across a *synaptic gap*. The transmitted chemicals either *excite* the receiving neuron, making it more likely to "fire," or they *inhibit* the neuron, making it less likely to become active. The amount of neurotransmitter released across the gap determines the relative strength of each dendrite's connection to the receiving neuron. In essence, each synapse "weights" the relative strength of its arriving input. The synaptically weighted inputs are summed. If the sum exceeds an adaptable threshold (or bias) value, the neuron sends a pulse down its axon to the other neurons in the network to which it connects.

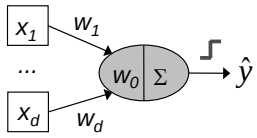
A key discovery of modern neurophysiology is that synaptic connections are adaptable; they change with experience. The more active the synapse, the stronger the connection becomes. Conversely, synapses with little or no activity fade and, eventually, die off (atrophy). This is thought to be the basis of learning.

Although there are branches of neural network research that attempt to mimic the underlying biological processes in detail, most neural networks do not try to be biologically realistic.

The terms *neuron* and *unit* are used interchangeably throughout this course.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The McCulloch-Pitts Neuron



- A McCulloch-Pitts neuron is defined by the equation:

$$\hat{y} = f\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)$$

- The step function, *f(.)*, turns each neuron into a linear classifier/discriminator.

6

In a seminal paper with the rather ominous title "A logical calculus of the ideas immanent in nervous activity," McCulloch and Pitts (1943) gave birth to the field of artificial neural networks. Using predicate calculus, they ***proved*** that networks of neurons could take any set of binary inputs and transform them into any desired set of binary outputs. That is, McCulloch and Pitts proved that neural networks are universal approximators, able to model any input-output relationship, to any degree of precision.

The fundamental element of a McCulloch-Pitts network is, not surprisingly, a *McCulloch-Pitts neuron.* As in real neurons, each input (x_i) is first weighted (w_i) and then summed. To mimic a neuron's threshold functionality, a bias value (w_0) is added to the weighted sum, predisposing the neuron to either a positive or negative output value. The result is known as the neuron's *net input*:

$$net = w_0 + \sum_{i=1}^{k} w_i x_i$$

Notice that this is the classic linear regression equation, where the bias term is the y-intercept and the weight associated with each input is the input's slope parameter. The McCulloch-Pitts neuron's final output was determined by passing its net input value through a *step function* (defined below), turning each neuron into a linear classifier/discriminator.

$$f(net) = \begin{cases} 1 \text{ if } net \geq 0 \\ -1 \text{ if } net < 0 \end{cases}$$

Although the nominal values 1 and 0 are typically used to represent true and false, in fact any two-valued system suffices. Thus, the values 1 and -1 are perfectly valid and, on occasion, more computationally convenient representations of a binary state.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The Hebb Learning Rule

- The strength of the connection between neurons *i* and *j* should be adjusted in accordance with the equation:

$$\Delta w_{ij} = \eta\, \hat{y}_i x_j$$

- The *learning rate*, η, scales the amount of adjustment.
- Small learning rates lead to long convergence times.
- Large learning rates risk divergence.

7

The weights of the neurons in a McCulloch-Pitts network had to be set manually. In fact, there was no known biologically plausible method of learning until Canadian neuropsychologist D. O. Hebb (1949) devised his now famous learning rule:

> "The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated,' so that activity in one facilitates activity in the other."

In more formal terms, Hebb's learning rule states that the strength of the connection between neurons i and j should be adjusted in accordance with the following equation:

$$\Delta w_{ij} = \eta \hat{y}_i x_j$$

where x_j is the activation arriving from another neuron and $\hat{y}_i$ is the receiving neuron's output.

The η term in the above equation is the neuron's *learning rate*, where $0 < \eta \leq 1$. Care must be taken when specifying the learning rate value. Setting it too small results in very long training times, and setting it too large can result in movement away from the desired solution (i.e., divergence). Unfortunately, the optimal magnitude of the learning rate is problem specific.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The Widrow-Hoff Delta Rule

- Hebb's learning rule is unstable.
- Widrow and Hoff proposed a variant of Hebb's rule, which is stable under a range of learning rates:

$$\Delta w_{ij} = \eta\,(y_i - \hat{y}_i)\,x_j$$

- They called their learning model the *delta rule.*
- Because the delta rule reduces the sum of squared error, it is also known as the *least mean squares rule*.

8

The Hebb learning rule has a fatal flaw: it results in weights that are unstable. Regardless of the learning rate that is specified, the weights that it generates will always either increase or decrease without bounds, ultimately producing either very large positive or negative values (Principe et al. 2000).

Widrow and Hoff (1960) came up with a variant of Hebb's learning rule that stabilized the weights for a wide range of learning rates. They called their variant the *delta rule*. The delta rule replaces the receiving neuron's output value, the $\hat{y}_i$ term in the Hebb rule, with its *error* value, ($y_i - \hat{y}_i$), yielding the following:

$$\Delta w_{ij} = \eta\left(y_i - \hat{y}_i\right)x_j$$

Although not at first obvious, the delta rule minimizes the sum of squared error:

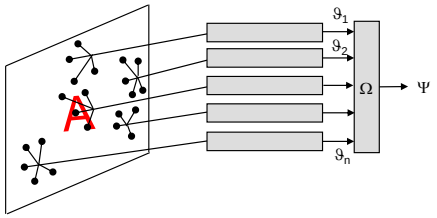
$$E = \frac{1}{2}\sum_i \left(y_i - \hat{y}_i\right)^2$$

This means that the Widrow-Hoff delta rule performs least mean squares parameter estimation, which is why the delta rule is also commonly known as the *LMS (least mean square) rule*.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The Perceptron

- The perceptron is a pattern-recognition device, initially designed for optical character recognition.



- Each processing unit is a McCulloch-Pitts neuron.
- A perceptron with n outputs is a discriminator function that divides the input space into n distinct regions.

9

A variant of the delta rule was first implemented in a computer model by Frank Rosenblatt (1959), who called his learning machine a *perceptron*:

> "As an analogue to the biological brain, the perceptron, more precisely, the theory of statistical separability seems to come closer to meeting the requirements of a functional explanation of the nervous system than any system previously proposed. … The future of information processing devices, which operate on statistical rather than logical principles seems to be clearly indicated."

(*Two theorems of statistical separability in the perceptron*, p. 449)

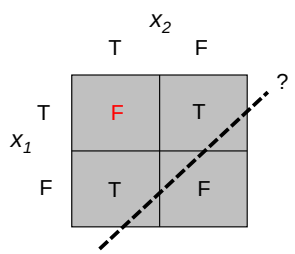
Rosenblatt's comparison to logic-based models of cognition was not accidental. The seminal McCulloch-Pitts paper had given birth not only to neural networks, but also to the field of artificial intelligence (AI). The two fledgling fields offered vastly different approaches to modeling intelligence. Although Rosenblatt and others in the neural network community focused on the statistical properties of neural networks as they learn, the artificial intelligence community focused on the logic processing capability of networks. From the artificial intelligence perspective, a single neuron implements an IF...THEN rule (i.e., if these inputs occur, then generate these outputs). And networks of neurons combine to form the complex logical predicates that guide our thinking and decision making.

Rosenblatt's dismissal of their fledgling field angered many in the artificial intelligence community. One such person was Marvin Minsky, a brilliant researcher and a former classmate of Rosenblatt. Rosenblatt had founded his optimism on the perceptron's early success learning patterns, like its ability to identify the characters A-Z when presented to an optical scanner (character recognition). Although perceptions worked on some problems, they did not work on others. No one knew why. Minsky decided to find out.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Limitations of Rosenblatt's Perceptron

- Minsky and Papert found that Rosenblatt's perceptron could solve only linearly separable problems.
- The exclusive-or truth table (see below) is an example of a problem that is not linearly separable.



10

In collaboration with Seymour Papert, Minsky wrote a book simply titled *Perceptions*. The book was a mathematical *tour de force* in which Minsky and Papert demonstrated that the basic perceptions proposed by Rosenblatt were severely limited. They demonstrated, for example, that the simple perceptron could never learn the exclusive-or (XOR) logical relationship, shown in tabular form above.

The reason for its failure is that a perceptron that directly connects input neurons to output neurons forms a linear model, capable only of making linearly separable discriminations.

A linearly separable problem is one in which a plane can be found in the input space that separates the +1 patterns from the -1 patterns. Such a plane must exist for each output.

Minsky and Papert discovered that, when the patterns to be learned were not linearly separable, the perceptron failed. Their book severely impacted the neural network field. To quote Johnson (1991):

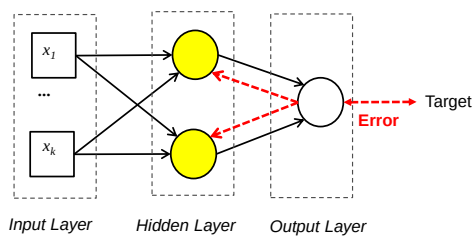
> "In 1972 *Perceptions* went into its second printing. On the first page of the new edition was a dedication: 'In memory of Frank Rosenblatt,' who had died alone in a boating accident that some thought was suicide. But the epitaph might as well have been applied to the entire field. It would be more than a decade before many people started taking neural networks seriously again." (p. 149)

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The Generalized Delta Rule

The *generalized delta rule* is a two-step process:

1. Propagate the input activation forward through the network and calculate the output error.
2. Back propagate the error, and adjust the weights as you go. If not converged, then go to Step 1.



11

Rosenblatt had experimented with *multilayer perceptions* (MLPs), in which the input layer neurons feed into a *hidden layer* of neurons, rather than directly into the output layer (see diagram). Adding a hidden layer of neurons was important because it made it possible to generate nonlinear input-output mappings, thereby **completely overcoming the Minsky-Papert objection**. What was not clear, however, was how the weights of these more complex networks could be learned.

The Widrow-Hoff learning rule needs both input and output values to adjust its connections. Because the output of the hidden layer neurons was passed through a discontinuous step function before being sent to the output layer neurons, there was no way to determine the pre-transformed net activation of the hidden units. The *generalized delta rule* (*back propagation*) solved this problem by replacing the discontinuous step function with a continuous sigmoid function. The *logistic* function (below) was initially used.

$$f(net) = \frac{1}{1+\exp(-net)} \text{ where } net = w_0 + \sum_{i=1}^{k} w_i x_i$$

Because it is continuous, the effect of a sigmoid can be "undone" by applying its derivative to the output that it generates. For the logistic sigmoid, this derivative is given by the equation:

$$f'(net) = f(net)(1 - f(net))$$

The recovered value enables the Widrow-Hoff delta rule to adjust any weight in a network in proportion to its contribution to the output layer error. (See "Details" on the next page for more information.)

Classic back propagation is often slow to converge on a solution. In the next chapter, you examine two more efficient variants of the back propagation algorithm: QPROP and RPROP.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Details (Self-Study)

Given:

- x_i is the ith input variable.
- $w_{ij}^{(m)}$ is the weight connecting the ith output to the jth input in layer m.
- $g(.)$ is the nonlinear activation function (for example, the logistic sigmoid).
- $g'(.)$ is the derivative of the nonlinear activation function.

Then:

1. Initialize the weights to small random values, to ensure that each hidden unit behaves differently. Optionally, initialize the hidden-to-output weights to zero, and the output bias to the target mean.
2. Choose input pattern $\mathbf{x}_k$ and apply it to the *input layer* so that $\mathbf{H}_k^{(0)} = \mathbf{x}_k$.
3. Propagate the input pattern ***forward*** through the network using the following equation:

$$\mathbf{H}_i^{(t)} = g\left(\mathbf{h}_i^{(t)}\right) = g\left(\sum_j w_{ij}^{(t)} \mathbf{H}_j^{(t-1)}\right)$$

4. Using the Widrow-Hoff delta rule, compute the weight update values for output layer M:

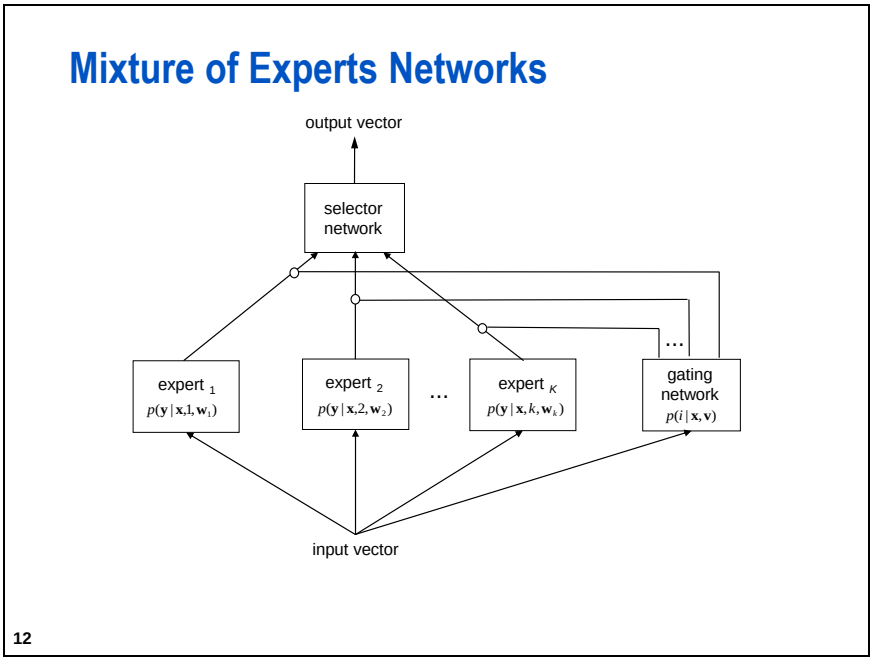
$$\delta_i^{(M)} = g'(\mathbf{h}_i^{(M)})\left[y_i^{(p)} - \mathbf{H}_i^{(M)}\right]$$

5. Compute the updates for the preceding layers by propagating the error ***backward***:

$$\boldsymbol{\delta}_i^{(m-1)} = g'(\mathbf{h}_i^{(m-1)})\sum_j w_{ij}^{(m)} \boldsymbol{\delta}_j^{(m)}$$, where $m = M, M\text{-}1, \ldots, 2$

6. Update all connections using $\Delta w_{ij}^{(m)} = \eta \boldsymbol{\delta}_i^{(m)} \mathbf{H}_j^{(m-1)}$ and $w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$.
7. If convergence has not been achieved, go to step 2 and repeat for the next pattern.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Recently there has emerged a point of consensus between the artificial intelligence and neural network camps with respect to the underlying structure of intelligence. It comes in the form of a *mixture model*. Mixture models use a divide-and-conquer strategy in which complex tasks are assigned to a number of connected, semi-autonomous specialists (i.e., agents or experts), themselves relatively mindless.

Minsky called this the "society of mind".

> I'll call 'Society of Mind' this scheme in which each mind is made of many smaller processes. These we'll call agents. Each mental agent by itself can only do some simple thing that needs no mind or thought at all. Yet when we join these agents in societies—in certain very special ways—this leads to intelligence. (*The Society of Mind*, 1986, p. 17)

Other members of the artificial intelligence community have proposed similar models. For example, roboticist Rodney Brooks' (1986) *subsumption architecture* decomposes complicated behavior into progressively simpler behavior modules, organized into layers.

Members of the neural network community have also embraced the mixture paradigm. Jacobs et al. (1991) used the following conditional relationship as the basis of their *mixture of experts* model:

$$p(\mathbf{y}\,|\,\mathbf{x},\mathbf{w}) = \sum_{i=1}^{k} p(i\,|\,\mathbf{x},\mathbf{v})\,p(\mathbf{y}\,|\,\mathbf{x},i,\mathbf{w}_i)$$

Here $p(\mathbf{y}\,|\,\mathbf{x},i,\mathbf{w}_i)$ is the ith expert's prediction, and $p(i\,|\,\mathbf{x},\mathbf{v})$ is the probability that the ith expert applies.

The experts' estimates are constrained to ensure that the predictions across all target classes sum to one. If $\beta_i(\mathbf{x},\mathbf{v})$ is the ith specialist's estimate, given the current input vector $\mathbf{x}$ and weight vector $\mathbf{v}$, then the value returned by expert "i" is given by the softmax equation:

$$p(i\,|\,\mathbf{x},\mathbf{v}) = e^{\beta_j(\mathbf{x},\mathbf{v})} \Big/ \sum_j e^{\beta_j(\mathbf{x},\mathbf{v})}$$

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

A *gating network* allocates each new case to one or more of the specialists. A *selector network* then combines each specialist's estimate with the probability that the specialist applies by minimizing the following likelihood function,

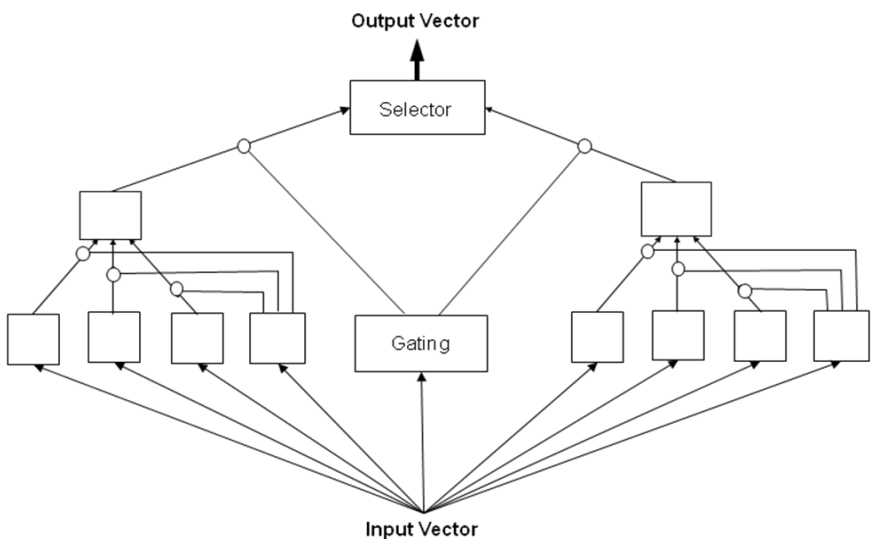
$$E = -\sum_n \ln\left\{\sum_{i=1}^{m} \alpha_i(\mathbf{x}^n)\phi_i(\mathbf{t}^n \mid \mathbf{x}^n)\right\}$$

where $\alpha_i(.)$ are mixing coefficients and $\phi_i(.)$ are Gaussian functions given by the equation:

$$\phi_i(\mathbf{t} \mid \mathbf{x}) = \frac{1}{(2\pi)^{c/2}} \exp\left\{-\frac{\left\|\mathbf{t} - \mu_i(\mathbf{x})^2\right\|}{2}\right\}$$

This produces an automatic soft clustering of the target vectors into groups associated with the Gaussian components (Bishop 1995).

Jordan and Jacobs (1994) later extended their model. The extended variant forms a tree with mixture of experts models at the leaves. (See illustration below.)



They called their extended mixture model a *hierarchical mixture of experts* network. In theory, the nesting can be repeated for any number of levels.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

1.01 Multiple Choice Poll

Which of the following is the key limitation of the simple perceptron?

a. It fails when the problem has too many inputs.

b. It can solve only linearly separable problems.

c. Both a and b.

13

1.2 Traditional Approaches to Nonlinear Modeling

Objectives

- Illustrate two traditional approaches to modeling nonlinear input-output relationships:
  1. nonlinear regression
  2. nonparametric regression

17

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Nonlinear Regression

- Nonlinear regression models are more difficult to specify and to estimate than linear models.
- At a minimum, they require the following:
  - relatively accurate initial parameter estimates
  - the completely specified nonlinear equation
  - an optimization method to efficiently guide the parameter search process

18

Assuming that the functional form (i.e., equation) defining the nonlinear relationship between *y* and *x* is known *a priori*, and that only the parameters are unknown, then the parameters can be estimated using a technique known as *nonlinear regression* (Seber and Wild 1989).

If the nonlinear equation is not known, one option is to assume that the input-output relationship takes on some hypothesized functional form. A popular choice is a polynomial. A single input, *x*, polynomial of degree *d* is given by the equation:

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d = w_0 + \sum_{k=1}^{d} w_k x^k$$

One reason for the polynomial's popularity is the *Weierstrass approximation theorem*. The theorem asserts that any continuous-valued function on a real interval [a:b], can be approximated arbitrarily closely by a polynomial function. This means that a linear regression model using polynomials of sufficient complexity is actually a *universal approximator*.

Nonlinear regression models are more difficult to estimate than linear models. Not only must you specify the full nonlinear regression expression to be modeled, an optimization method must be used to efficiently guide the parameter search process. You must also provide initial parameter estimates. The value of these initial parameter estimates is critical. Starting at a bad location in the parameter space results in an inferior solution or, perhaps, even failure to achieve convergence at all.

In SAS, nonlinear regression models are constructed using the NLIN procedure. The NLIN procedure offers the following optimization options: gradient descent, Marquardt method, modified Gauss-Newton method (default), Newton's method, and steepest descent. The advantages and disadvantages of these optimization methods are discussed in Chapter 3.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Nonlinear Regression

c1s2d1

In this demonstration, you fit a nonlinear regression model to exponential decay data using PROC NLIN. The **DECAY** data set contains simulated data generated by adding noise (random variation) to the values generated by the function:

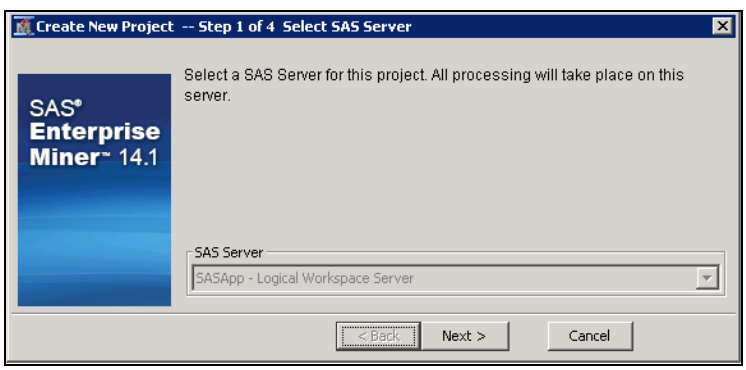
$$Ey = \vartheta_1\left(\left(\frac{\vartheta_3}{\vartheta_3-\vartheta_4}\right)\exp(-\vartheta_4 x)+\left(\frac{\vartheta_1}{\vartheta_2}-\frac{\vartheta_3}{\vartheta_3-\vartheta_4}\right)+\exp(-\vartheta_3 x)\right)^{-1}$$

The variables in the **DECAY** data set are listed below.

Variable	Description	Role	Level
EY	The (usually unknown) true y-coordinate	Rejected	Interval
X	The x-coordinate	Input	Interval
Y	The y-coordinate (i.e., the true target values with noise added)	Target	Interval

The objective is to determine appropriate values for the parameters ϑ_1,ϑ_2, ϑ_3, and ϑ_4.

1. Create a new project by selecting **New Project**. The Create New Project Wizard appears in step 1.

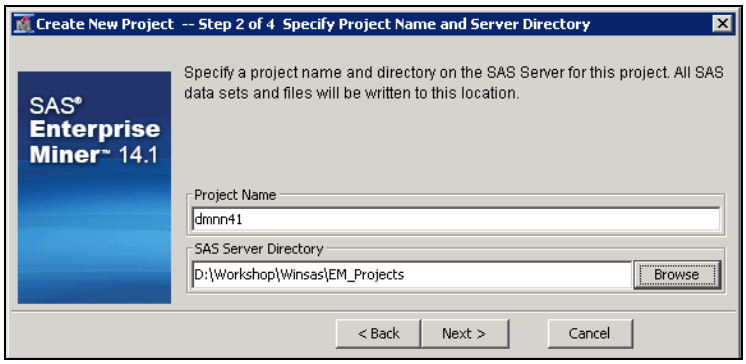


   In this configuration of SAS Enterprise Miner, the only available server for processing is the host server that is listed above.

2. Click **Next**.

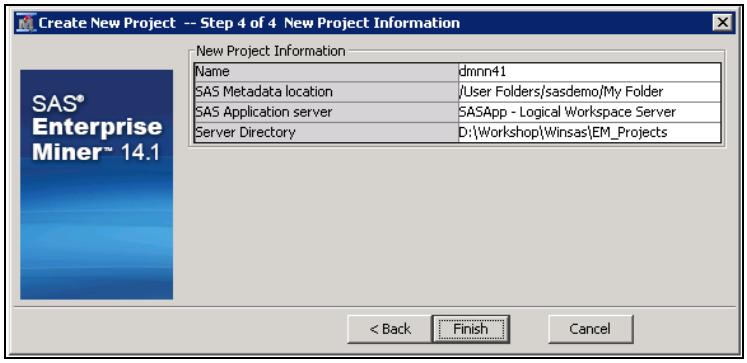
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3. Enter **dmnn41** in the **Project Name** field and **D:\Workshop\Winsas\EM_Projects** in the **SAS Server Directory** field. This creates the folder **dmnn41** in the SAS server directory.



4. Click **Next** to reveal the location for the project's metadata.

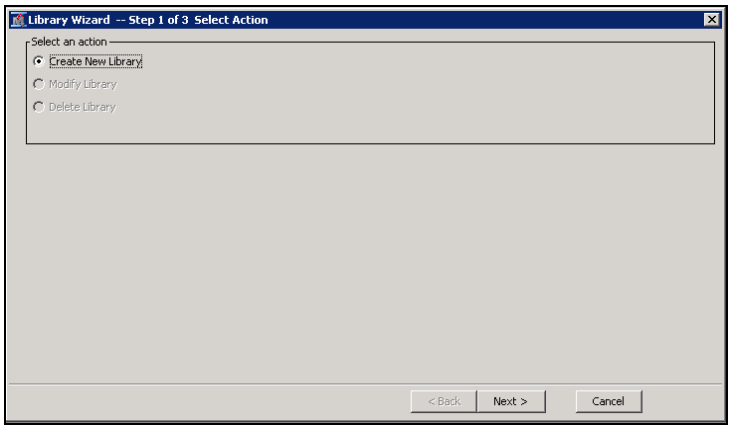
5. Click **Next** to reveal the new project information.



6. Click **Finish** to complete the project setup. The Enterprise Miner project interface appears.

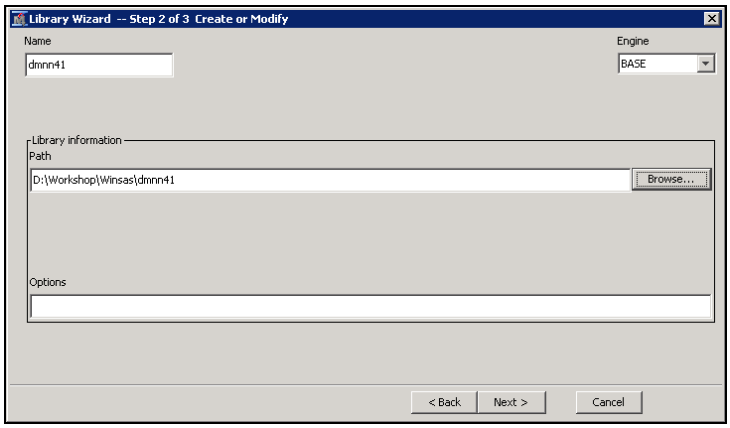
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

7. You need to assign the **dmnn41** library. Select **File** ⇨ **New** ⇨ **Library** to access the Library Wizard. Because there is no library to modify or delete, only the Create New Library option is available.

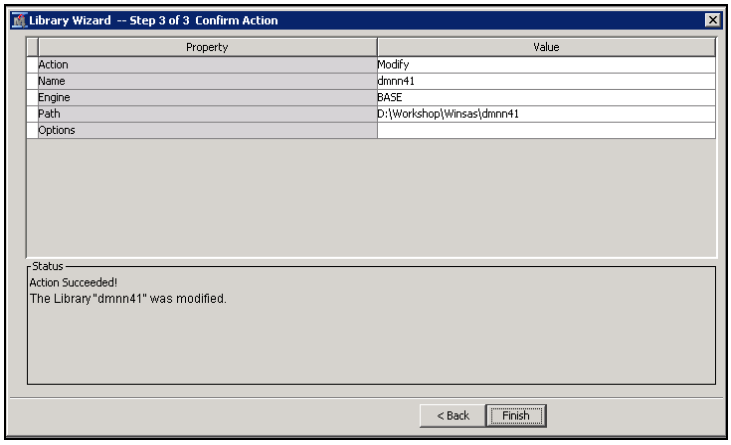


8. Click **Next** to begin creating the new library.

9. Enter **dmnn41** in the **Name** field, and type **D:\Workshop\Winsas\dmnn41** as the path.



10. Click **Next** to continue to the final step. The Status box shows that the library was created.

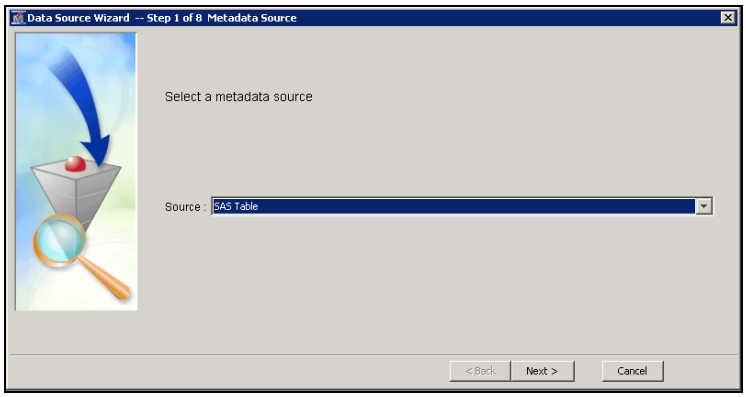


11. Click **Finish** to complete the library definition.

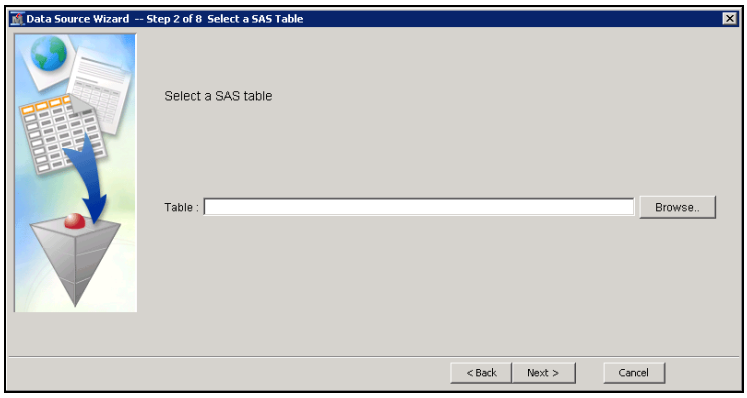
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

12. Right-click **Data Sources** in the project panel (upper left corner) and select **Create Data Source**.

This activates the Data Source Wizard.

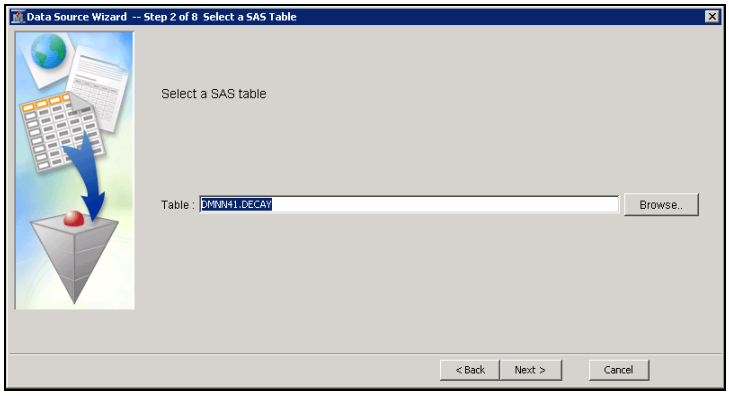


13. Click **Next** to progress to the second step of the metadata definition process.



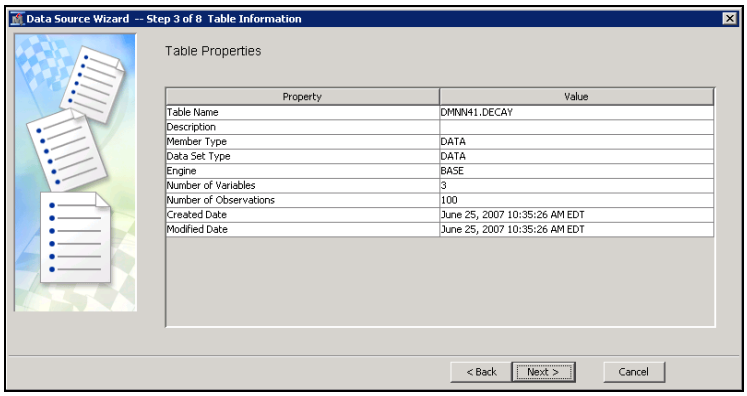
14. Click **Browse** to choose a SAS table from the available libraries. One of the options is the **dmnn41** library that you just defined. Click the **dmnn41** library.

15. Select the **DECAY** data table and click **OK** to continue.

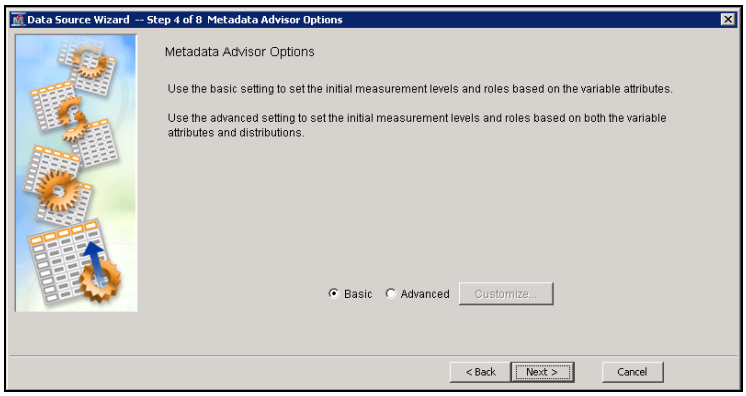


Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

16. Click **Next** to display information about the **DECAY** table, such as the number of variables (3) and the number of observations (100) it contains.



17. Again, click **Next** to begin modifying the metadata.

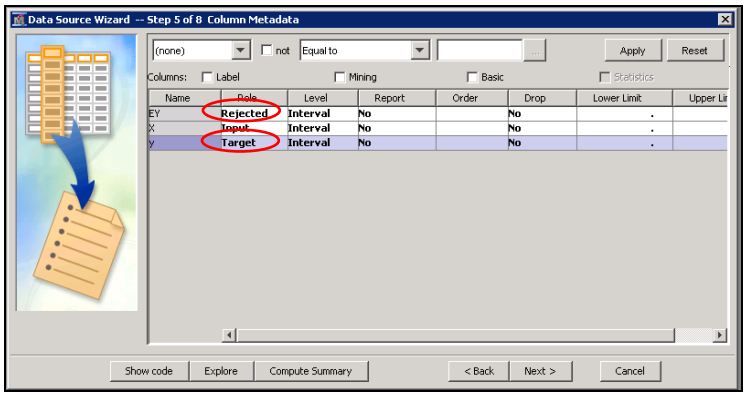


The available metadata advisor options are as follows:

- Basic — Use this option when you already know the variable roles and measurement levels. The initial role and level are based on the variable type and format values.
- Advanced — Use this option when you want SAS Enterprise Miner to automatically set the variable role and measurement level.

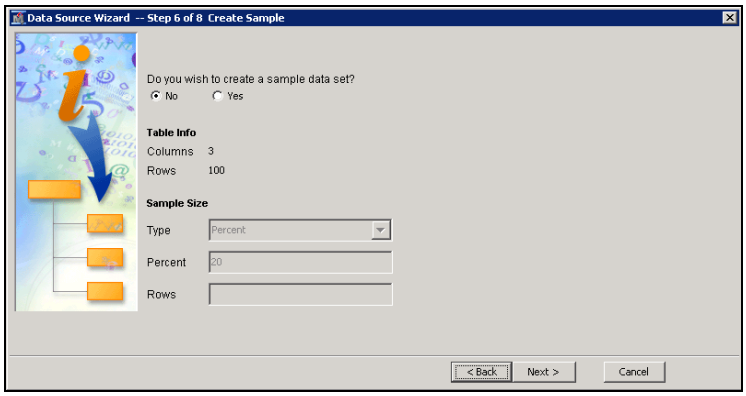
In this case, the (default) **Basic** setting suffices, so just click **Next** to proceed.

18. Change the role of **EY** to **Rejected** and the role of **Y** to **Target**, as shown below, and then click **Next**.



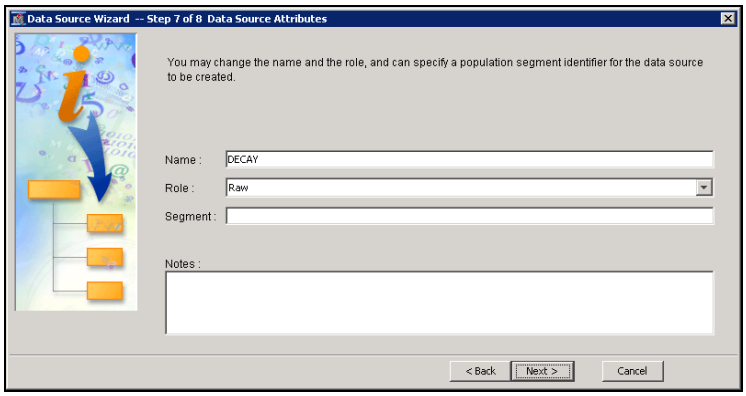
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

19. Click **Next**. This saves your variable role definitions and brings up the Create Sample menu.

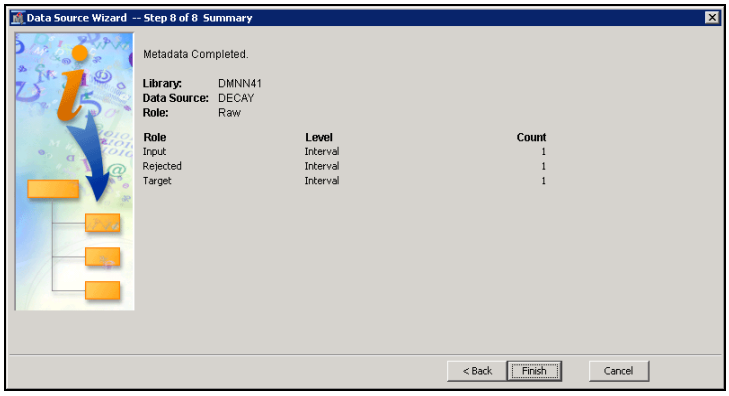


The Sample property indicates whether a sample of the active data should be performed at training time. Set this property to **Yes** to create a sample. The default No prevents the creation of a sample. In this case, no sample is required.

20. Click **Next** again. The data source name and role information are displayed.

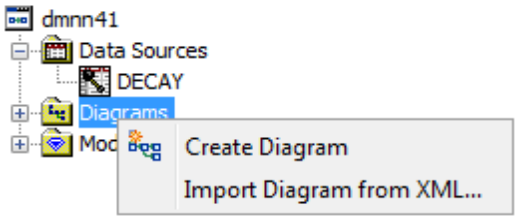


21. The default role of **Raw** is appropriate, so click **Next** and then **Finish** to complete the metadata definition process. This adds the **DECAY** data source to the **dmnn41** library.

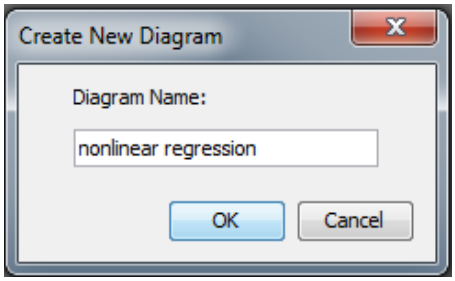


Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

22. Right-click **Diagrams** in the project panel and select **Create Diagram**.



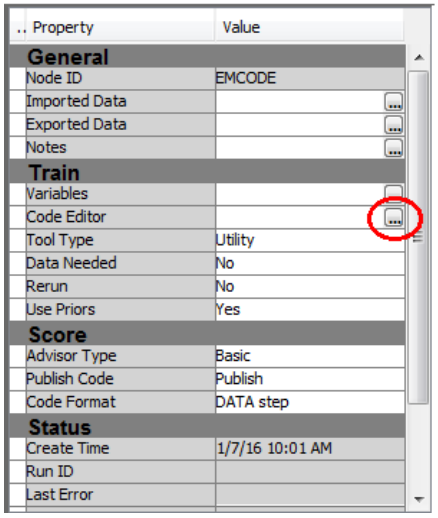
23. Enter the name **nonlinear regression** in the **Diagram Name** field, and click **OK**.



24. Drag and drop the **DECAY** data set onto the diagram.

25. Also, drop a **SAS Code** node (Utility tab) onto the diagram. Connect it to the **DECAY** data source. Your diagram should now look like the one shown below.

26. Activate the SAS Code node's properties panel and click the ellipsis (**…**) next to **Code Editor**.



27. Right-click in the white space of the Training Code editor, and select **Open** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

28. Select the program **c1s2d1.sas** (in d:\workshop\winsas\dmnn41). The program is described below.

First the global system and graphics options are initialized.

```
options nodate nonumber;
goptions reset=all device=actximg;
```

Then three symbols are defined.

```
symbol1 c=b v=none i=join;      /* true values */
symbol2 c=bl v=circle i=none;   /* observations */
symbol3 c=r v=none i=join l=3;  /* predicted values */
```

Next the NLIN procedure is called. The NLIN procedure requires that you provide initial estimates of the parameters. These estimates are specified in the PARMS statement. In this first call to the NLIN procedure, good initial estimates (that is, estimates near the correct values) are provided.

```
title1 h=1.5 f=swissb 'Exponential Decay Data';
title2 'Gauss-Newton Solution with Good Initial Estimates';
proc nlin data=&EM_IMPORT_DATA maxiter=500;
   parms theta1=100 theta2=2 theta3=.5 theta4=-.01;
   model y=theta1/((theta3/(theta3-theta4))*exp(-theta4*x)
         +(theta1/theta2-(theta3/(theta3-theta4)))*exp(-theta3*x));
   output p=p_y out=result;
run;
```

The equation that is to be fit is given by the MODEL statement. It is a literal transcription of the equation giving the true relationship between **x** and **y**. The OUTPUT statement writes the predicted value, **p_y**, into **RESULT**, which is then plotted.

```
proc gplot data=result;
   plot ey*x=1 y*x=2 p_y*x=3/overlay description="plot 1";
run;
```

Plots appear in alphabetical order of the description field in the GSEG catalog. To ensure that the plots will be displayed in the order, the DESCRIPTION= option is used.

In the second call to PROC NLIN, the MODEL and OUTPUT statements are identical to the first call. All that has changed are the initial estimates; now the initial estimates are not close to the final model.

```
title2 'Gauss-Newton Solution with Bad Initial Estimates';
proc nlin data=&EM_IMPORT_DATA maxiter=500;
   parms theta1=10 theta2=.2 theta3=.05 theta4=-.001;
   model y=theta1/((theta3/(theta3-theta4))*exp(-theta4*x)
         +(theta1/theta2-(theta3/(theta3-theta4)))*exp(-theta3*x));
   output p=p_y out=result;
run;
```

Again the true solution, the observations, and the proposed solution are plotted.

```
proc gplot data=result;
   plot ey*x=1 y*x=2 p_y*x=3/overlay description="plot 2";
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The last call to NLIN starts with the same bad initial estimates as before, only now the optimization method has been changed from the (default) Gauss-Newton method to Marquardt optimization. The Marquardt optimization method is less susceptible to bad initial estimates than Gauss-Newton.

```
title2 'Marquardt Solution with Bad Initial Estimates';
proc nlin data=&EM_IMPORT_DATA maxiter=500 method=marquardt;
   parms theta1=10 theta2=.2 theta3=.05 theta4=-.001;
   model y=theta1/((theta3/(theta3-theta4))*exp(-theta4*x)
         +(theta1/theta2-(theta3/(theta3-theta4)))*exp(-theta3*x));
   output p=p_y out=result;
run;
```

The proposed solution when Levenberg-Marquardt optimization was used is then plotted.

```
proc gplot data=result;
   plot ey*x=1 y*x=2 p_y*x=3/overlay description="plot 3";
run;
```

29. Click the running person icon (at the top of the window) to run the program from within the SAS Code node. Confirm the run, and select **Results** when the run is complete.

    The first item in the output listing is the Variable Summary, which displays the variable metadata.

```
Variable Summary

            Measurement    Frequency
Role           Level         Count

INPUT        INTERVAL          1
REJECTED     INTERVAL          1
TARGET       INTERVAL          1
```

30. Scroll down to line **24** to see the training history of the model with good initial parameter estimates.

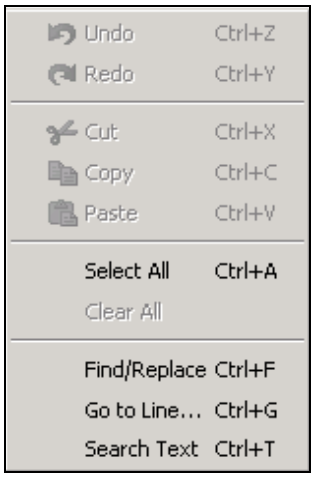
```
Gauss-Newton Solution with Good Initial Estimates

The NLIN Procedure
Dependent Variable y
Method: Gauss-Newton

                         Iterative Phase
                                                          Sum of
 Iter      theta1      theta2      theta3      theta4     Squares

    0       100.0      2.0000      0.5000     -0.0100      3866.4
    1       104.3      2.4780      0.4556     -0.0108      3668.3
    2       104.7      2.5454      0.4554     -0.0109      3654.8
    3       104.7      2.5397      0.4557     -0.0109      3654.8
    4       104.7      2.5394      0.4558     -0.0109      3654.8

NOTE: Convergence criterion met.
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Using Gauss-Newton optimization (the default method), the NLIN procedure only required four iterations to reach convergence. The derived values for the theta parameters are given in the last iteration of the training history. Substituting these derived theta values into the equation yields:

$$104.7\left(\left(\frac{0.4558}{0.4558+0.0109}\right)\exp(0.0109x)+\left(\frac{104.7}{2.5394}-\frac{0.4558}{0.4558+0.0109}\right)+\exp(-0.4558x)\right)^{-1}$$

31. Right-click anywhere in the Output window's white space. The following menu appears:



32. Select the **Go to Line** option and type **88** in the pop-up window. You jump to NLIN's results using the Gauss-Newton optimization, in this case when the initial parameter estimates are bad.

```
Gauss-Newton Solution with Bad Initial Estimates

The NLIN Procedure
Dependent Variable y
Method: Gauss-Newton

                         Iterative Phase
                                                          Sum of
 Iter      theta1      theta2      theta3      theta4     Squares

    0     10.0000      0.2000      0.0500    -0.00100      406918
    1     22.7982      0.5148      0.0215     -0.0112      406873
    2      -621.1      2.2301      0.0260      0.2348      367216
    3      -293.3      2.2299      0.0260      0.2966      367208
    4      -126.8      2.2296      0.0260      0.3652      367188
    5    -13.9309      2.2287      0.0260      0.4301      366844
    6    -10.3698      2.2263      0.0260      0.3848      366692
    7     -2.1944      2.2167      0.0260      0.4150      364687
    8     -0.8068      2.1806      0.0257      0.3883      359454
    9     -0.1959      2.0646      0.0251      0.4428      323865
   10     -0.1197      1.8264      0.0242      0.4924      304555
  ...         ...         ...         ...         ...         ...
  377    -9.34E-7      0.7801    8.177E-7      0.6905     19734.9
  378    -9.34E-7      0.7804    8.177E-7      0.6904     19734.9
  379    -9.34E-7      0.7803    8.177E-7      0.6904     19734.9

NOTE: Convergence criterion met but a note in the log indicates
a possible problem with the model.
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Not only did the algorithm take 379 iterations to converge on a solution, the note at the bottom of the listing suggests that there is a potential problem with the derived model. Moreover, the proposed solution is quite different that the estimates obtained when good starting estimates were provided.

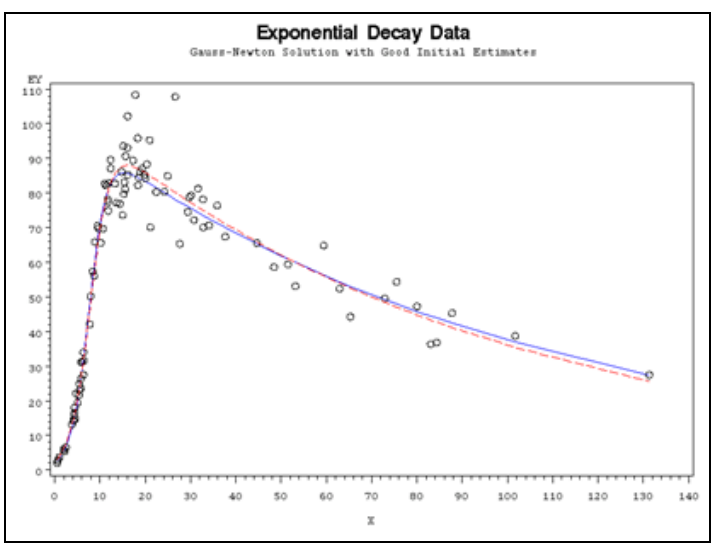
33. Again right-click in the Output window's white space, and select **Go to Line**.

34. This time type the line number **532** and scroll down to see the NLIN procedure's solution, given bad initial estimates, when Marquardt rather than Gauss-Newton optimization was used.

```
The NLIN Procedure
Dependent Variable y
Method: Marquardt

                         Iterative Phase
                                                          Sum of
 Iter      theta1      theta2      theta3      theta4     Squares

    0     10.0000      0.2000      0.0500    -0.00100      406918
    1      9.2876      3.4312      0.1225    -0.00290      339114
    2     58.5524      3.6753      2.0353     0.00807      119312
    3     55.5930      1.5580      1.0459     0.00551     64289.1
    4     58.9384      0.4730      0.7846     0.00298     38723.4
    5     80.7025      0.5959      0.7016    -0.00479     10198.2
    6     98.5834      1.4636      0.4480    -0.00969      9360.7
    7       105.0      2.8197      0.4153     -0.0110      4221.6
    8       104.7      2.6969      0.4462     -0.0109      3661.6
    9       104.7      2.5463      0.4550     -0.0109      3654.8
   10       104.7      2.5406      0.4557     -0.0109      3654.8
   11       104.7      2.5394      0.4558     -0.0109      3654.8
   12       104.7      2.5394      0.4558     -0.0109      3654.8

NOTE: Convergence criterion met.
```

Even though the initial estimates were bad, only 12 iterations were required to reach convergence. The derived parameters are identical to the solution generated from the good initial estimates.

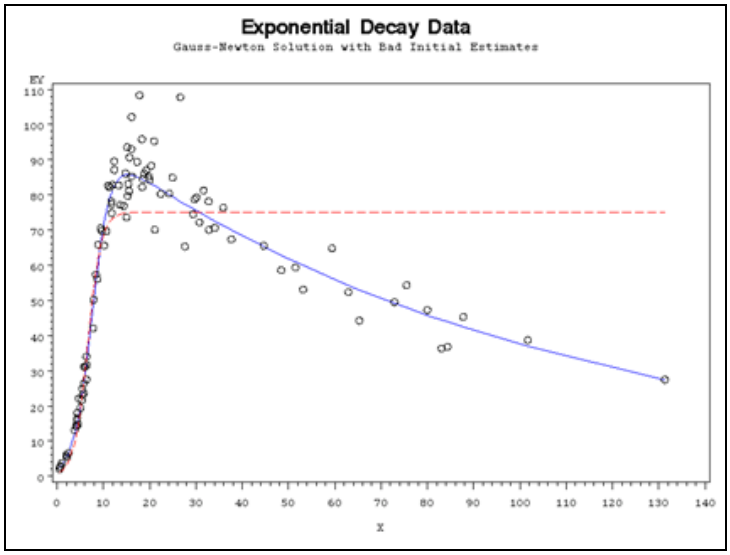
35. To see the scatter plots, select **View** ⇨ **SAS Results** ⇨ **Train Graphs**. The first plot shows the proposed solution given good initial parameter estimates.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

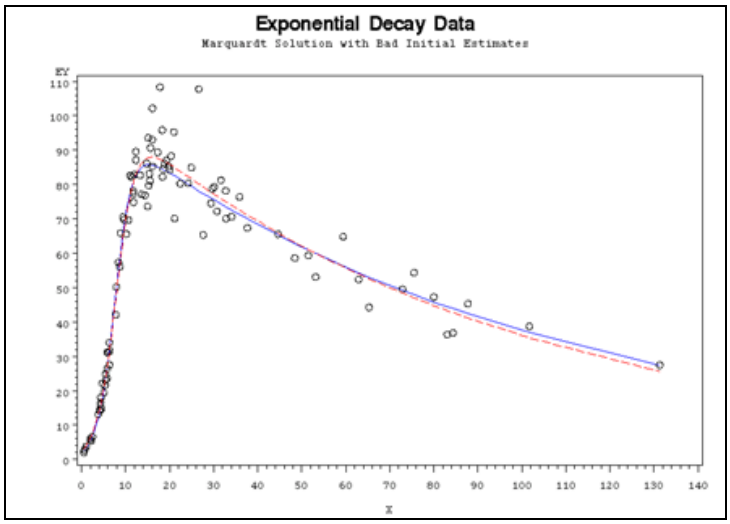
Given that both the nonlinear equation and good initial parameter values were known *a priori*, not surprisingly the NLIN procedure's final fit is a good one.

36. Click the right arrow at the bottom of the page (second icon from the right) to see the second plot.



In this case, the initial estimates were ***not*** good (i.e., not close to the final estimates). The model failed to capture the decay portion of the signal.

37. To see the final plot, click the right arrow again.



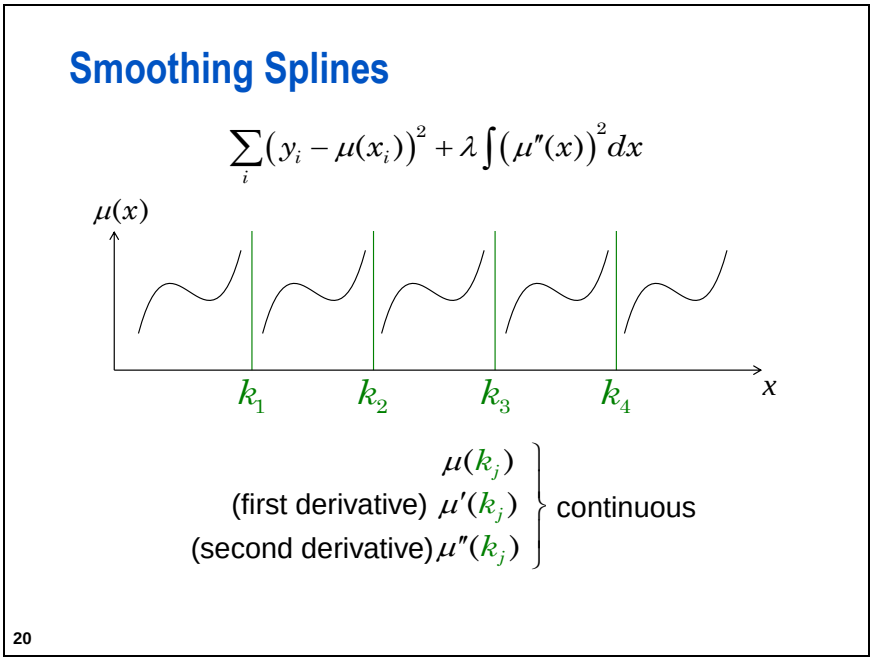
Although the initial estimates were bad, when Levenberg-Marquardt optimization was used the fit is now a good one. This illustrates that, in PROC NLIN, the optimization method specified is a crucial factor in determining the quality of the proposed solution. This is also true of neural networks.

38. Close the graph window, the Results window, and the SAS Code node.

39. Close the "nonlinear regression" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Nonparametric regression (Hastie and Tibshirani 1990, Fan and Gijbels 1996) is another popular method of modeling nonlinear predictive relationships. The method's popularity largely arises from the fact that it can fit nonlinear data without having to specify the functional form of the model.

First smoothing splines, made up of piecewise cubic polynomials, are generated. (See the diagram above.) The splines are then joined continuously and smoothly at a set of knots, k_j , to produce estimates along the entire input range. There is a knot at every data point. Despite the obvious over-parameterization that this implies, surprisingly a smoothing spline model does not necessarily over-fit data.

The parameters are estimated using the following penalized least squares equation:

$$\sum_i (y_i - \mu(x_i))^2 + \lambda \int (\mu''(k_j))^2 dx$$

The term on the left side of the plus sign performs a standard least squares fit. The right side is a penalty term that favors curves in which the average squared second derivative is low, discouraging a bumpy fit. As the value of the smoothing parameter increases ($0 \le \lambda \le 99$), the fit becomes less flexible.

Craven and Wahba (1979) suggest that cross validation is one way to find an appropriate lambda value. Cross validation is available in the TPSPLINE and LOESS procedures.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Nonparametric Regression

c1s2d2

This demonstration illustrates the use of smoothing spline nonparametric regression. Perhaps reflecting their popularity, smoothing splines can be fit using a variety of SAS procedures, including the GPLOT, MIXED, KDE, TPSPLINE, and TRANSREG procedures. In this case, the GPLOT procedure is used.

The **DECAY** data, described previously, is again used.

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter the name **nonparametric regression** in the **Diagram Name** field and click **OK**.
3. Drag and drop the **DECAY** data source onto the diagram.
4. Also add a **SAS Code** node (Utility tab). Connect it to the **DECAY** data source.

5. Click the ellipsis (**…**) next to **Code Editor** in the SAS Code node's properties panel.
6. Select the **Training Code** editor (the default Code Editor window).
7. Right-click in the white space and select **Open** from the pop-up menu.
8. Open the program **c1s2d2.sas** (in d:\workshop\winsas\dmnn41). The program is described below.

First, the system and graphics options are initialized. Then two symbols are defined.

```
options nonumber nodate;
goptions reset=all dev=actximg;

symbol1 c=bl v=circle i=none;
symbol2 c=b v=none i=join;
```

Next, several plots are requested.

```
title1 h=1.5 f=swissb 'Exponential Decay Data';
title2 'Smoothing Spline (Lambda=45)';
proc gplot data=&EM_IMPORT_DATA;
   symbol3 c=r v=none l=3 i=sm45;
   plot y*x=1 ey*x=2 y*x=3 / overlay description="plot 1";
run;
```

In the first plot, the I=SM45 option in the SYMBOL statement requests that a smoothing spline using an intermediate lambda value (λ = 45) be generated.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

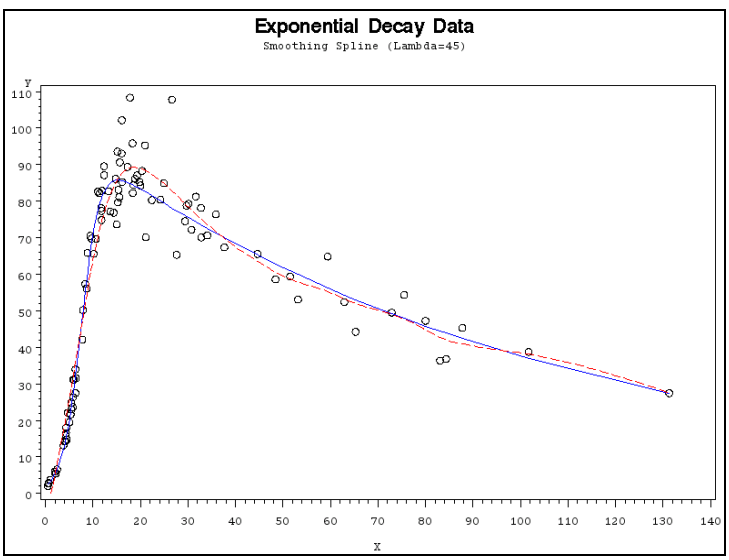
The second plot generates a smoothing spline using the smallest possible lambda value ($\lambda = 0$).

```
title2 'Smoothing (Lambda=0)';
proc gplot data=&EM_IMPORT_DATA;
   symbol3 c=r v=none l=3 i=sm0;
   plot y*x=1 ey*x=2 y*x=3 / overlay description="plot 2";
run;
```

The final plot generates a smoothing spline using the largest possible lambda value ($\lambda = 99$).

```
title2 'Smoothing Spline (Lambda=99)';
proc gplot data=&EM_IMPORT_DATA;
   symbol3 c=r v=none l=3 i=sm99;
   plot y*x=1 ey*x=2 y*x=3 / overlay description="plot 3";
run;
```

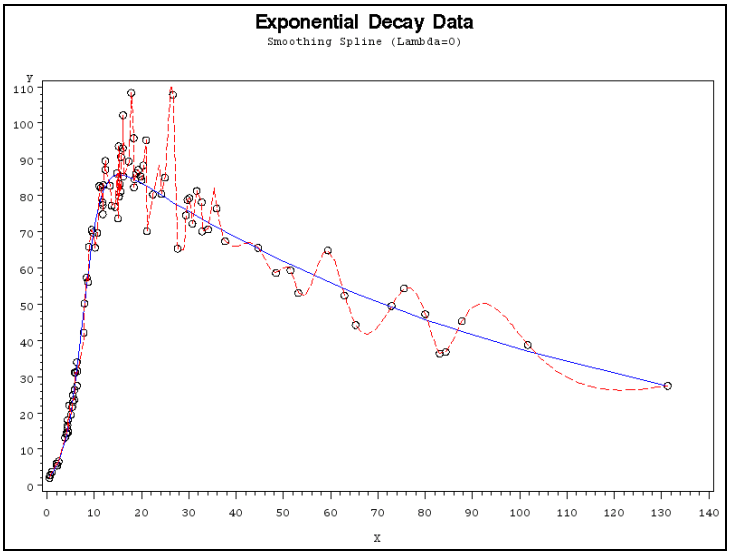
9. Click the running person icon at the top of the window to run the program.
10. Save the program, when you are asked, by clicking **Yes**.
11. Click **OK** when the program has finished.
12. Click the **Results** icon (the document icon to the right of the Run icon).
13. To display the generated graphs, select **View** ⇨ **SAS Results** ⇨ **Train Graphs** and maximize the plot. The first plot shows the model's prediction when lambda is at an intermediate value ($\lambda = 45$).



Although there is some deviation from the true model (the solid line), particularly at the apex, the intermediate lambda value results in a good estimate (the dashed line) of the truth. The waviness observed in the predicted line reflects the "stitching together" of the piecewise spline functions.

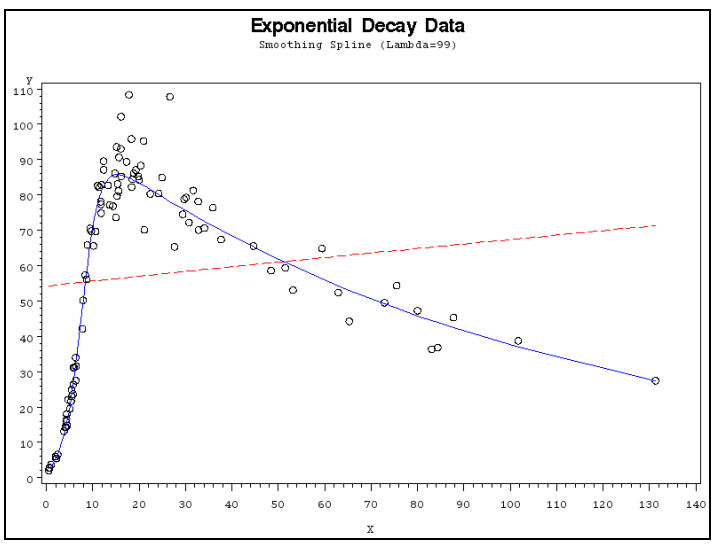
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

14. Click the right arrow button at the bottom of the window to see the prediction when the smoothing parameter is at its lowest value ($\lambda = 0$).



The penalty term is ignored altogether, so the fit is determined entirely by the least squares term. The fit that minimizes the sum of squared error value passes through each observation (no residuals).

15. Click the right arrow again to see the prediction when lambda is at its maximum ($\lambda = 99$).



Because the lambda term has 99 times the influence of the least squares component, the best fit is given by the "curve" that maximizes smoothness, i.e., a straight line.

16. Close the Results windows and the SAS Code node.

17. Close the "nonparametric regression" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Limitations of Traditional Approaches

- Traditional nonlinear methods are limited with respect to the number of inputs that they can consider.
- Nonlinear models can fail because it is very difficult to specify their functional form (equation).
- Nonparametric regression models can fail because of the relative sparseness of data in higher dimensions.
- Neural networks, which require no specified form, often work well in sparse, high-dimensional spaces.

22

Traditional nonlinear modeling techniques become vastly more difficult as the number of inputs increase. For example, it is uncommon to see parametric nonlinear regression models with more than a few inputs, as a suitable parametric function is usually not derivable. And smoothing splines and other nonparametric regression methods suffer because of the relative sparseness of data in higher dimensions.

In contrast, neural networks are also flexible multivariate function estimators that generally perform well in sparse, high-dimensional spaces. Moreover, it is not necessary to specify their functional form.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

1.3 Advantages and Disadvantages of Neural Networks

Objectives

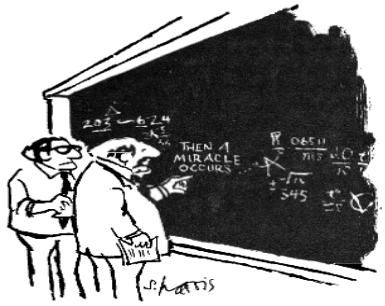
- Outline a few key advantages and disadvantages offered by neural networks.

25

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

No Functional Form

- The function describing the input-output relationship does not need to be specified ... or even fully understood.

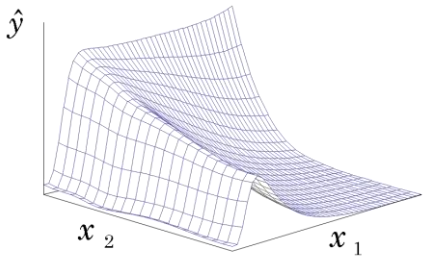


"I think you should be more explicit here in step two."

26

Although neural networks are parametric nonlinear regression models, they behave like nonparametric regression (smoothing splines), in that it is not necessary to specify the functional form of the model. This allows construction of models when the relationship between the inputs and outputs is unknown.

Universal Approximation



- Given enough neurons and time, a neural network can model any I/O relationship, to any degree of precision.

27

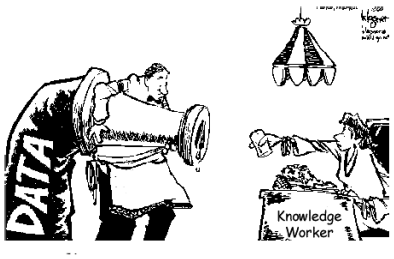
The chief benefit of artificial neural networks is their unlimited flexibility. As was noted previously, a neural network is a *universal approximator*, which means that, with a sufficient number of hidden units and enough time, a neural network can model any input-output relationship, no matter how complex.

The question of how many layers and, in particular, how many neurons are needed in each layer for a given modeling task is a very difficult one to answer. This issue is discussed in Chapter 2.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Speed

- Neural networks are one of the fastest scoring models.
- They are well suited to filtering large volumes of data.



28

To list speed as one of the advantages of a neural network might surprise some people, particularly in light of the much-publicized slow training times of early neural networks. After it is trained, however, a neural network is among the fastest executing predictive models. This means that a trained neural network can efficiently score large volumes of data. For example, neural networks are used by banks to assess all of their credit card transactions for potential fraud patterns. Neural networks also are commonly used to provide real-time evaluation and monitoring of stock and futures movements.

Lack of Interpretability

- This is the famous *black-box objection*, often raised merely to disparage neural networks.
- There are two ways to respond to this objection:
  1. by admitting that neural networks are most relevant to pure prediction tasks
  2. by applying other modeling techniques, such as decision trees, to try to help “open” the black box

29

This is the famous *black box* objection to neural networks. Suggestive of mystery, the term “black box” is often used simply to disparage neural networks by implying that they can never be interpreted.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

There are two ways to respond to this criticism.

1. The first response is to recognize that, in many tasks, pure prediction is the goal. Understanding how the inputs affect the prediction is of secondary importance. In other applications, the opposite is true: predictive power is a consideration only to the extent that it validates the interpretive power of the model. Neural networks are most appropriate for pure prediction tasks.

2. The second response is to show ways in which the black box has, at least partially, been "opened". Various approaches are considered in Chapter 4, including direct weight examination through Hinton diagrams, and input sensitivity assessment. A particularly interesting method of opening the black box is *decomposition* (Tsukimoto 2000), which contends that the weights in a neural network can be approximated by a set of IF-THEN rules. A related approach is to use a decision tree to interpret the neural network's predictions. This option is explored in Chapter 5. (See Surrogate Models.)

Disenchantment

"It is shown that, at least for the data used in this study, the fit achieved [by regression] is approximately the same, but the process of configuring and setting up a neural network for a database marketing application is not straightforward, and may require extensive experimentation and computer resources."

Zahavi and Levin. 1997. "Applying Neural Computing to Target Marketing." *Journal of Direct Marketing*.

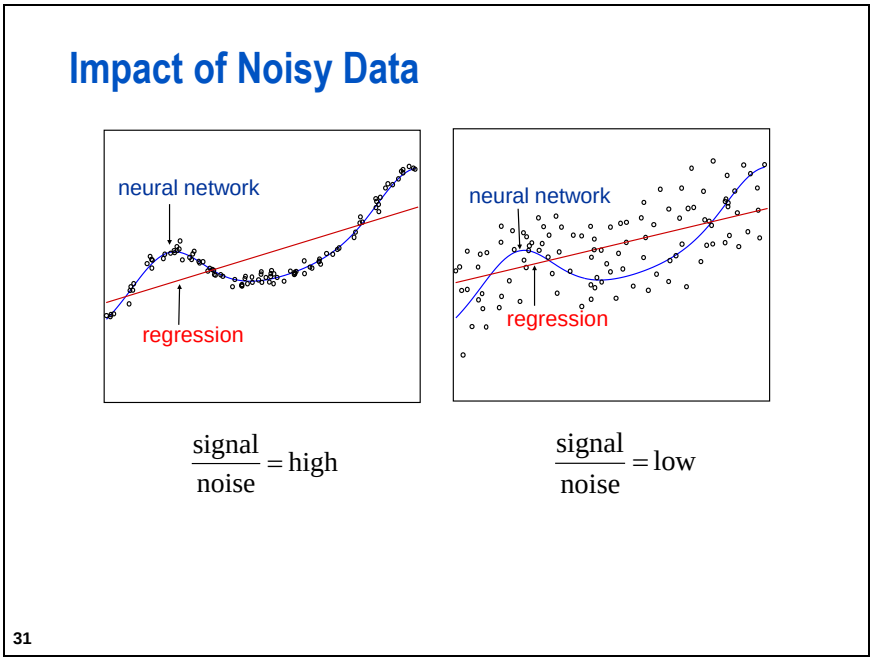
30

Sometimes neural networks do not outperform simpler and easier to implement models like regression. This has led to disenchantment with the more complex neural networks. A possible explanation for this failure was suggested by David Shepard Associates (1999) in *The New Direct Marketing*:

"…if marketers think they can blindly use [neural networks] without the aid of an experienced statistician or an AI expert, they are making, in our opinion, a very serious mistake …"

Not only is there a wide array of neural network architectures available today, the staggering number of options within any given architecture makes successful model construction less likely if the modeler is unfamiliar with the theoretical and practical implications of each option.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Another possible explanation for this disenchantment with the relative performance of neural networks in some situations, is found in the signal-to-noise ratio.

To illustrate, in the left panel of the above diagram, the nonlinear model that is produced by the nonlinear neural network would clearly produce a superior fit to the data, as compared to the fit produced by the linear regression model. There is a strong pattern (signal) relative to the amount of variation (noise) around the pattern; the signal-to-noise ratio is high.

The situation in the right panel is different; the signal-to-noise ratio is low. The regression and neural network models will likely offer a comparable fit to the data. Ockham's razor would, therefore, imply that, in this case, there would be no advantage to using the more complex neural network.

Ockham's razor states that "entities must not be multiplied beyond necessity." Thus, if competing hypotheses are equal in other respects, Ockham's razor recommends choosing the hypothesis with the fewest postulates. In short, the simplest hypothesis is usually the correct one.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Two Spirals Problem

c1s3d1

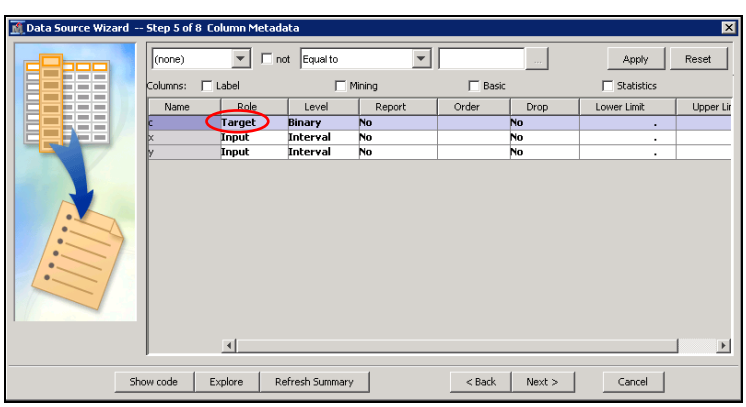
To illustrate the nonlinear modeling capability of a neural network when the signal-to-noise ratio is high, a classic benchmark test of model performance known as *the two spirals problem* is explored.

The variables in the **SPIRALS** data set are listed below.

Variable	Description	Role	Level
C	The spiral class of the member (0 or 1).	Target	Binary
X	The x-coordinate.	Input	Interval
Y	The y-coordinate.	Input	Interval

The task is to separate two predefined classes, using only the *x* and *y* coordinates of the observations as input. The problem is that the classes are intertwined into two concentric circles!

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter the name **two spirals** in the **Diagram Name** field and click **OK**.
3. Right-click **Data Sources** in the project panel and select **Create Data Source**.
4. Click **Next** to create a SAS data table.
5. Click **Browse** and select the **SPIRALS** data in the **dmnn41** library. Click **OK** to continue.
6. Click **Next** to display information about the **SPIRALS** data set. For example, there are three variables and 194 observations in the table.
7. Click **Next** to start the metadata definition process.
8. Select the **Advanced** radio button, in this case just to set the level of variable **C**, which identifies the members of each spiral, to **Binary**. Click **Next** to proceed.
9. Change the role of the variable **C** to **Target**. No other changes need be made.



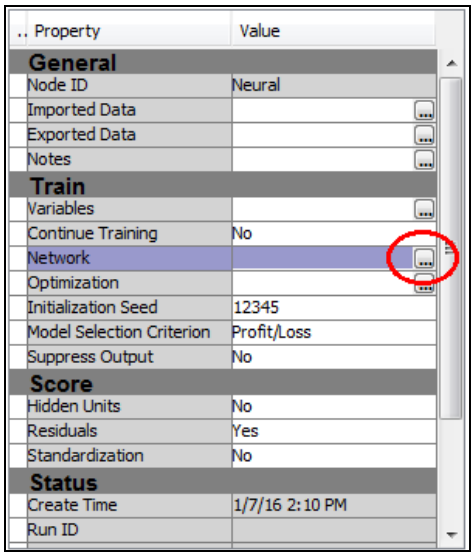
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

10. Click **Next** four times, and then click **Finish** to complete the metadata definition.

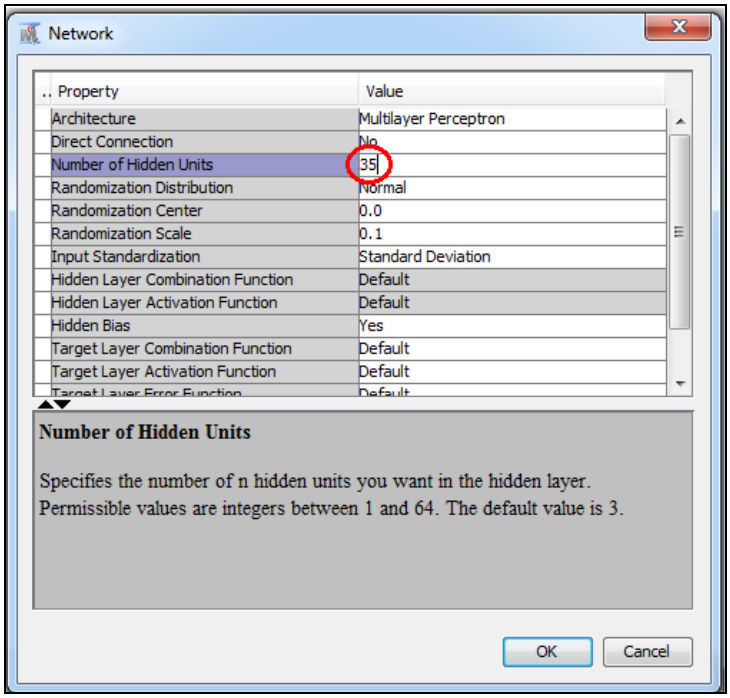
11. Drag and drop the newly created **SPIRALS** data source onto the diagram.

12. Click the **Model** tab and add a **Neural Network** node (fifth icon from the right) to the diagram. Connect it to the **SPIRALS** data source.

13. Activate the Neural Network node's properties panel, and click the ellipsis (**…**) next to the **Network** field (in the Train section of the properties panel).



14. Change the **Number of Hidden Units** property to **35**. This constructs a single hidden layer multilayer perceptron network, the default architecture, consisting of 35 hidden units.



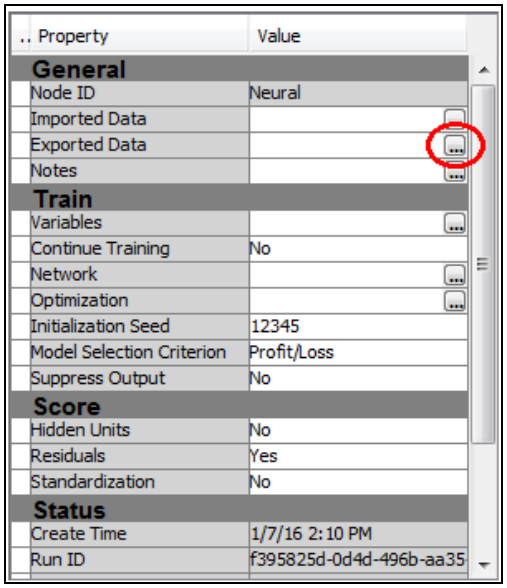
15. No other changes need to be made, so click **OK** to close the Network definition window.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

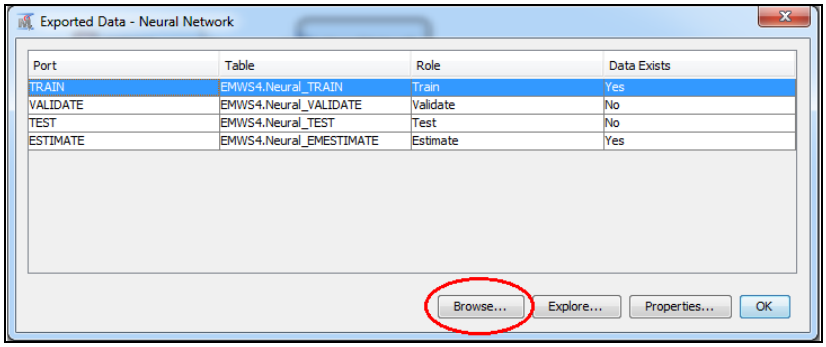
16. Run the diagram from the Neural Network node by right-clicking the node and selecting **Run**. Confirm the run when asked.

17. Click **OK** (not Results) when training is complete.

18. In the Neural Network node's properties panel, click the ellipsis (**...**) next to **Exported Data** (in the General section) to open the data set output by the neural network.



19. Highlight **TRAIN** (top row) and click **Browse** to open a sample of the output training data set.



Two of the output fields are **From: c** and **Into: c**. They represent, respectively, the actual class of the observation (which is the same as **c**), and the predicted classification. These variables will be used to generate a classification matrix that can then be used to assess the neural network's accuracy.

20. Close the Browse and Exported Data windows, and return to the diagram.

21. Add a **SAS Code** node (Utility tab) to the flow. Connect it to the **Neural Network** node. (See below.)

22. Click the ellipsis (**…**) next to the **Code Editor** field in the SAS Code node's properties panel and select the **Training Code** editor.

23. Right-click in the white space and select **Open** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

24. Open the program **c1s3d1.sas** (in d:\workshop\winsas\dmnn71). The program is described below.

First, the system and graphics options are initialized, and two global symbols are defined. The SYMBOL1 statement produces red squares, and the SYMBOL2 statement generates black triangles.

```
options nodate nonumber;
goptions reset=all device=actximg;

symbol1 c=red v=square;
symbol2 c=black v=triangle;
```

Next, the association between the actual classification of each observation, **F_c**, and the observations' predicted class, **I_c**, is displayed in the form of a classification (or confusion) matrix.

```
title1 h=1.5 f=swissb 'Two Spirals Problem';
title2 'Classification Matrix';
proc freq data=&EM_IMPORT_DATA;
   tables F_c*I_c;
run;
```

The predicted class membership is then plotted.

```
title2 'Plot of Predicted Membership';
proc gplot data=&EM_IMPORT_DATA;
   plot y*x=I_c;
run;
```

25. Run the program from within the SAS Code node by clicking the running man. Confirm the run.

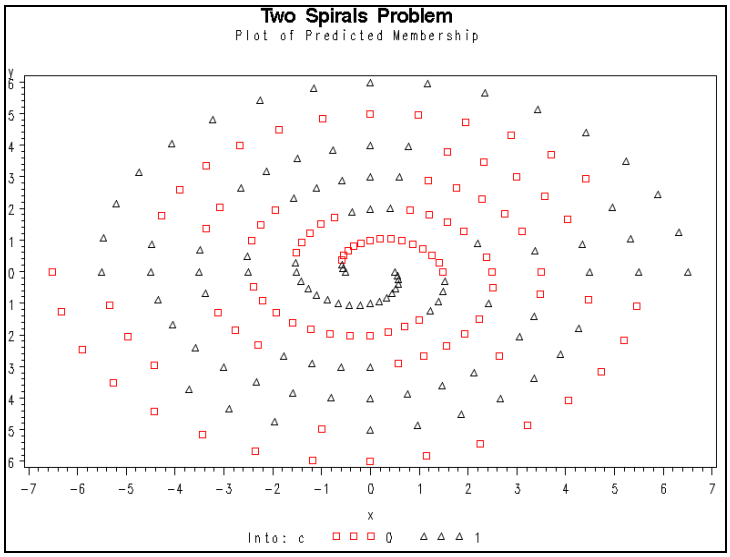
26. Click **Results** when the program is finished, and scroll to line **29** to see the classification matrix.

```
The FREQ Procedure

Table of F_c by I_c

F_c(From: c)     I_c(Into: c)

Frequency|
Percent  |
Row Pct  |
Col Pct  |0       |1       |  Total
---------+--------+--------+
0        |     64 |     33 |     97
         |  32.99 |  17.01 |  50.00
         |  65.98 |  34.02 |
         |  66.67 |  33.67 |
---------+--------+--------+
1        |     32 |     65 |     97
         |  16.49 |  33.51 |  50.00
         |  32.99 |  67.01 |
         |  33.33 |  66.33 |
---------+--------+--------+
Total          96       98      194
            49.48    50.52   100.00
```

As you can see, the neural network's classification performance was <u>not</u> a particularly good one.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

27. To see the proposed solution, select **View** ⇨ **SAS Results** ⇨ **Train Graphs**. Maximize the plot.

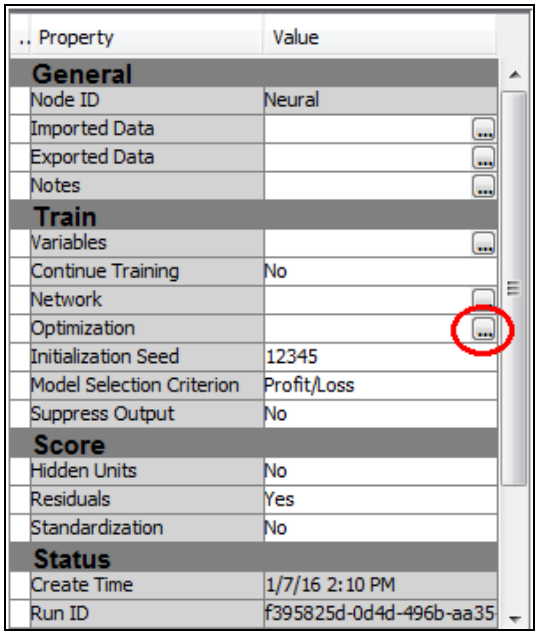


As is evident, the neural network clearly failed to discriminate between the classes. What happened?

The answer is that the Neural Network node was simply not allowed enough passes through the data to allow it to generate a better solution.

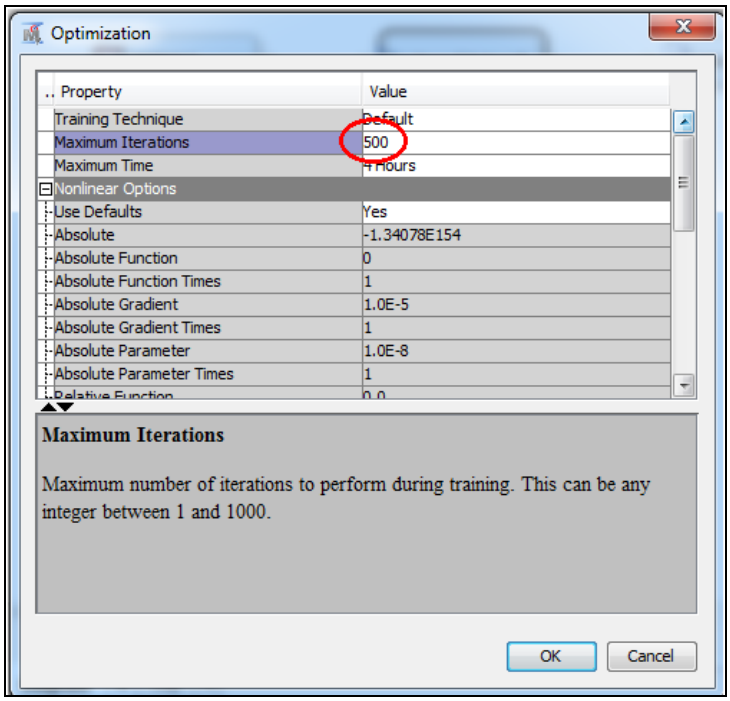
28. Close the SAS Code node.

29. Click the **Neural Network** node to again activate its properties panel. Then click the ellipsis (**...**) next to the **Optimization** field (in the Train section).



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

30. Enter **500** for the **Maximum Iterations** property. Click **OK** to save the change.



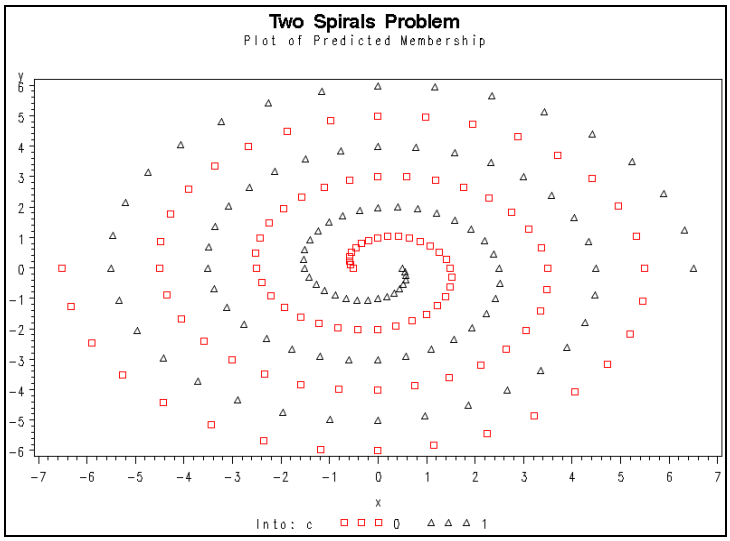
31. Re-run the diagram from the SAS Code node, and examine the results when training is completed.

32. Go to line **29**. The neural network has now recovered the original class membership.

```
The FREQ Procedure

Table of F_c by I_c

F_c(From: c)     I_c(Into: c)

Frequency|
Percent  |
Row Pct  |
Col Pct  |0       |1       |  Total
---------+--------+--------+
0        |     97 |      0 |     97
         |  50.00 |   0.00 |  50.00
         | 100.00 |   0.00 |
         | 100.00 |   0.00 |
---------+--------+--------+
1        |      0 |     97 |     97
         |   0.00 |  50.00 |  50.00
         |   0.00 | 100.00 |
         |   0.00 | 100.00 |
---------+--------+--------+
Total          97       97      194
            50.00    50.00   100.00
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

33. To see the proposed solution, select **View** ⇨ **SAS Results** ⇨ **Train Graphs**. Maximize the plot. The scatter plot shows that the neural network now perfectly separates the two interwoven spirals.



34. Close the Results window and the SAS Code node.

35. Close the "two spirals" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

1.02 Poll

Even after training is completed, neural networks are usually slow to generate their estimates/decisions.

- ❍ True
- ❍ False

33

1.4 Chapter Summary

Although inspired by biology, neural networks largely draw their methods from statistical physics. There are many different neural network algorithms. They fall into two broad classes:

- unsupervised learning algorithms, which have no target (dependent) variable
- supervised learning algorithms, which are guided by the provided target variables

This course focuses on two supervised learning models: the multilayer perceptron algorithm and the less well known, but (arguably) even more powerful, radial basis function algorithm.

Neural networks are not new. They were first formalized in 1943, in a paper by McCulloch and Pitts. Using predicate calculus, McCulloch and Pitts proved that networks of binary neurons are universal approximators, able to model any given input-output relationship, to any degree of precision. The weights in a McCulloch-Pitts network, which represent the adaptive connection strengths between the neurons, had to be set manually. There was no known, biologically plausible, learning rule. In 1949, Hebb devised the first plausible learning rule, formalized by the following equation:

$$\Delta w_{ij} = \eta \hat{y}_i x_j$$

Unfortunately, the simple Hebb rule is unstable. In 1960, Widrow and Hoff came up with an important variant of the Hebb rule that stabilized the weights for a wide range of learning rate (η) values. The delta rule, as this learning rule was known, is given by the following equation:

$$\Delta w_{ij} = \eta(y_i - \hat{y}_i)x_j$$

The Widrow-Hoff delta rule performs least mean squares parameter estimation, which is why it is also commonly known as the least mean squares (LMS) rule.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The delta rule (actually its mathematical equivalent) was first implemented in a computer model by Frank Rosenblatt in 1959. Rosenblatt called his computer implementation a perceptron

> As an analogue to the biological brain, the perceptron, more precisely, the theory of statistical separability seems to come closer to meeting the requirements of a functional explanation of the nervous system than any system previously proposed. … The future of information processing devices that operate on statistical, rather than logical, principles seems to be clearly indicated.

His comparison to logical models was not accidental; the McCulloch-Pitts paper had also given birth to the field of artificial intelligence (AI). From the AI perspective, a neuron implements an IF…THEN rule, and networks of neurons combine to form the logical predicates used to guide decision making.

Rosenblatt's dismissal of the fledgling field angered many in the AI community. One person who was particularly upset was Marvin Minsky. An early neural network pioneer himself, Minsky knew that perceptrons worked well on some problems, but did not work on others. He decided to find out why. In collaboration with his graduate student, Seymour Papert, Minsky (1969) analyzed the pattern processing capabilities of the perceptron. Minsky and Papert discovered that, if the patterns to be discriminated were not linearly separable, the perceptron would invariably fail.

The Minsky-Papert analysis so severely impacted the early artificial neural network field, that it would be more than a decade before many people started taking neural networks seriously again!

The Minsky and Papert analysis only applied to linear perceptrons. Rosenblatt had also experimented with multiple-layer perceptron networks in which the output of one set or layer of neurons fed directly into a second (hidden) layer of neurons before being passed to the output layer. Because a multilayer perceptron can generate nonlinear relationships, it completely overcomes the limitations described by Minsky and Papert. What Rosenblatt did not know was how the hidden layer weights could automatically be determined. The problem was that the delta rule needs to know both the input and output activations. But the hidden neurons did not provide the output value before it was transformed by the discontinuous step function. So the delta rule could not be applied.

The algorithm that eventually solved this problem is the generalized delta rule (i.e., back propagation). It solved the problem by replacing the hidden layer's discontinuous step function with a continuous sigmoid function that could be mathematically "undone." This meant that the hidden unit output values could be recovered, allowing the delta rule to be applied to the hidden units.

But the Minsky-Papert objection is not the only criticism of neural networks. Another criticism is the black box objection. Suggestive of mystery, the term black box is often just used to disparage neural networks by implying that they can never be interpreted.

Although it is true that neural networks do not easily lend themselves to interpretation, neural networks are not alone in being opaque. The same criticism can be levied against nearly any sophisticated model (e.g. factor analysis). Moreover, new ways of "opening" the black box are continuing to emerge. And for many tasks (for example, fraud detection), pure prediction is the goal. Interpretation is of secondary or no importance. Neural networks are most appropriate for pure prediction tasks.

Another criticism is that neural networks do not always outperform simpler, easier to implement, methods like linear regression. One possible explanation for this is the vast array of neural network architectures, and the staggering number of options within any given architecture. If the modeler is unfamiliar with the practical and theoretical implications of each option, it makes success much less likely. Alternatively, the signal-to-noise ratio might be so low that nearly any modeling method would provide the same quality of fit to the data. But if the signal-to-noise ratio is high, and the relationship is nonlinear, a neural network's universal approximation capability and the fact that there is no need to specify the functional form of the relationship, makes a neural network hard to beat.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

1.5 Solutions

Solutions to Student Activities (Polls and Quizzes)

1.01 Multiple Choice Poll – Correct Answer

Which of the following is the key limitation of the simple perceptron?

a. It fails when the problem has too many inputs.

b. It can solve only linearly separable problems.

c. Both a and b.

14

1.02 Poll – Correct Answer

Even after training is completed, neural networks are usually slow to generate their estimates/decisions.

❍ True

❍ False

34

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Chapter 2 Network Architecture

2.1 Linear Perceptrons **2-3**

Demonstration: Modeling Assorted Target Distributions 2-8

2.2 Multilayer Perceptrons **2-19**

Demonstration: Compositional Data 2-26

Exercises 2-38

2.3 Radial Basis Functions **2-40**

Demonstration: Plotting Basis Functions 2-49

Exercises 2-58

2.4 Chapter Summary **2-62**

2.5 Solutions **2-63**

Solutions to Exercises 2-63

Solutions to Student Activities (Polls and Quizzes) 2-65

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.1 Linear Perceptrons

Objectives

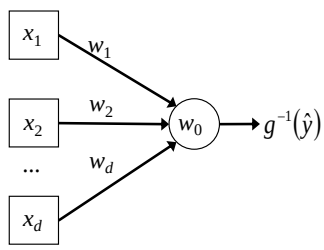
- Define the linear perceptron neural network.
- Describe combination and activation functions.
- Show how a linear perceptron is a generalized linear model that is able to model many target distributions.

3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Linear Perceptron (Generalized Linear Model)

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{d} w_i x_i$$



4

A McCulloch-Pitts neuron can easily be extended by replacing its discontinuous step function with a continuous output transformation known as an *activation function*, $g(.)$.

$$\hat{y} = g(w_0 + \sum_{i=1}^{d} w_i x_i)$$

The equation can equivalently be expressed as a *link function,* $g^{-1}(.)$. A link function is the mathematical inverse (like log and exponent or square and square root) of an activation function.

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

When used in conjunction with an appropriate *error function* (discussed in Chapter 3), these activation functions turn a McCulloch-Pitts neuron into a generalized linear model, capable of fitting a number of target types, including normal, gamma, Poisson, binary, multinomial, and ordinal targets.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Combination Functions

The following combination functions are available:

- Linear — sums the weighted inputs (default).
- Additive — sums the inputs without weighting.
- Equal Slopes — sums the weighted inputs using the same weight. The biases can differ.

5

Just as in a biological neuron, the first step in neural activation is to collect the arriving inputs. This is accomplished through a *combination function*. There are three combination functions:

Linear The neuron first weights the inputs before summing.

$$\sum_{i=1}^{d} w_i x_i$$

This is appropriate in a wide range of neural networks, including the perceptron and radial basis function architectures. **It is the default option.**

Additive The neuron adds the inputs together without weighting them.

$$\sum_{i=1}^{d} x_i$$

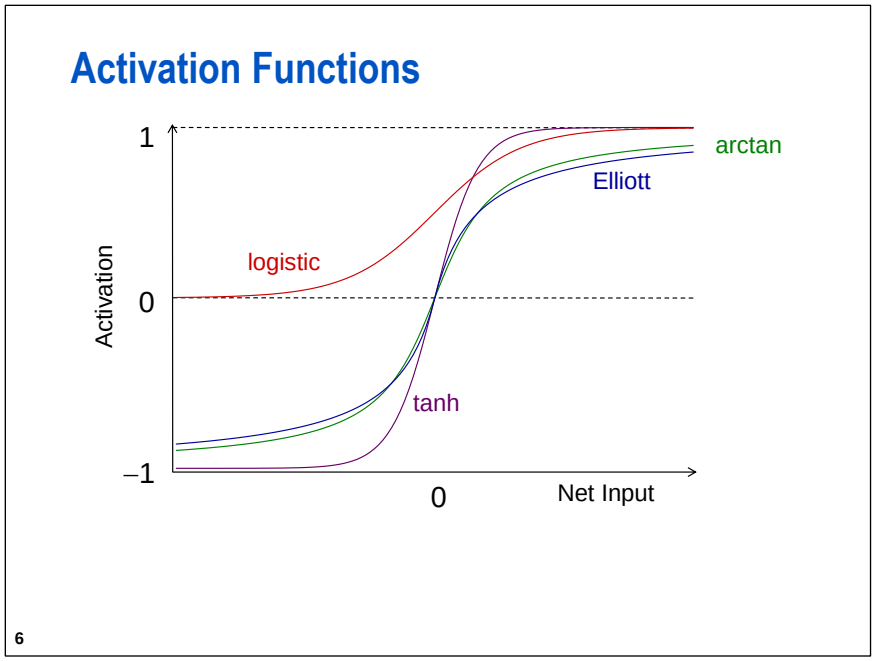
Although rarely used, the additive combination function is appropriate when collecting inputs that have already been transformed.

Equal Slopes The neuron uses the same weight value for each input, although the biases can differ.

$$\sum_{i=1}^{d} w x_i$$

The equal slopes option is appropriate when modeling ordinal targets. However, due to the nonlinearity introduced by the hidden layer, **this option offers no advantage when modeling an ordinal target with a multilayer network**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



After the weighted inputs and the bias have been combined, the neuron's net input is passed through an activation function. For the perceptron, the defining hidden unit activation functions are all members of the sigmoid family. The most famous member of the sigmoid family is the logistic function:

$$\text{logistic}(net) = \frac{1}{1+e^{-net}} = \hat{p}\text{, where } net = w_0 + \sum_{i=1}^{d} w_i x_i$$

The logistic activation function constrains its output to the range 0:1, making it an ideal for generating probability ($\hat{p}$) estimates. In statistics, the logistic function is better known as the *logit-link* function:

$$\text{logit}\ (\hat{p}) = \ln(\frac{\hat{p}}{1-\hat{p}}) = \ln(odds)$$

Although the logistic activation function was used in early neural network research, many other sigmoidal activation functions exist. One that plays a key role in the Neural Network node is the *hyperbolic tangent* (tanh). In fact, **it is the default hidden unit activation function**.

$$\tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

The hyperbolic tangent ranges from -1 to 1. This means that the inflection point is at 0 rather than at 0.5, as it is in the logistic sigmoid. This offers a small advantage during network initialization.

Two other sigmoid functions that also have a -1:1 range are the *Elliott* and *arc tangent* functions:

$$\text{elliott}(net) = \frac{net}{1+|net|} \qquad \arctan(net) = \frac{2}{\pi}\tan^{-1}(net)$$

Interestingly, **with respect to the hidden layer neurons, it does not seem to matter which of the sigmoid activation functions is used.** The logistic, hyperbolic tangent, Elliot, and arc tangent activation functions all perform more or less equivalently.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Several useful non-sigmoidal activation functions are also available. For example, the *exponential* activation function generates values that range from 0 to ∞. This is particularly useful when fitting distributions that are undefined for negative input values (e.g., Poisson or gamma distributions).

$$\text{exponential}(net) = e^{net}$$

If you are fitting a multinomial target, the *softmax* activation function (see below) is appropriate. It is the inverse of the *generalized logit-link* function.

$$\text{softmax}(net_i) = \frac{e^{neti}}{\sum_j e^{net_j}}$$

Because softmax divides the output activation of each neuron by the sum of the output by all participating neurons, it ensures that the estimates sum to one. This produces a distributed effect.

Finally, the identity activation function does not transform its argument at all. (See below.) This is useful when the desired response range is -∞ to ∞, such as when a normal target distribution is assumed.

$$\text{identity}(net) = net$$

Identity is the default output activation function.

Although not currently offered in the Neural Network node, an activation function known as the *rectifier* has now become the *de facto* standard in neural networks. Although many variants exist, the rectifier activation function is usually defined by the following equation:

$$rectifier\,(net) = \max(0, net)$$

It has been argued to be more biologically plausible than the widely used logistic sigmoid, and its more practical counterpart, the hyperbolic tangent (Wikipedia). A neuron using the rectifier activation function is called a *rectified linear unit* or, simply, a *rectilinear unit* (ReLU).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Modeling Assorted Target Distributions

c2s1d1

Underlying the Neural Network node is one of the most powerful ***statistical*** procedures in SAS, the NEURAL procedure. To illustrate its statistical capability, multinomial, ordinal, normal, Poisson, and gamma target distributions will all be fit using a single neuron. It is contended that the final weights generated by the linear perceptron generally closely match the parameter estimates produced by conventional statistical methods such as the GENMOD and LOGISTIC procedures.

The **FITNESS** data (Rawlings 1998) will be used. It contains information about 31 fitness club members who participated in a 1.5 mile run. The researchers were interested in determining whether any of several continuous measurements help predict the member's fitness (**oxygen_consumption**) level.

The variables in the **FITNESS** data set are as follows:

Variable	Description	Role	Level
age	The age (years) of the fitness club member	Rejected	Interval
gender	Gender of the fitness club member	Rejected	Nominal
maximum_pulse	The member's maximum pulse rate during the run	Rejected	Interval
name	First name of the fitness club member	Rejected	Nominal
oxygen_ consumption	A measure of oxygen transfer (i.e., fitness level)	Target	Interval
performance	Another measure of overall fitness	Input	Interval
rest_pulse	The member's pulse rate when at rest	Rejected	Interval
runtime	Time (minutes) it took the member to run 1.5 miles	Rejected	Interval
run_pulse	The member's pulse rate at the end of the run	Rejected	Interval
weight	The weight (kilograms) of the fitness club member	Rejected	Interval

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter the name **assorted targets** in the **Diagram Name** field and click **OK**.
3. Right-click **Data Sources** in the project panel (upper left corner) and select **Create Data Source**.
4. Click **Next** to progress to the second step of the metadata definition process.
5. Click **Browse** to choose a SAS table from the available libraries. Click the **dmnn41** library and select the **FITNESS** data table. Click **OK** to continue.
6. Click **Next** to display information about the **FITNESS** table, such as the number of variables (10) and the number of observations (31) it contains.
7. Again, click **Next** to begin modifying the metadata.
8. In this case, the (default) **Basic** setting suffices, so click **Next** to proceed.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

9. Change the role of all of the input variables, except for **performance**, to **Rejected**. Also change the role of **oxygen_consumption** to **Target**.

Name	Role	Level	Report	Order	Drop	Lower Limit	Upper Limit
Age	Rejected	Interval	No		No	.	.
Gender	Rejected	Nominal	No		No	.	.
Maximum_Pulse	Rejected	Interval	No		No	.	.
Name	Rejected	Nominal	No		No	.	.
Oxygen_Consumption	Target	Interval	No		No	.	.
Performance	Input	Interval	No		No	.	.
Rest_Pulse	Rejected	Interval	No		No	.	.
RunTime	Rejected	Interval	No		No	.	.
Run_Pulse	Rejected	Interval	No		No	.	.
Weight	Rejected	Interval	No		No	.	.

For clarity of exposition, only a single input is used in this demonstration. The Neural Network node could, of course, just as readily implement a generalized multiple regression model.

10. Click **Next** three times and then click **Finish**, completing the metadata definition process.
11. Drag and drop the **FITNESS** data source onto the diagram.
12. Add a **SAS Code** node (Utility tab) to the diagram and connect it to the **FITNESS** data source.

13. Activate the SAS Code node's property panel, and click the ellipsis (**…**) next to **Code Editor**.
14. Right-click in the (default) Training Code editor window, and select **Open** from the pop-up menu.
15. Read in program **c2s1d1.sas** (in d:\workshop\winsas\dmnn41). The program is listed below.

```
options nodate nonumber;

proc rank data=&EM_IMPORT_DATA groups=3 out=&EM_EXPORT_TRAIN;
   var oxygen_consumption;
   ranks target_d;
run;
```

To illustrate the Neural Network node's ability to model categorical (i.e. nominal and ordinal) target variables, the RANK procedure is used to create three (approximately) equal sized groups. The rank variable **target_d** reflects the group membership of each observation.

Next the GENMOD procedure is asked to generate the parameter values under the assumption that the target is normally distributed. Notice that identity activation and normal distribution are specified.

```
title1 'Normally Distributed Target';
proc genmod data=&EM_EXPORT_TRAIN;
   model oxygen_consumption=performance/link=identity dist=normal;
run;
```

Again GENMOD is invoked, only this time exponential activation (LINK=LOG) and Poisson distribution (DIST=POISSON) are specified.

```
title1 'Poisson Distributed Target';
proc genmod data=&EM_EXPORT_TRAIN;
   model oxygen_consumption=performance/link=log dist=poisson;
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The code required to generate the fit to a gamma distributed target is almost identical to the Poisson code above. Only the distribution to be fit changes (i.e., DIST=GAMMA).

```
title1 'Gamma Distributed Target';
proc genmod data=&EM_EXPORT_TRAIN;
   model oxygen_consumption=performance/link=log dist=gamma;
run;
```

PROC LOGISTIC will be used to fit the discrete targets. More specifically, to fit a multinomial target, a generalized logit link function (LINK=GLOGIT) is requested. The REF=FIRST specification of the target indicates that the reference class for the group variable is to be the first class (i.e., class 0). This is necessary to get the same results as those produced by the Neural Network node.

```
title1 'Multinomial Target (Generalized Logit)';
proc logistic data=&EM_EXPORT_TRAIN;
   model target_d(ref=first)=performance/link=glogit;
run;
```

Finally, to fit an ordinal target, PROC LOGISTIC is again called, although this time it specifies that the cumulative logit link function (LINK=CLOGIT) should be applied. The DESC option in the PROC LOGISTIC statement reverses the sorting order for the levels of the response variable.

```
title1 'Ordinal Target (Cumulative Logit)';
proc logistic data=&EM_EXPORT_TRAIN desc;
   model target_d=performance/link=clogit;
run;
```

16. Run the flow from within the SAS Code node by clicking the running man icon found at the top of the page. Confirm the run by selecting **Yes** when you are asked.
17. Click the **Results** icon when training is complete.
18. Open the Output window (bottom right corner) and scroll to line **58** to see the parameter values generated by PROC GENMOD under the assumption that the target is normally distributed.

Analysis Of Maximum Likelihood Parameter Estimates

Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits		Wald Chi-Square	Pr > ChiSq
Intercept	1	34.5487	1.9466	30.7334	38.3641	314.99	<.0001
Performance	1	0.2264	0.0327	0.1623	0.2906	47.82	<.0001
Scale	1	3.2870	0.4174	2.5627	4.2160		

The scale reported by the GENMOD procedure is sometimes a transformation of the dispersion parameter in the log-likelihood function. With the exception of the normal distribution, it does not correspond to the variance of an observation.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

19. Scroll to line **105** to see the results under the assumption that the target is Poisson distributed.

Analysis Of Maximum Likelihood Parameter Estimates

Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits		Wald Chi-Square	Pr > ChiSq
Intercept	1	3.5865	0.0874	3.4153	3.7578	1685.60	<.0001
Performance	1	0.0047	0.0014	0.0019	0.0075	10.90	0.0010
Scale	0	1.0000	0.0000	1.0000	1.0000		

20. Scroll to line **152** to see the results under the assumption that the target is Gamma distributed.

Analysis Of Maximum Likelihood Parameter Estimates

Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits		Wald Chi-Square	Pr > ChiSq
Intercept	1	3.5845	0.0408	3.5044	3.6645	7701.31	<.0001
Performance	1	0.0048	0.0007	0.0034	0.0061	48.07	<.0001
Scale	1	211.8727	53.7734	128.8367	348.4260		

21. Next the PROC LOGISTIC results are shown. First, scroll to line **227** to see the results generated under the assumption that the target (**target_d**) is multinomial.

Analysis of Maximum Likelihood Estimates

Parameter	target_d	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	1	-2.5276	1.8901	1.7883	0.1811
Intercept	2	1	-7.4648	2.7521	7.3573	0.0067
Performance	1	1	0.0530	0.0371	2.0429	0.1529
Performance	2	1	0.1310	0.0480	7.4593	0.0063

22. Lastly, examine the results under the assumption that the discrete target is ordinal (line **309**).

Analysis of Maximum Likelihood Estimates

Parameter		DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	2	1	-6.3897	1.8439	12.0082	0.0005
Intercept	1	1	-4.2804	1.6202	6.9795	0.0082
Performance		1	0.0956	0.0301	10.0790	0.0015

23. Close the Training Code window.

24. Add a **Metadata** node to the flow, and connect it to the output of the **SAS Code** node. The flow should now look similar to the one shown below.

25. In the Variables section of the Metadata node's properties panel, click the ellipsis (**…**) next to **Train**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

26. Change the level of the **target_d** variable to **Nominal**.

Name	Hidden	Hide	Role	New Role	Level	New Level	New Order	New Report
Age	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Gender	N	**Default**	Rejected	**Default**	Nominal	**Default**	**Default**	**Default**
Maximum_Pulse	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Name	N	**Default**	Rejected	**Default**	Nominal	**Default**	**Default**	**Default**
Oxygen_Consumption	N	**Default**	Target	**Default**	Interval	**Default**	**Default**	**Default**
Performance	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**
Rest_Pulse	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
RunTime	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Run_Pulse	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Weight	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
target_d	N	**Default**	Target	**Default**	Interval	**Nominal**	**Default**	**Default**

27. Click **OK** to save these changes.
28. Right-click the **Metadata** node and select **Run** from the pop-up window.
29. Click **OK** (not Results) when the node has finished.
30. Add a **Neural Network** node (Model tab) to the diagram and connect it to the **Metadata** node.

31. Because neural networks can model multiple targets at the same time, to first model continuous target distributions, click the ellipsis (**,..**) next to **Variables** in the Train section of the Neural network node's properties panel, and set the Use status of **target_d** to **No**.

Name	Use	Report	Role	Level
Age	**Default**	**No**	Rejected	Interval
Gender	**Default**	**No**	Rejected	Nominal
Maximum_Pulse	**Default**	**No**	Rejected	Interval
Name	**Default**	**No**	Rejected	Nominal
Oxygen_Consumption	**Yes**	**No**	Target	Interval
Performance	**Default**	**No**	Input	Interval
Rest_Pulse	**Default**	**No**	Rejected	Interval
RunTime	**Default**	**No**	Rejected	Interval
Run_Pulse	**Default**	**No**	Rejected	Interval
Weight	**Default**	**No**	Rejected	Interval
target_d	**No**	**No**	Target	Nominal

32. Click **OK** to save the change.
33. Next click the ellipsis (**…**) next to the **Network** field.
34. Change the Architecture property to **Generalized Linear Model**, and click **OK** to save your change.

Property	Value
Architecture	Generalized Linear Model
Direct Connection	No
Number of Hidden Units	3
Randomization Distribution	Normal
Randomization Center	0.0
Randomization Scale	0.1
Input Standardization	Standard Deviation
Hidden Layer Combination Function	Default
Hidden Layer Activation Function	Default
Hidden Bias	Yes
Target Layer Combination Function	Default
Target Layer Activation Function	Default
Target Layer Error Function	Default

35. The default for a continuous target is a normal distribution, so just run the Neural Network node by right-clicking the node and selecting **Run** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

36. Access the Results window and maximize the Output panel (bottom right panel). Go to line **118**.

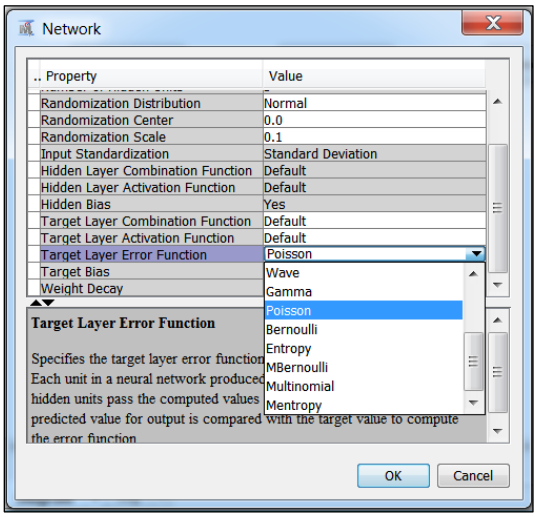
```
                    Optimization Results
                     Parameter Estimates
                                                        Gradient
                                                       Objective
 N Parameter                              Estimate       Function

 1 Performance_Oxygen_Consumption          0.226446    2.842171E-13
 2 BIAS_Oxygen_Consumption                34.548745    3.208903E-15
```

The **performance_oxygen_consumption** parameter (0.226446) is the slope coefficient and **BIAS_oxygen_consumption** is the y-intercept (34.548745).

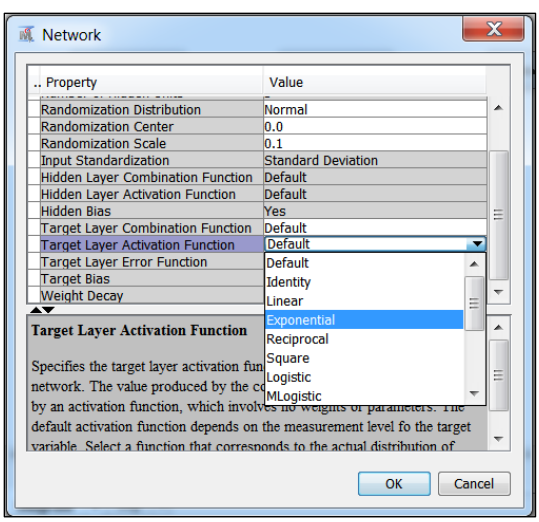
37. Close the Results window.

38. Activate the Neural Network node's properties panel and click the ellipsis (**…**) next to **Network**.

39. To fit the target under the assumption that it is Poisson distributed, change the Target Layer Error Function property from Default to **Poisson**.



40. Because Poisson distributions are undefined for negative arguments, also change the Target Layer Activation Function property from Default to the **Exponential** activation (i.e., log-link) function.

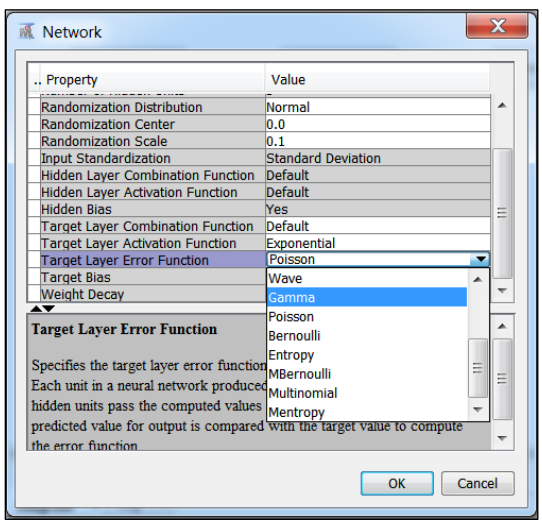


Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

41. Click **OK** to save your changes.
42. Rerun the Neural Network node. When training is complete, click **Results**.
43. Maximize the Output panel and go to line **118** to see the slope and y-intercept parameters under the assumption that the target is Poisson distributed.

```
                        Optimization Results
                         Parameter Estimates
                                                        Gradient
                                                       Objective
 N Parameter                                Estimate     Function

 1 Performance_Oxygen_Consumption           0.004730   0.000005225
 2 BIAS_Oxygen_Consumption                  3.586540   4.3404534E-9
```

44. Close the Results window.
45. To get the parameter values under the assumption that the target is gamma distributed, reopen the Network window and change the Target layer Error Function property to **Gamma**.



46. Like the Poisson distribution, the gamma distribution is undefined for negative arguments. However, the Target layer Activation Function property is already set to exponential, so click **OK**.
47. Right-click the **Neural Network** node and select **Run**. When training is complete, click **Results**.
48. Maximize the Output panel, and go to line **118** to see the slope and y-intercept parameters generated under the assumption that the target is gamma distributed.

```
                        Optimization Results
                         Parameter Estimates
                                                        Gradient
                                                       Objective
 N Parameter                                Estimate     Function

 1 Performance_Oxygen_Consumption           0.004766  -0.000000143
 2 BIAS_Oxygen_Consumption                  3.584453   6.669134E-11
```

49. Close the Results window.
50. To model discrete (nominal and ordinal) targets, click the ellipsis (**…**) next to **Variables** in the Train section of the Neural Network node's properties panel.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

51. Change the Use status of **target_d** to **Yes** and **oxygen_consumption** to **No**. Click **OK**.

Name	Use	Report	Role	Level
Age	**Default**	**No**	Rejected	Interval
Gender	**Default**	**No**	Rejected	Nominal
Maximum_Pulse	**Default**	**No**	Rejected	Interval
Name	**Default**	**No**	Rejected	Nominal
Oxygen_Consumption	**No**	**No**	Target	Interval
Performance	**Default**	**No**	Input	Interval
Rest_Pulse	**Default**	**No**	Rejected	Interval
RunTime	**Default**	**No**	Rejected	Interval
Run_Pulse	**Default**	**No**	Rejected	Interval
Weight	**Default**	**No**	Rejected	Interval
target_d	**Yes**	**No**	Target	Nominal

52. Reopen the Neural Network node's Network window and change both the Target layer Activation Function and Target layer Error Function properties back to **Default**. Click **OK** to save your changes.

When the target is nominal, the Neural Network node's default settings are the *linear* combination function, the *mlogistic* activation function, and the *mbernoulli* error function.

53. Right-click the **Neural Network** node and select **Run** from the pop-up window. Confirm the run.

54. Click **Results** when training is complete, and maximize the Output window.

55. To see the final parameter values under the assumption that the target is multinomial, go to line **157**.

```
                    Optimization Results
                    Parameter Estimates
                                                Gradient
                                               Objective
 N Parameter                      Estimate       Function

 1 Performance_target_d2          0.130967       -0.001191
 2 Performance_target_d1          0.053036        0.000763
 3 BIAS_target_d2                -7.464316    -0.000018140
 4 BIAS_target_d1                -2.526960     0.000015321
```

Because there are three classes to be modeled, two linear equations are required. Thus, two slope parameters and two intercepts are generated.

56. Close the Results window.

57. Finally, to model an ordinal target, activate the Metadata node's properties panel and click the ellipsis (**…**) next to **Train** in the Variables section.

58. Change the new level of the **target_d** variable to **Ordinal**. Click **OK** to save the change.

Name	Hidden	Hide	Role	New Role	Level	New Level	New Order	New Report
Age	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Gender	N	**Default**	Rejected	**Default**	Nominal	**Default**	**Default**	**Default**
Maximum_Puls	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Name	N	**Default**	Rejected	**Default**	Nominal	**Default**	**Default**	**Default**
Oxygen_Consu	N	**Default**	Target	**Default**	Interval	**Default**	**Default**	**Default**
Performance	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**
Rest_Pulse	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
RunTime	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Run_Pulse	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
Weight	N	**Default**	Rejected	**Default**	Interval	**Default**	**Default**	**Default**
target_d	N	**Default**	Target	**Default**	Interval	**Ordinal**	**Default**	**Default**

When the target is ordinal, the Neural Network node's default settings are the *EQslopes* combination function, the *logistic* activation function, and the *mbernoulli* error function.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

59. Run the flow from the Neural Network node, and examine the results when they are available.
60. Maximize the Output window (bottom right) and go to line **161**. The output listing is shown below.

```
                    Optimization Results
                    Parameter Estimates
                                                Gradient
                                               Objective
 N Parameter                     Estimate       Function

 1 Performance_target_d2         0.095427      -0.021465
 2 BIAS_target_d2               -6.383805      -0.000214
 3 BIAS_target_d1               -4.274702      -0.000108
```

Although two equations are again required to model the three target classes, when the target is ordinal, the slope of the equations will all be the same. Only the bias (intercept) changes.

61. Close the Results window.
62. Close the “assorted targets” diagram.

Summary

The following table summarizes the results.

Distribution	Parameter Identifiers		Parameter Estimates	
	Standard Method	Neural Procedure	Standard Method	Neural Procedure
Normal	Intercept	BIAS_Oxygen_Consumption	34.5487	34.548745
	Performance	Performance_Oxygen_Consumption	0.2264	0.226446
Poisson	Intercept	BIAS_Oxygen_Consumption	3.5865	3.586540
	Performance	Performance_Oxygen_Consumption	0.0047	0.004730
Gamma	Intercept	BIAS_Oxygen_Consumption	3.5845	3.584453
	Performance	Performance_Oxygen_Consumption	0.0048	0.004766
Multinomial	Intercept 1	BIAS_target_d1	-2.5276	-2.526960
	Intercept 2	BIAS_target_d2	-7.4648	-7.464316
	Performance 1	Performance_target_d1	0.0530	0.053036
	Performance 2	Performance_target_d2	0.1310	0.130967
Ordinal	Intercept 1	BIAS_target_d1	-6.3897	-6.383805
	Intercept 2	BIAS_target_d2	-4.2804	-4.274702
	Performance	Performance_target_d2	0.0956	0.095427

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Generalized linear models are simpler to estimate than neural networks. The objective function of a generalized linear model is convex, so there is only one extreme point (the global maximum). Thus, any set of initial values should give essentially the same parameter estimates, with only small differences explained by the different stopping rules or different iterative algorithms.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.01 Poll

A linear perceptron is a nonlinear model.

- ❍ True
- ❍ False

8

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

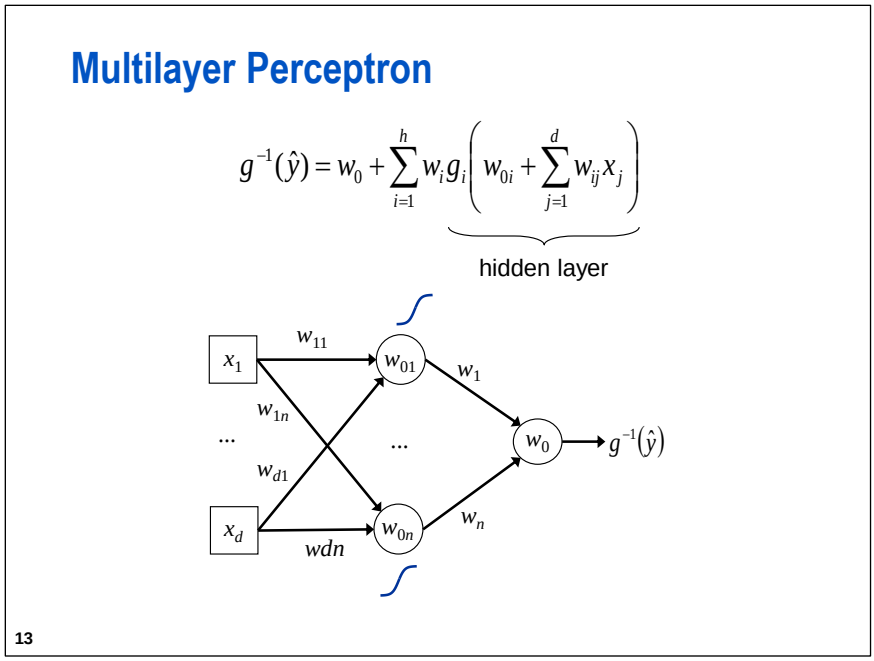
2.2 Multilayer Perceptrons

Objectives

- Detail the following neural network architectures:
  - multilayer perceptron
  - skip-layer perceptron

12

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



There are three layers in the basic multilayer perceptron (MLP) neural network:

1. An *input layer* containing a neuron/unit for each input variable. The input layer neurons have no adjustable parameters (weights). They simply pass the positive or negative input to the next layer.
2. A *hidden layer* with hidden units that (by default) perform a sigmoidal transformation of the weighted and summed input activations.
3. An *output layer* that shapes and combines the nonlinear hidden layer activation values.

A single hidden-layer multilayer perceptron constructs a limited extent region, or 'bump', of large values surrounded by smaller values (Principe et al. 2000). The intersection of the hyper-planes created by a hidden layer consisting of three hidden units, for example, forms a triangle-shaped bump.

The hidden and output layers ***must*** be connected by a nonlinear function in order to act as separate layers. Otherwise, the multilayer perceptron collapses into a linear perceptron. More formally, if matrix **A** is the set of weights that transforms input matrix **X** into the hidden layer output values, and matrix **B** is the set of weights that transforms the hidden unit output into the final estimates **Y**, then the linearly connected multilayer network can be represented as **Y**=**B**[**A**(**X**)]. However, if a single-layer weight matrix **C**=**BA** is created, exactly the same output can be obtained from the single-layer network, i.e. **Y**=**C**(**X**).

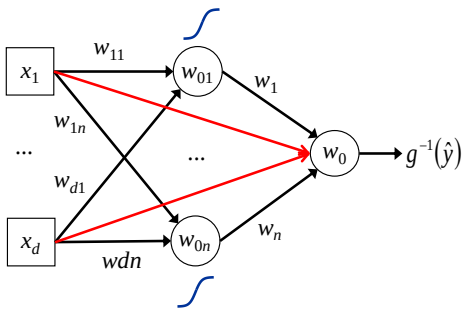
The number of parameters in an MLP with *k* interval inputs grows quickly with the number of hidden units, *h,* considered. The number of parameters is given by the equation $h(k+1)+h+1=h(k+2)+1$.

The "number of parameters" equations in this chapter assume that the inputs are interval or ratio level. Each nominal or ordinal input increases *k* by the number of classes in the variable, minus 1.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Skip-Layer Perceptron

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i g_i \underbrace{\left(w_{0i} + \sum_{j=1}^{d} w_{ij} x_j \right)}_{\text{hidden layer}} + \underbrace{\sum_{k=1}^{d} w_{1k} x_k}_{\text{skip layer}}$$



14

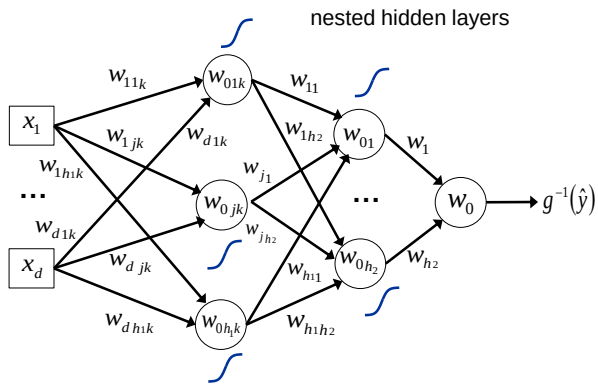
By adding direct connections from the input to output layers, bypassing the hidden layer, it is possible to combine the linear and nonlinear neural network paradigms. The result is known as a *skip-layer network*.

Because a multilayer perceptron is already a universal approximator, in general there is little to be gained by adding direct connections. Adding direct connections does ***not*** make it any more universal. However, a multilayer perceptron is inherently stationary (Leisch et al. 1999). This means that it will tend to perform poorly when applied to nonstationary data. In this case, adding direct connections can help.

The number of parameters in a skip-layer network with k inputs and h hidden units is $h(k+2)+k+1$.

MLP with Two Hidden Layers

$$g^{-1}(\hat{y}) = w_0 + \sum_{k=1}^{h_2} w_k g_k \underbrace{\left(w_{0k} + \sum_{j=1}^{h_1} w_{jk}\, g_j \underbrace{(w_{0jk} + \sum_{i=1}^{d} w_{ijk} x_i)} \right)}_{\text{nested hidden layers}}$$



15

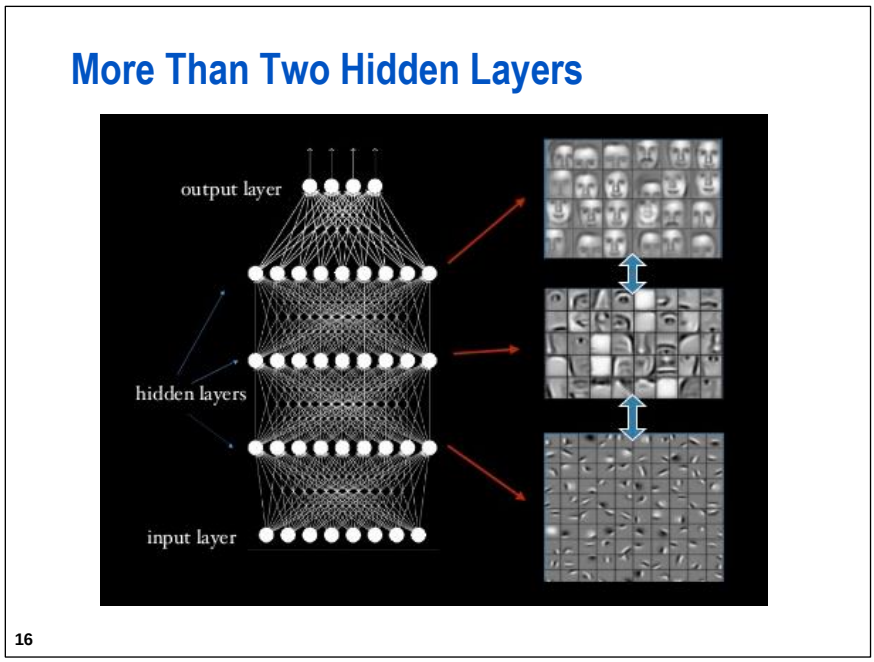
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

As Sarle (1997) writes in his Neural Network FAQ, “If you have only one input, there seems to be no advantage to using more than one hidden layer. But things get much more complicated when there are two or more inputs.”

When a second layer of hidden units is added, the single layer network’s ‘bumps’ form disjoint regions (Principe et al. 2000). The number of neurons in the second hidden layer determines the number of bumps formed in the input space (Principe et al. 2000). Now approximations in different areas of the input space can be adjusted independently of each other. This gives an MLP with two hidden layers the ability to realize discontinuous input-output mappings.

Unfortunately, the number of parameters in a two-layer network grows very quickly. If there are h_1 and h_2 units in the first and second hidden layers respectively, and k (interval) inputs, the number of parameters is given by the following equation:

$$h_1(k+1)+h_2(h_1+1)+h_2+1$$

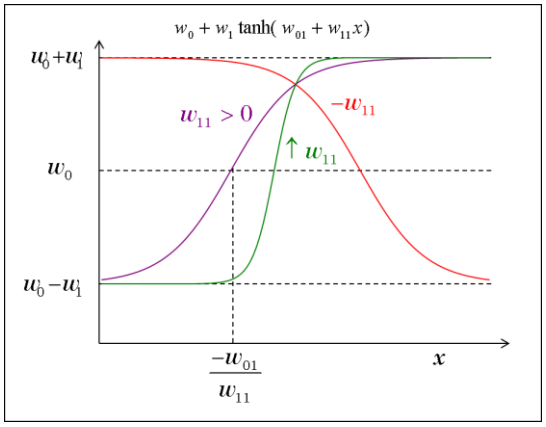


You need at most two hidden layers to approximate any function (Cybenko 1988). In fact, with gradient-based learning methods, it has been found that when more than two hidden layers are used, learning often slows to a crawl. This is known as the *vanishing gradient problem*. This problem arises because the chain rule, used to update the hidden unit weights, has the effect of multiplying n small numbers to compute the gradients of the front layers in an n-layer network. The gradient therefore decreases exponentially with n.

Recently, however, a major advance in neural network research known as *deep learning* (Hinton 2006) has shown that consideration of more than two hidden layers can offer substantial gains. The layers are typically trained in pairs, thereby overcoming the vanishing gradient problem. Deep learning methods transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. In order to model a target vector, a final layer of output units can be added to the top of the network, and the whole system can be fine-tuned using backpropagation.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Shaping the Sigmoid



17

The weights and biases give the sigmoidal surfaces their range. For example, because the maximum value returned by the hyperbolic tangent activation function is one, the upper bound of the hyperbolic tangent's activation range is given by the output unit's bias plus its weight:

$$w_0 + w_1 \tanh(w_{01} + w_{11}x) = w_0 + w_1(1) = w_0 + w_1$$

Conversely, the minimum activation value of the hyperbolic tangent function is negative one, which means that the lower bound of the hyperbolic tangent's activation range is as follows:

$$w_0 + w_1 \tanh(w_{01} + w_{11}x) = w_0 + w_1(-1) = w_0 - w_1$$

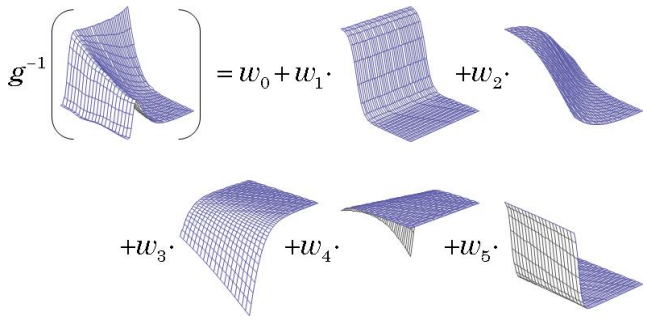
The weights and biases also give the sigmoid surfaces their flexibility. The sign of the weight associated with input x controls the direction of the sigmoid. Positive weight values produce the familiar s-shaped curve, and a negative weight value flips the sigmoid horizontally. The larger the absolute value of the weight, the steeper the curve. **Steep sigmoids are often held to be responsible for overfitting**.

Weight decay is one way to help keep the sigmoids from becoming too steep. Early stopping is another way. Both methods are discussed in Chapter 3.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Sigmoidal Basis Functions

By combining (adding) the weighted sigmoids from the hidden layer neurons, any shape can be modeled.



18

The output of a multilayer perceptron with one hidden layer is a weighted linear combination of the sigmoidal *basis functions* generated by the hidden layer neurons. Changing the orientation and steepness of the component sigmoidal basis functions, and then combining them, enables you to fit any target.

How Many Hidden Units Do You Need?

- The optimal number of units required in each hidden layer is problem specific.
- Principe et al. (2000) offered the following guidelines:
  - The number of units in the first hidden layer should be about twice the number of input dimensions.
  - The number of units in the second hidden layer reflects the number of distinct regions needed.
- The optimal number can be determined empirically.

19

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The question of the number of hidden units required is more difficult to answer than the required number of layers. If the network has too many hidden units, it will model random variation (i.e., noise) as well as the desired pattern (i.e., signal). This means that the model will fail to generalize. Conversely, having too few hidden units will fail to adequately capture the underlying signal. This means that the error value will tend to stabilize at a high value. Again, the model will fail to generalize.

Unfortunately, **the optimal number of hidden units is problem specific.**

There are, however, guidelines. For example, Principe et al. (2000) suggest that the number of units in the first hidden layer should be about twice the number of input dimensions. This will reflect the number of discriminant functions in the input space. If a second hidden layer is required, then the number of hidden units in the second layer should reflect the number of distinct regions needed (Principe et al. 2000).

The appropriate number of hidden units is, perhaps, best determined empirically. You start with a linear network and measure its performance on some appropriate metric, like the Schwarz-Bayesian criterion. Then increase the number of hidden units by one, and observe the impact on the network's fit. Continue adding hidden units until the network's performance drops. The final network is given by the number of hidden units in the network prior to the hidden unit addition that degraded the model's performance.

The sequential network construction algorithm, discussed in Chapter 4, illustrates incremental search. The Search option in the Autoneural node also implements incremental search.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Compositional Data

c2s2d1

A big advantage of neural networks is that they can model multiple targets, even targets of different types, ***in the same model***. The problem of compositional data illustrates the importance of being able to consider information about other targets when determining the parameters for a given target. In a compositional data set, each observation is an *n*-tuple of nonnegative numbers that sum to 1. A classic example of a compositional data set is the breakdown of a rock into its constituent minerals. For example, the analysis of a given rock sample shows that it is (60% $element_1$, 30% $element_2$, 10% $element_3$).

In this demonstration, the **PRODUCTS** data set is used. It consists of 200 commercial customers that have purchased at least one of three different products (**PROD1**, **PROD2**, or **PROD3**). The targets are the proportion of sales of each of these three products. The sum of the proportions is one for all cases. There is only a single input variable, a scaled measure of the customer's tenure with the company (**CAGE**).

Variable	Description	Role	Level
CAGE	A scaled measure of customer's tenure with the company	Input	Interval
PROD1	Proportion of sales of product one	Target	Interval
PROD2	Proportion of sales of product two	Target	Interval
PROD3	Proportion of sales of product three	Target	Interval

First, the predicted proportion of sales of each product will be separately modeled. As will be shown, this creates a problem. The alternative is to simultaneously estimate the predicted proportion of sales of each product. The softmax activation function can be used in combination with the multiple entropy error function, to constrain the predicted proportions to sum to one.

1. Right-click **Diagrams** in the project panel, and select **Create Diagram**.
2. Enter **compositional data** in the **Diagram Name** field and click **OK**.
3. Right-click **Data Sources** in the project panel and select **Create Data Source**. Click **Next**.
4. Browse the **dmnn41** library, and select the **PRODUCTS** data table. Click **OK**.
5. Click **Next** to display information about the data table. There are 200 observations and 4 variables.
6. Again, click **Next** to start the metadata definition process.
7. In this case, the **Basic** setting suffices, so click **Next** to proceed.
8. Change the role of **PROD1**, **PROD2**, and **PROD3** to **Target**.

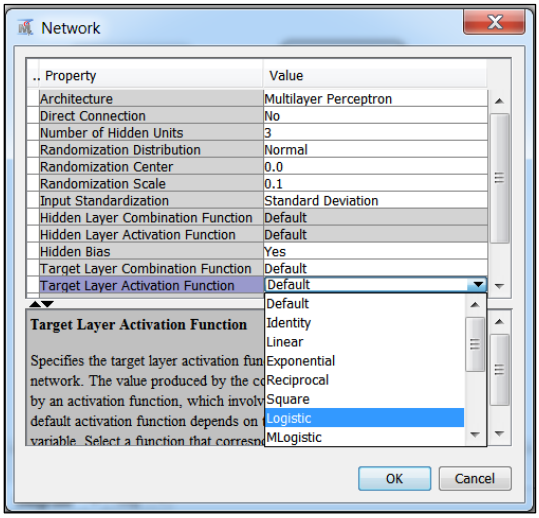
Name	Role	Level	Report	Order	Drop	Lower Limit
CAGE	Input	Interval	No		No	.
PROD1	Target	Interval	No		No	.
PROD2	Target	Interval	No		No	.
PROD3	Target	Interval	No		No	.

9. Click **Next** three times, and then click **Finish** to complete the metadata definition process.
10. Drag and drop the newly created **PRODUCTS** data table onto the diagram.

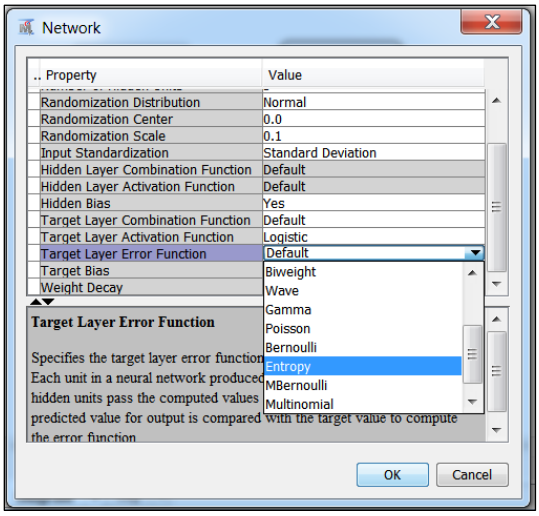
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

11. Add a **Neural Network** node (Model tab) to the diagram. Connect it to **PRODUCTS**.

12. Activate the Neural Network node's properties panel, and click the ellipsis (**…**) next to **Network**.
13. Change the Target Layer Activation Function property to **Logistic**.



14. Also change the Target Layer Error Function property to **Entropy**.

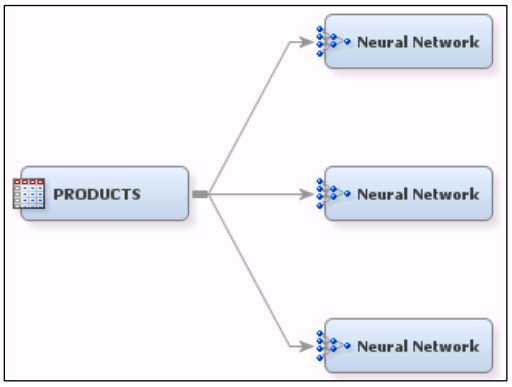


Although the identity link and normal error function could be used to separately model each product, constraining the expected target to a value between zero and one is more appropriate. Consequently, the logit link and cross-entropy error function have been specified. This also accounts for the fact that proportions closer to ½ have greater variability.

15. Click **OK** to save the changes.
16. Right-click the **Neural Network** node, and select **Copy** from the pop-up menu.
17. Place the cursor anywhere in the white space and right-click.
18. Select **Paste** from the pop-up window to paste a copy of the node onto the diagram.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

19. Repeat steps 17 and 18 above to copy a third version of the Neural Network node onto the diagram.
20. Connect the two copied nodes to the **PRODUCTS** data table. Your diagram should now look similar to the one shown below.



21. Select the top **Neural Network** node, and click the ellipsis (**…**) next to the Variables field.
22. Set the Use status of **PROD2** and **PROD3** to **No**. This means that only **PROD1** will be modeled.

Name	Use	Report	Role	Level
CAGE	Default	No	Input	Interval
PROD1	Yes	No	Target	Interval
PROD2	No	No	Target	Interval
PROD3	No	No	Target	Interval

23. Click **OK** to save the change.
24. Right-click the **Neural Network** node and select **Rename** from the pop-up menu.
25. Rename the node **Neural Network (1)** to reflect the fact that it models **PROD1**.
26. Next select the second **Neural Network** node.
27. Click the ellipsis (**...**) adjacent to its **Variables** field in the Neural Network node's properties panel.
28. Set the Use status of **PROD1** and **PROD3** to **No**. Click **OK** to save the change.

Name	Use	Report	Role	Level
CAGE	Default	No	Input	Interval
PROD1	No	No	Target	Interval
PROD2	Yes	No	Target	Interval
PROD3	No	No	Target	Interval

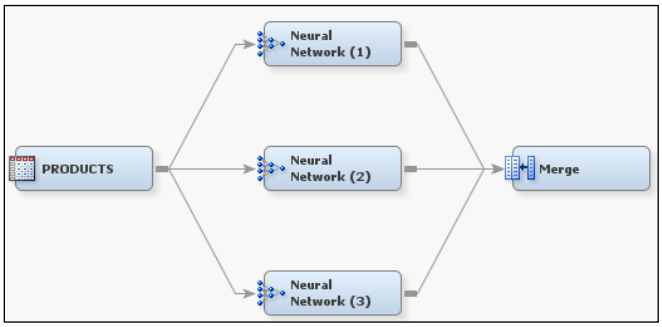
29. Right-click the **Neural Network** node and select **Rename** from the pop-up menu.
30. Rename the node **Neural Network (2)** to reflect the fact that it models **PROD2**.
31. Select the final **Neural Network** node.
32. Click the ellipsis (**...**) adjacent to its **Variables** field in the Neural Network node's properties panel.
33. Set the Use status of **PROD1** and **PROD2** to **No**. Click **OK** to save the change.

Name	Use	Report	Role	Level
CAGE	Default	No	Input	Interval
PROD1	No	No	Target	Interval
PROD2	No	No	Target	Interval
PROD3	Yes	No	Target	Interval

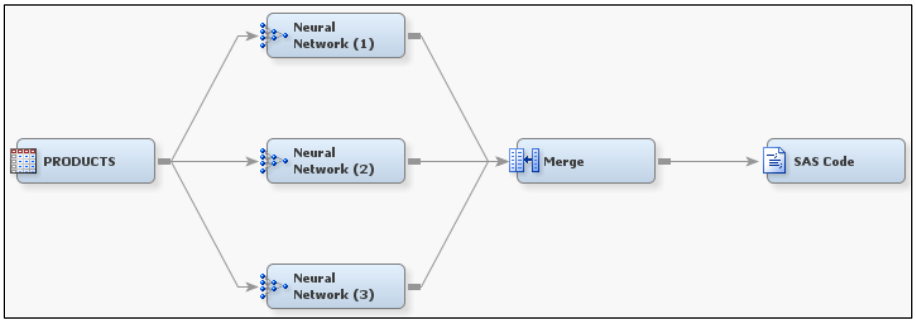
34. Right-click the **Neural Network** node and select **Rename** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

35. Rename the node **Neural Network (3)** to reflect the fact that it models **PROD3**.
36. Because you will need the predictions made by all three neural networks, drag and drop a **Merge** node (Sample tab) onto the diagram. Connect it to all three networks.



37. Also add a **SAS Code** node (Utility tab) to the diagram, and connect it to the **Merge** node. Your diagram should now look similar to the one shown below.



38. In the SAS Code node's properties panel, click the ellipsis (**…**) next to the **Code Editor** field.
39. In the Training Code editor, right-click in the white space and select **Open** from the menu.
40. Select the program **c2s2d1.sas** (in d:\workshop\winsas\dmnn41). The program is detailed below.

    First, the global systems and graphics options are initialized (not shown). Next an axis with a label, which is to be rotated by 90 degrees, and three global symbols, are defined.

```
axis1 label=(c=black a=-90 r=90);

symbol1 c=b v=none i=splines;
symbol2 c=bl v=circle i=none;
symbol3 c=r v=none i=splines l=3;
```

The SYMBOL1 statement gives the predicted relationship between *x* and *y*. It is a blue (C=B) line that uses splines (I=SPLINES) to interpolate. SYMBOL2 declares that the observations are black (C=BL) circles (V=CIRCLE). SYMBOL3, reflecting the total probability, is displayed as a red (C=R) line.

A new variable, **total**, is created in a DATA step by summing the predicted propensities to buy each product (**p_prod1**, **p_prod2**, **p_prod3**), as estimated by the independent neural networks.

```
data results;
   set &EM_IMPORT_DATA;
   total=sum(of p_prod1-p_prod3);
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

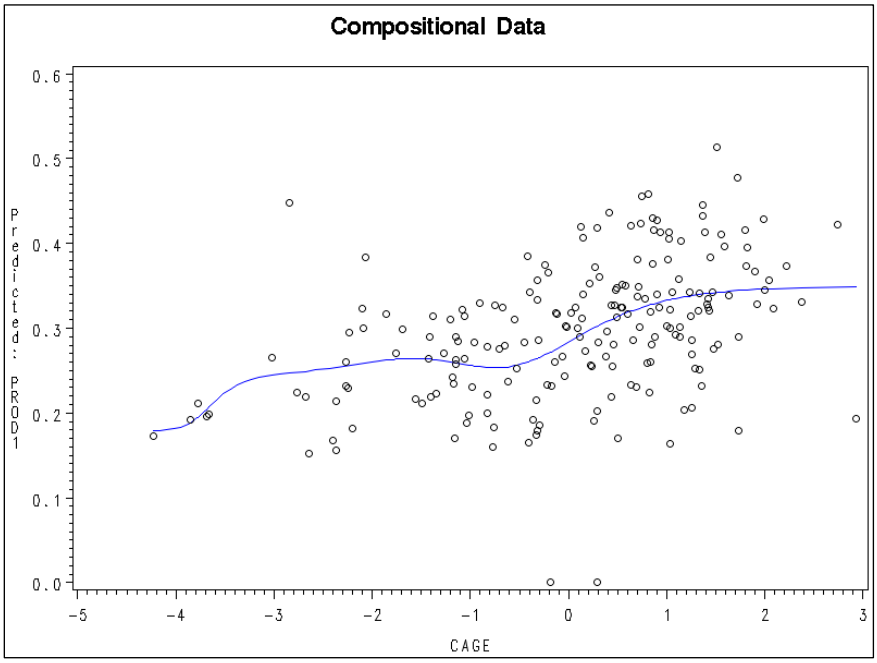
The last component of the program requests that PROC GPLOT generate a scatter plot of the solution. Each plot displays the probability of purchasing the corresponding product against the observations.

The final plot overlays the individual product propensities and the total of the three propensities.

```
title1 h=1.5 f=swissb 'Compositional Data';
proc gplot data=results;
   plot p_prod1*cage=1 prod1*cage=2 /
        overlay vaxis=axis1 description="plot 1";
   plot p_prod2*cage=1 prod2*cage=2 /
        overlay vaxis=axis1 description="plot 2";
   plot p_prod3*cage=1 prod3*cage=2 /
        overlay vaxis=axis1 description="plot 3";
   plot (p_prod1 p_prod2 p_prod3)*cage=1 total*cage=3 /
        overlay vaxis=axis1 description="plot 4";
run;
```

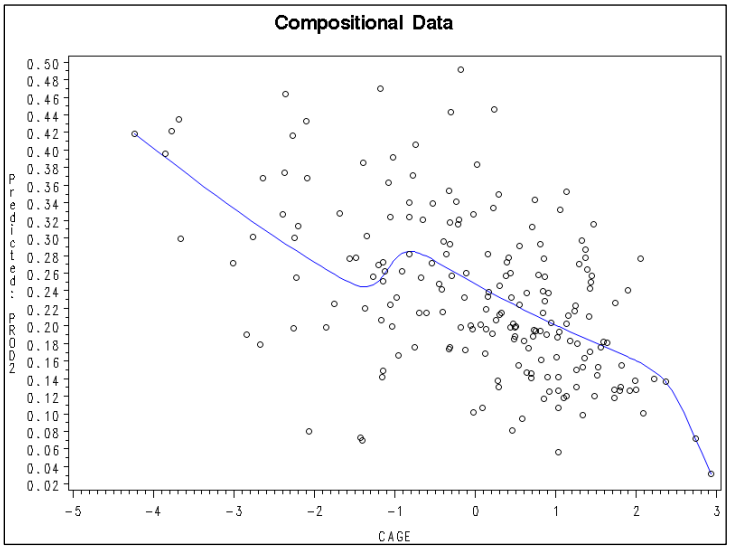
41. Click the running man icon to run the diagram from within the SAS Code node.
42. Confirm the run and click **Results** when the run is completed.
43. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs**, and maximize to see the requested plots.

    The first plot displayed shows the Neural Network node's solution for product one (**PROD1**).

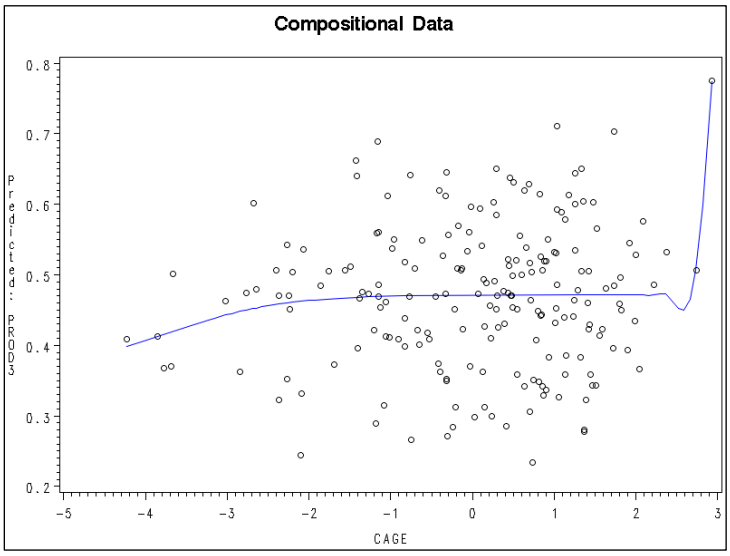


Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

44. Click the right arrow to see the proposed solution for the second banking product (**PROD2**).

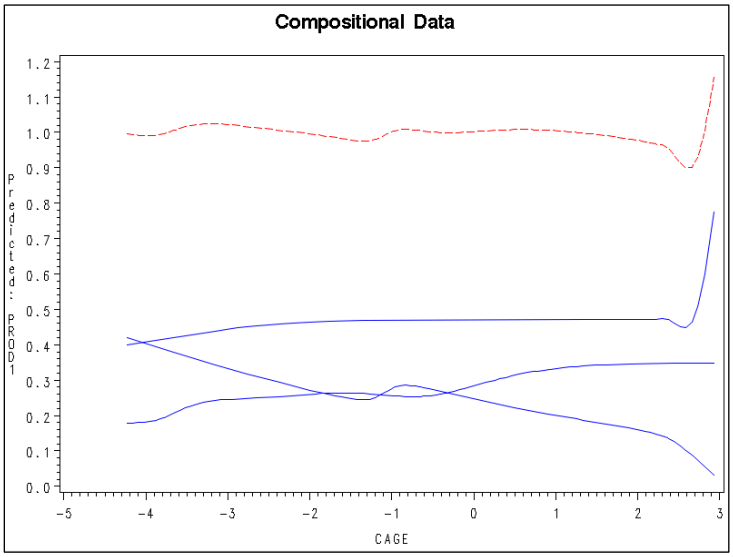


45. Click the right arrow again to see the neural network's predictions for product three (**PROD3**).



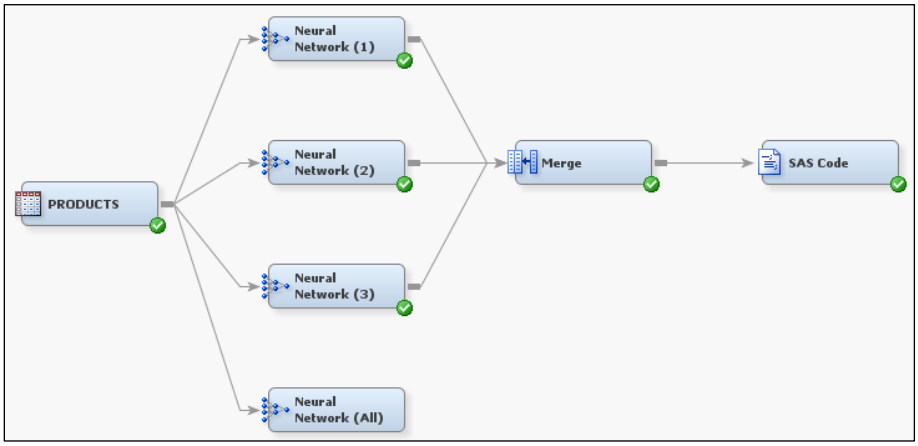
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

46. Click the right arrow one more time to show all three predicted plots overlaid with their total.



Notice that the sum of the three predicted proportions (the dashed line) is not always equal to one, as it should be when the data is compositional.

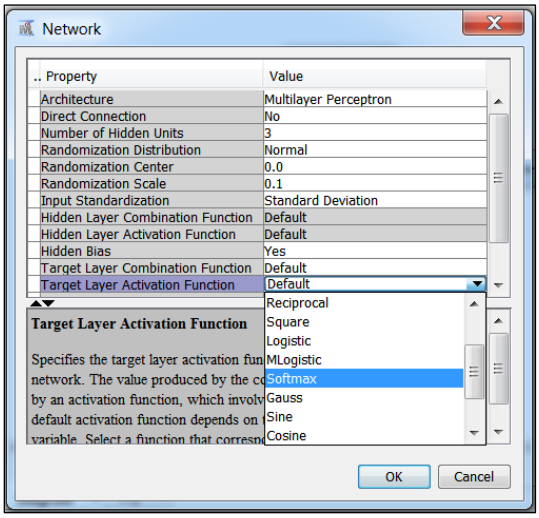
47. Close the SAS Code node Results window, and the SAS Code node.
48. Add a fourth Neural Network node (Model tab), and rename it **Neural Network (All)**.
49. Connect the new **Neural Network** node to the **PRODUCTS** data source.



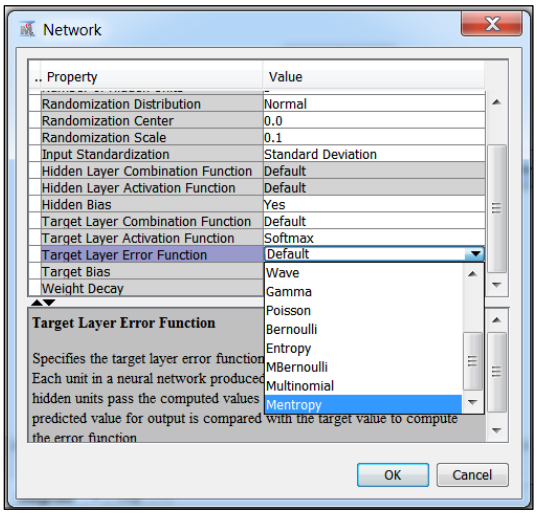
50. Activate the Neural Network node's properties panel, and click the ellipsis (**…**) next to **Network**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

51. Change the Target Layer Activation Function property to **Softmax**.



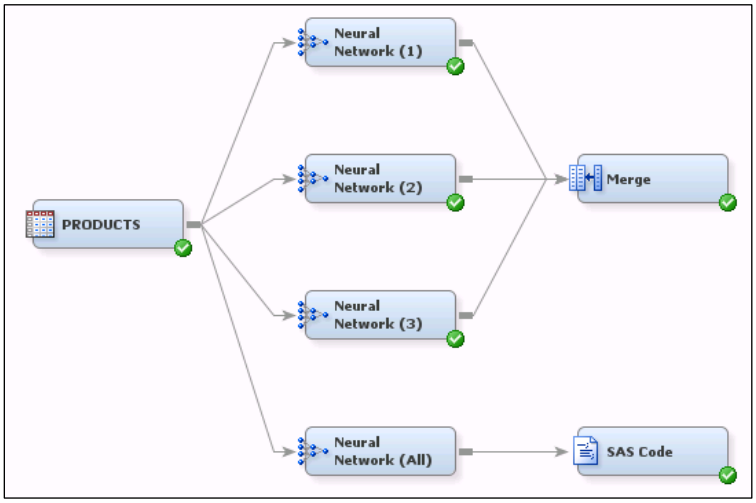
52. Also change the Target Layer Error Function property to **Mentropy**.



53. Click **OK** to save the changes.

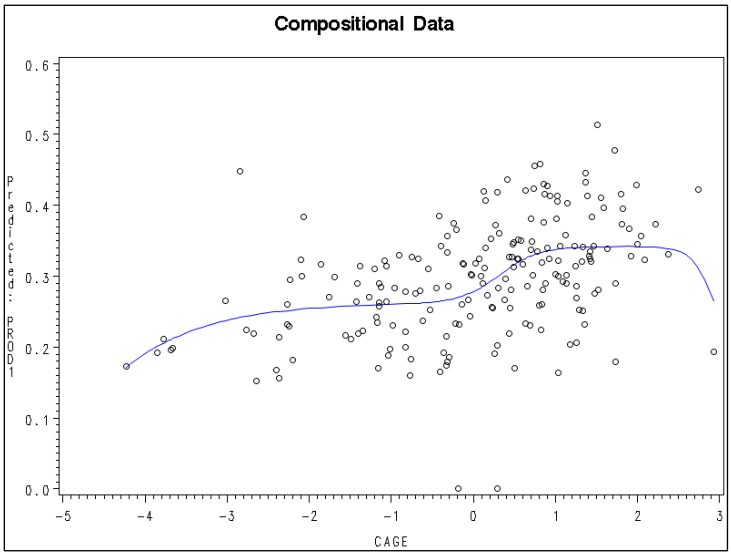
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

54. Remove the connection between the **SAS Code** node and the **Merge** node, and reconnect the **SAS Code** node to the **Neural Network (all)** node, as shown in the diagram below.



55. Run the diagram from the SAS Code node, and click **Results** when they are available.

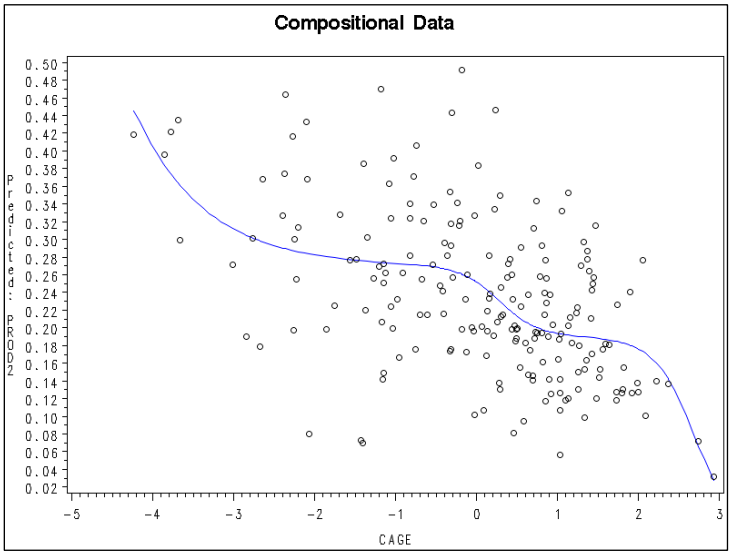
56. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** and maximize to see the new plots.



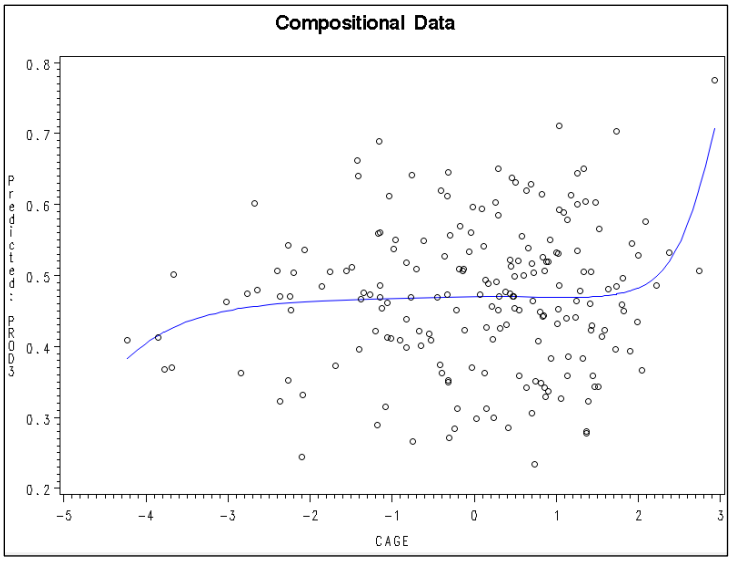
The first plot shows the Neural Network node's estimation of the proportion of product one (**PROD1**) purchased. The model (predicted line) has changed from the earlier **PROD1*CAGE** plot.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

57. Click the right arrow to see the proposed solution for the second banking product (**PROD2**). The sharp bend that was present in the earlier plot of **PROD2*CAGE** is now missing.

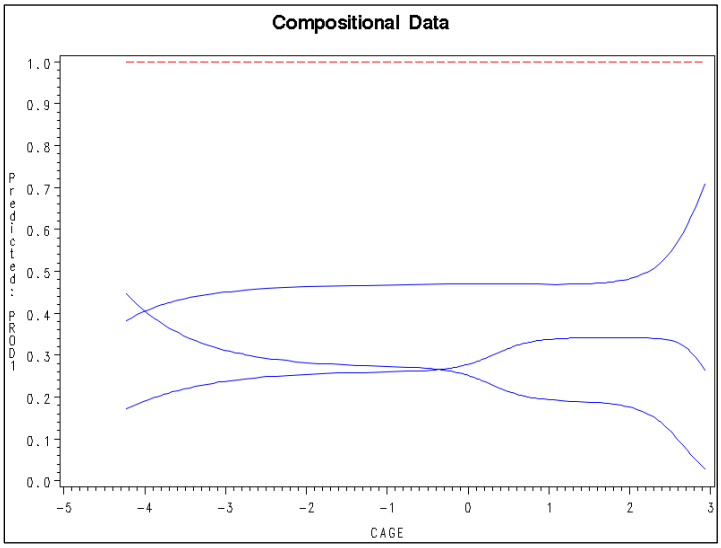


58. Click the right arrow again to reveal the neural network's predictions for product three (**PROD3**). The final plot is similar to the plot generated when **PROD3** was specified as the only target.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

59. Lastly, click the right arrow one last time to show the predicted plots and their combined total.



Notice that the three predicted sales proportions (dashed line) now properly sum to one.

60. Close the SAS Code node results.

61. Close the "compositional data" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.02 Poll

The addition of direct connections between the input and output layers (a skip-layer network) is usually beneficial.

- ❍ True
- ❍ False

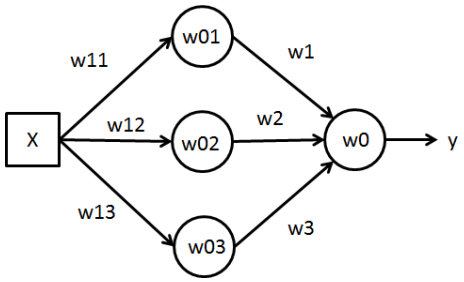
21

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Exercises

1. Constructing a Multilayer Perceptron Using PROC NLIN

In this exercise, you write the PROC NEURAL code to create the three-hidden-unit MLP shown below. Use the hyperbolic tangent (TANH) as the hidden unit activation function.



a. Right-click **Diagrams** in the project panel, and select **Create Diagram**.

b. Enter **exercise 2.1** in the **Diagram Name** field, and click **OK**.

c. Drag and drop the **DECAY** data set onto the diagram.

d. Add a **SAS Code** node (Utility tab) to the diagram, and connect it to the **DECAY** data source.

e. In the SAS Code node's properties panel, and click the ellipsis (**…**) next to **Code Editor**.

f. In the (default) Training Code editor, select **Open** from the pop-up menu.

g. Select program **exercise2.1.sas** (in d:\workshop\winsas\dmnn41). The program contains all required statements except for the MODEL statement, which you are asked to complete.

```
options nodate nonumber;
goptions reset=all device=actximg;

symbol1 c=b v=none i=join;
symbol2 c=bl v=circle i=none;
symbol3 c=r v=none i=join l=3;

title1 'Three Hidden Unit MLP in PROC NLIN';
proc nlin data=&EM_IMPORT_DATA method=marquardt convergeobj=1e-6;
   parms w0=60 w1=0 w2=0 w3=0 w01=-.01,.01 w02=-.03,.03
         w03=.05,.05 w11=-.02,.02 w12=-.04,.04 w13=-.06,.06;

   model *** your code goes here ***;

   output p=yhat out=result;
run;

proc gplot data=result;
   plot ey*x=1 y*x=2 yhat*x=3/overlay;
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

h. Fill in the missing NLIN procedure's MODEL statement.

Hint: Because you are asked to construct a three-hidden-unit MLP, you will need a bias term and three weighted hyperbolic tangent functions. Therefore, the solution will have this form:

```
Y = w0 + w1*tanh(…) + w2*tanh(…) + w3*tanh(…);
```

i. Run the flow from within the SAS Code node by right-clicking the running man icon.

j. Confirm the run and save your changes when asked.

k. Click **OK** when training is complete.

l. Click the **Results** icon, and go to line **184** in the output listing.

- Is the model significant at the alpha=0.05 level?

m. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs**, and maximize the plot.

- Based on a visual comparison to the true model (the blue line in the plot), does the PROC NLIN implementation of the MLP appear to result in a good fit to the data?

n. Close the train graphs and Results windows.

o. Close the "exercise 2.1" diagram.

End of Exercises

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.3 Radial Basis Functions

Objectives

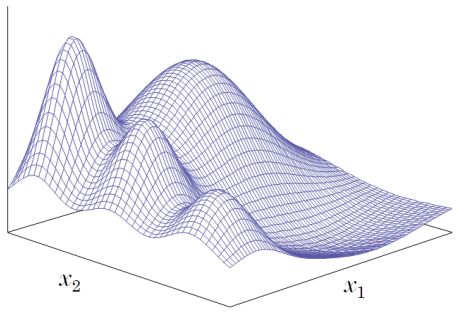
- Detail ordinary and normalized radial basis functions.

26

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Overview of Radial Basis Functions

- Ordinary Radial Basis Functions (ORBFs)
- Normalized Radial Basis Functions (NRBFs)



27

Radial basis function (RBF) networks offer a powerful alternative to their much more famous kin, the MLP. As Wasserman (1993) notes, RBFs have begun to attract a great deal of interest due to these factors:

- their ability to generalize (i.e. they are also universal approximators)
- their rapid training (often an order of magnitude faster than an MLP)
- the fact that they exhibit less of a tendency than an MLP to fall into bad local minima.

Radial basis function networks have many uses, including function approximation, classification, time series prediction, and system control. First formulated in a 1988 paper by Broomhead and Lowe, an RBF network is an artificial neural network that uses radial basis functions (Gaussians), instead of sigmoids, as the hidden unit activation functions. There are two radial basis function variants:

- ***Ordinary radial basis functions*** (ORBFs), which use an exponential activation function to generate a Gaussian localized at the hidden unit weights, and
- ***Normalized radial basis functions*** (NRBFs), in which the activation of each hidden unit is a product of the cooperative activation of all participating hidden units in the layer.

Normalized radial basis function networks usually outperform ordinary radial basis function networks.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Combination Functions

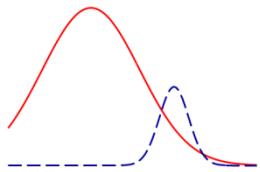
The combination functions that are appropriate when you construct a radial basis function are as follows:

- **XRADIAL** unequal height and width
- **EQRADIAL** equal height and width
- **EWRADIAL** equal width
- **EHRADIAL** equal height
- **EVRADIAL** equal volume

28

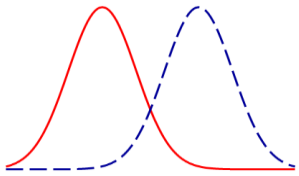
There are five varieties of radial basis functions available in the Neural Network node. They differ with respect to the degree of flexibility that they offer and the number of parameters that they require.

XRadial generates a radial basis function with unequal widths and unequal heights for all units in the layer. This is the least constrained and, thus, the most flexible variant. Because one parameter is needed to store the width of each Gaussian, and another is required to store its height, this is also the most memory-intensive option. If k is the number of inputs and h is the number of hidden units, the total number of parameters is given by $h(k+3)$.



$\text{Altitude}_i \neq \text{Altitude}_j$ and $\text{Width}_i \neq \text{Width}_j$

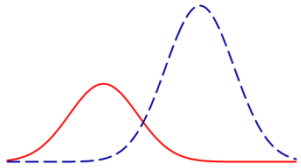
EQRadial generates a radial basis function with equal heights and equal widths for all units in the layer. Because it uses the same value for the height and width of each Gaussian, the total number of parameters is only $h(k+1)+1$. As noted, this decrease on memory load comes at a price; EQRadial is the least flexible of the radial basis functions.



$\text{Altitude}_i = \text{Altitude}_j$ and $\text{Width}_i = \text{Width}_j$

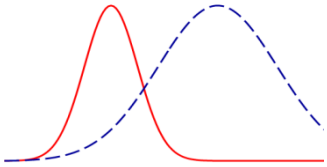
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

EWRadial generates a radial basis function with equal widths, but not equal heights, for all units in the hidden layer. The total number of parameters is given by $h(k+2)+1$.



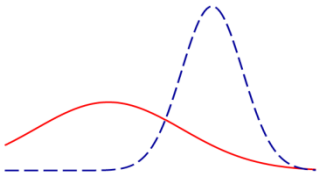
$\text{Altitude}_i \neq \text{Altitude}_j$ and $\text{Width}_i = \text{Width}_j$

EHRadial generates a radial basis function with equal heights, but not equal widths, for all units in the hidden layer. The total number of parameters is $h(k+2)$.



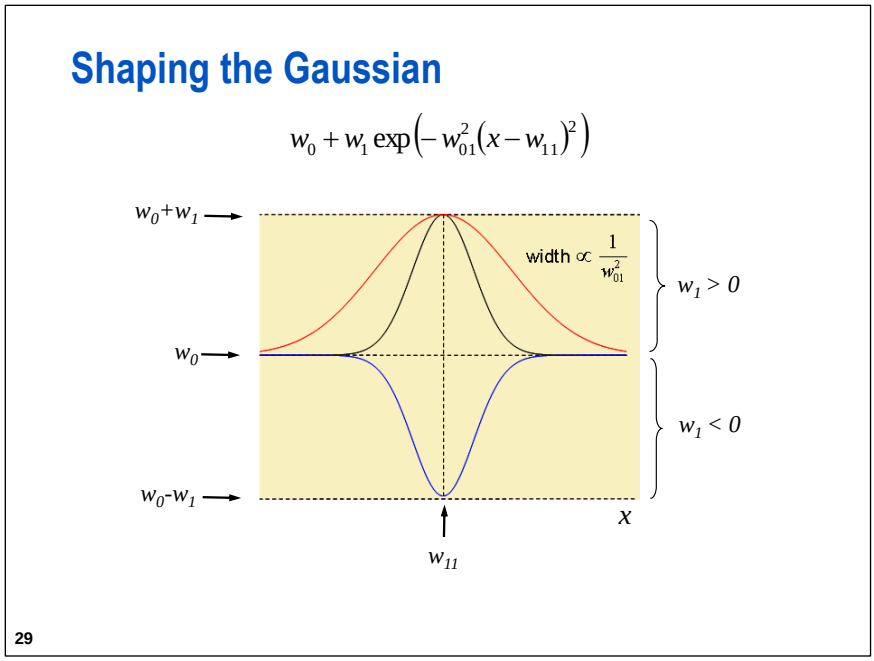
$\text{Altitude}_i = \text{Altitude}_j$ and $\text{Width}_i \neq \text{Width}_j$

EVRadial generates a radial basis function with equal volumes for all units in the hidden layer. The higher the Gaussian, the narrower it is (and vice versa), so that the total volume of each Gaussian in the network is constant. ***This option should probably rarely, if ever, be used***, because it is less reliable than the other methods. The total number of parameters in the equal volume radial variant is $h(k+2)$.



$\text{Altitude}_i = \text{Width}_j$

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Just as the hidden unit weights alter the direction and shape of the sigmoid in a multilayer perceptron, the weights in an ordinary radial basis function alter the Gaussian's direction and shape. The sign (+/-) of the hidden-to-output weight determines whether the response surface will be concave or convex. When the weight associated with the exponential activation function, w_1, is positive, the Gaussian is oriented in its familiar concave position. Conversely, when w_1 is negative, the Gaussian is inverted (i.e., convex).

Because the argument to the exponent function is negated, large distances between the input vector, **x**, and the Gaussian's mean vector, **w**, generate large negative values, which means that its value will approach zero. Therefore, the ***minimum*** value of the "upright" (concave) Gaussian is the following:

$$w_0 + w_1 \exp(-w_{01}^2 (x - w_{11})^2) = w_0 + w_1(0) = w_0$$

On the other hand, the smaller the squared Euclidean distance between **x** and **w**, the closer the output of the exponent function is to its maximum value of one. Thus, the ***maximum*** value returned is given by

$$w_0 + w_1 \exp(-w_{01}^2 (x - w_{11})^2) = w_0 + w_1(1) = w_0 + w_1$$

This, of course, means that the bottom of the "inverted" (convex) Gaussian is given by $w_0 - w_1$.

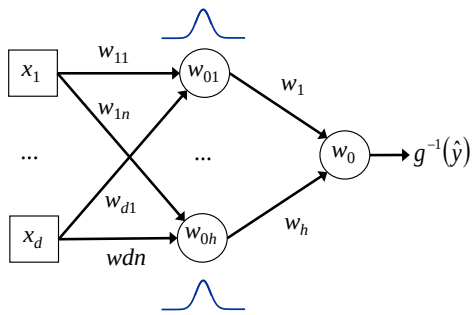
And, as was noted previously, the role of the hidden unit bias term, w_{01}, has also changed; it now controls the width of the Gaussian. Specifically, the magnitude of the bias term is proportional to the reciprocal of the squared width of the associated Gaussian basis function.

$$\text{width} \propto \frac{1}{w_{01}^2}$$

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Ordinary Radial Basis Function

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i \underbrace{\exp\left[-w_{0i}\left(\sum_j (w_{ij} - x_j)^2\right)\right]}_{\text{hidden unit}}$$



30

The output layer of an ordinary radial basis function is exactly the same as the output layer of a multilayer perceptron. If $\mathbf{H}_i$ is the output of hidden unit i, the output layer activation is given by the following:

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i H_i$$

It is only the behavior of the hidden units that has changed. They are now Gaussian surfaces centered at positions in the input space defined by the input-to-hidden weights. And the bias term now controls the Gaussian's width. More formally, the activation of hidden unit $\mathbf{H}_i$ in an ORBF is given by the following:

$$H_i = \exp\left[-w_{0i}^2\left(\sum_j (w_{ij} - x_j)^2\right)\right]$$

where x_{ij} is the jth input to hidden unit $\mathbf{H}_i$, w_{ij} is the jth weight to $\mathbf{H}_i$, and w_{0i} is the bias of hidden unit $\mathbf{H}_i$. The term in the inner brackets is the squared Euclidean distance between input vector, $\mathbf{x}$, and the Gaussian mean, which is given by the hidden unit's weight vector, $\mathbf{w}$. Smaller distances indicate that the input and mean vectors are closer together in the n-dimensional space. In other words, when the squared Euclidean distance is small, the hidden unit better represents the input vector.

Two variants of the ordinary radial basis function paradigm are available in the Neural Network node: equal width (ORBFEQ) and unequal width (ORBFUN). These variants are defined below.

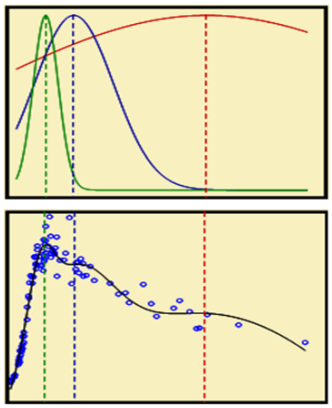
ORBFEQ $$H_i = exp\left\{-w_0^2 \sum_j \left(w_{ij} - x_j\right)^2\right\}$$

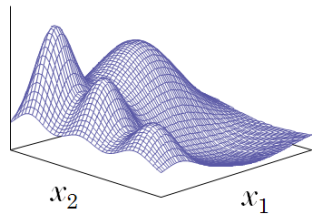
ORBFUN $$H_i = exp\left\{-w_{0i}^2 \sum_j \left(w_{ij} - x_j\right)^2\right\}$$

The only difference between these two variants is that the term controlling the width of the Gaussian is subscripted (w_{0i}) in the ORBFUN, and it is constant (w_0) in the ORBFEQ.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The Localized Effect Problem



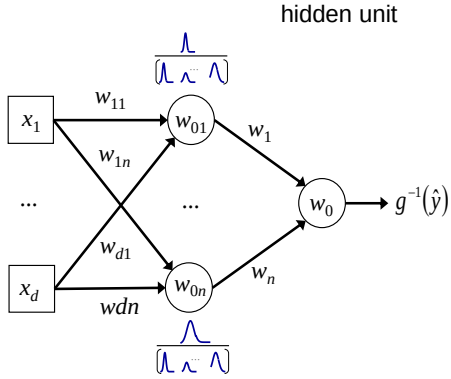


31

In theory ordinary radial basis function networks are universal approximators. In practice, however, they are often ineffective multivariate function estimators. The problem is that the Gaussian basis functions have a localized effect around their center. This means that a large number of hidden units might be required in order to avoid an excessively bumpy fit to the target distribution, making ordinary radial basis functions vulnerable to the *curse of dimensionality*.

Normalized Radial Basis Function

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i \underbrace{\text{softmax}\left\{ f\,.\ln(a_i) - w_{0i}^2 \left(\sum_j (w_{ij} - x_j)^2 \right) \right\}}_{\text{hidden unit}}$$



32

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

A normalized radial basis function (NRBF) overcomes the localization problem exhibited by ordinary radial basis functions by imposing the constraint that the hidden unit activations must add to one. The softmax activation function (Section 2.1) is used to ensure that this constraint is met. Because the output activation of each hidden unit is first divided by the sum of the activation of all hidden units in the layer, the output of each hidden unit is no longer a simple Gaussian.

The equal heights and widths (NRBFEQ) and the equal heights (NRBFEH) variants are defined below.

NRBFEQ $$H_i = \text{softmax}\left\{ -w_0^2 \sum_j \left(w_{ij} - x_j\right)^2 \right\}$$

NRBFEH $$H_i = \text{softmax}\left\{ -w_{0i}^2 \sum_j \left(w_{ij} - x_j\right)^2 \right\}$$

The equal widths (NRBFEW), unequal widths and heights (NRBFUN) and equal volumes (NRBFEV) variants are given by the following equations:

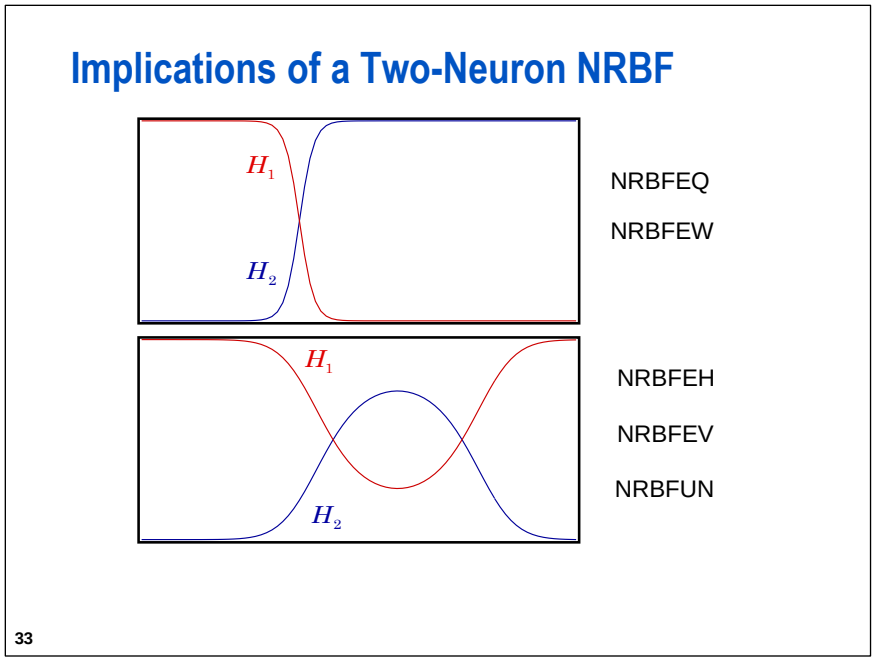
NRBFEW $$H_i = \text{softmax}\left\{ f.\ln(a_i) - w_0^2 \left(\sum_{j=1}^{d} (w_{ij} - x_j)^2 \right) \right\}$$

NRBFUN $$H_i = \text{softmax}\left\{ f.\ln(a_i) - w_{0i}^2 \left(\sum_{j=1}^{d} (w_{ij} - x_j)^2 \right) \right\}$$

NRBFEV $$H_i = \text{softmax}\left\{ f.\ln(w_{0i}) - w_{0i}^2 \left(\sum_{j=1}^{d} (w_{ij} - x_j)^2 \right) \right\}$$

Notice that an altitude (a_i) parameter has been added. Ordinary radial basis functions did not have an altitude parameter, as it would have been redundant with the hidden-to-output weights. A “fan-in” (f) parameter, reflecting the number of connections feeding into the hidden unit, has also been added.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



It is not surprising that the Neural Network node does not allow a one-neuron normalized radial basis function because it generates a constant. That is, if the output of the neuron is given by the formula

$$v_i = \exp\left[f.\ln(a_i) - w_{0i}^2 \left(\sum_{j=1}^{d} (w_{ij} - x_j)^2 \right) \right]$$

then the NRBF's softmax function would yield the following:

$$H_1 = \frac{v_1}{\sum_{i=1}^{1} v_i} = \frac{v_1}{v_1} = 1.$$

Interestingly, a two-hidden-unit normalized radial basis function is also impractical (Potts 2004). When the widths are equal (that is, either NRBFEQ or NRBFEW), a two-hidden-unit NRBF degenerates into two identical, but opposing, sigmoidal basis functions:

$$H_1 = \frac{v_1}{v_1 + v_2} = \frac{1}{1 + \frac{v_2}{v_1}} = \frac{1}{1 + \exp(\ln(a_2) - \ln(a_1) - w_{02}^2 \sum (x_j - w_{j2})^2 + w_{01}^2 \sum (x_j - w_{j1})^2)}$$

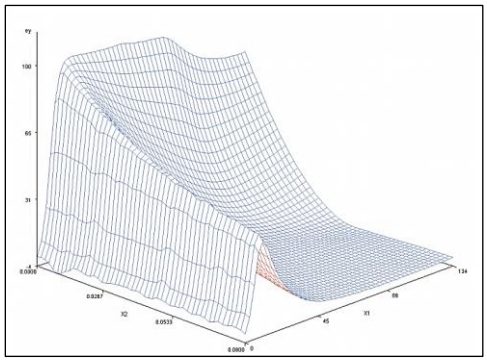
And the bases of the three unequal widths variants (NRBFEH, NRBFEV, or NRBFUN) form identical, but opposing, non-monotonic surfaces (see slide). Therefore, although two hidden units are allowed by the interface, ***the minimum number of hidden units to use in any NRBF is three***.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Plotting Basis Functions

c2s3d1

In this demonstration, you plot the basis functions generated by several RBF architectures as they attempt to model the three-dimensional nonlinear pattern (shown below), contained in the **ECC2D** data set.



The variables in the **ECC2D** data source are listed below.

Variable	Description	Role	Level
ey	The usually unknown true *y*-coordinate of the observation	Rejected	Interval
x1	The x_1-coordinate of the observation	Input	Interval
x2	The x_2-coordinate of the observation	Input	Interval
y	The *y*-coordinate of the observation	Target	Interval

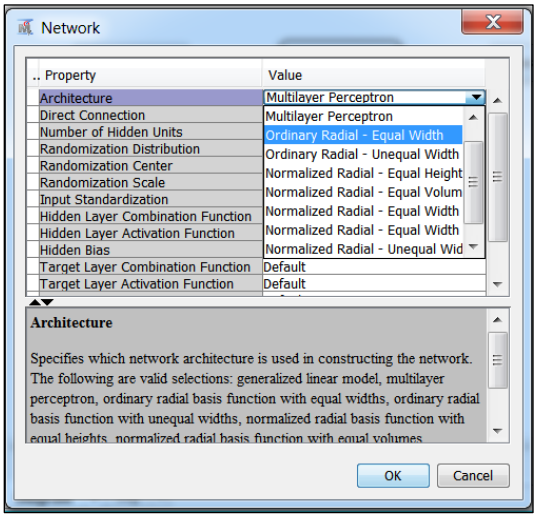
1. Right-click **Diagrams** and select **Create Diagram**.
2. Enter the name **basis functions** in the **Diagram Name** field and click **OK**.
3. Right-click **Data Sources** and select **Create Data Source**.
4. Click **Next** to create a SAS table.
5. Browse to find the **ECC2D** data table (in the **dmnn41** library). Click **OK**.
6. Click **Next** to display the **ECC2D** metadata. There are 500 observations and 4 variables in the table.
7. Click **Next** again to start the metadata definition process.
8. The **Basic** setting suffices in this case, so click **Next** to proceed.
9. Change the role of **y** to **Target**, and change the role of **ey** to **Rejected**.

Name	Role	Level	Report	Order	Drop	Lower Limit
ey	Rejected	Interval	No		No	.
X1	Input	Interval	No		No	.
X2	Input	Interval	No		No	.
y	Target	Interval	No		No	.

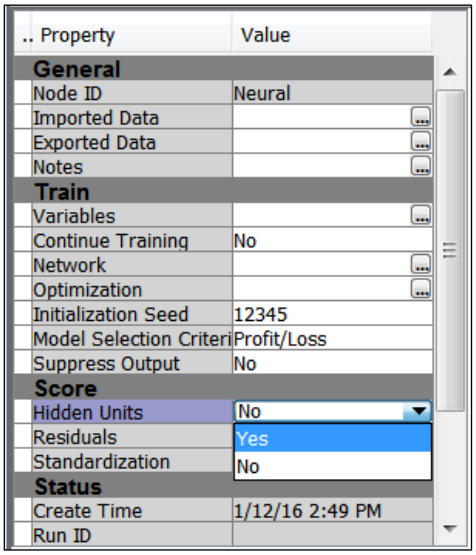
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

10. Click **Next** three times and then click **Finish** to complete the data source definition.
11. Drag and drop the **ECC2D** data source onto the diagram.
12. Add a **Neural Network** node (Model tab) to the diagram, and connect it to the **ECC2D** data source.

13. Click the ellipsis (**...**) next to the **Network** field in the Neural Network node's properties panel.
14. Change the Architecture property to **Ordinary Radial – Equal Width**. This is the least flexible RBF.



15. Click **OK** to save you change.
16. In the Score section of the panel, change the Hidden Units property to **Yes**. This outputs the final hidden unit activation values so that they can be plotted.



17. Add a **SAS Code** node to the diagram and connect it to the output of the **Neural Network** node.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

18. In the SAS Code node's properties panel, click the ellipsis (**…**) next to the **Code Editor** field.
19. In the Training Code editor, right-click in the white space and select **Open** from the menu.
20. Select the program **c2s3d1.sas** (in d:\workshop\winsas\dmnn41). The program is detailed below.

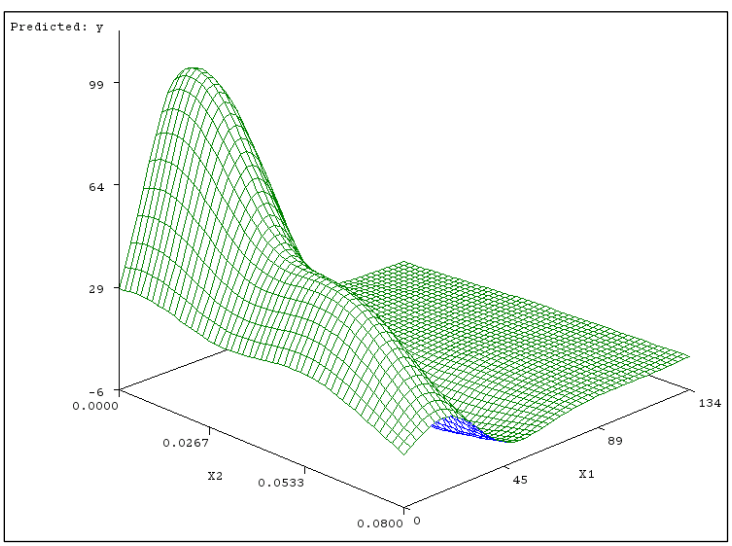
```
options nodate nonumber;
goptions reset=all device=actximg;

proc g3grid data=&EM_IMPORT_DATA out=results;
   grid x1*x2=p_y r_y h11 h12 h13/naxis1=50 naxis2=50 spline;
run;

proc g3d data=results;
   plot x1*x2=p_y/rotate=315 description="plot 1";
   plot x1*x2=r_y/rotate=315 description="plot 2";
   plot x1*x2=h11/rotate=315 description="plot 3";
   plot x1*x2=h12/rotate=315 description="plot 4";
   plot x1*x2=h13/rotate=315 description="plot 5";
run;
```

The G3GRID procedure interpolates (using splines) intermediate data values that facilitate the plots. The G3D procedure is then called to generate the following three-dimensional plots:

- the predicted value (**p_y**) against the two inputs (**x1** and **x2**)
- the residual value (**r_y**) against **x1** and **x2**
- each of the three hidden unit activation values (**h11** to **h13**) against **x1** and **x2**

21. Run the diagram from within the SAS Code node by clicking the running man icon.
22. Confirm the run, and click **Results** when training is complete.
23. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** and maximize to see the requested plots.

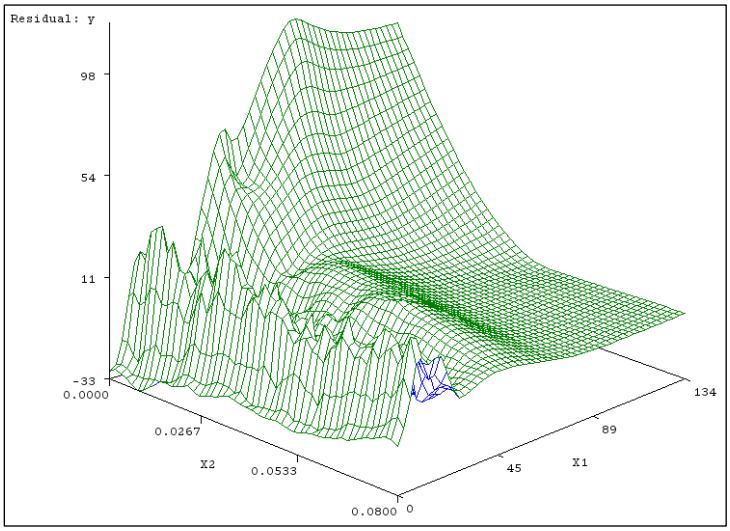
    The first plot is the network's predicted fit using the highly restrictive three hidden-unit ordinary radial basis function equal height variant. The fit to the target distribution is not a good one.



Although some of the ridge in the target distribution has been reproduced, not only is the reproduced ridge "bumpy," the remainder of the target distribution is missing altogether.

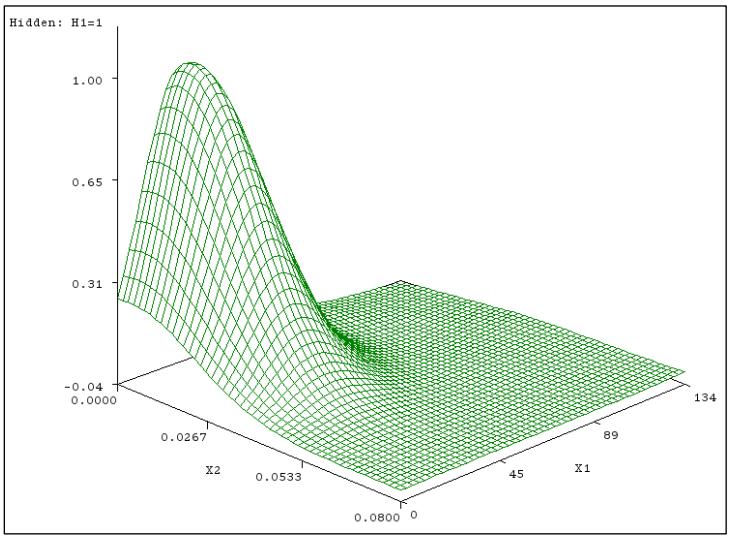
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

24. The poor fit is confirmed by the residual plot. (Click the right arrow to see the residual plot.)



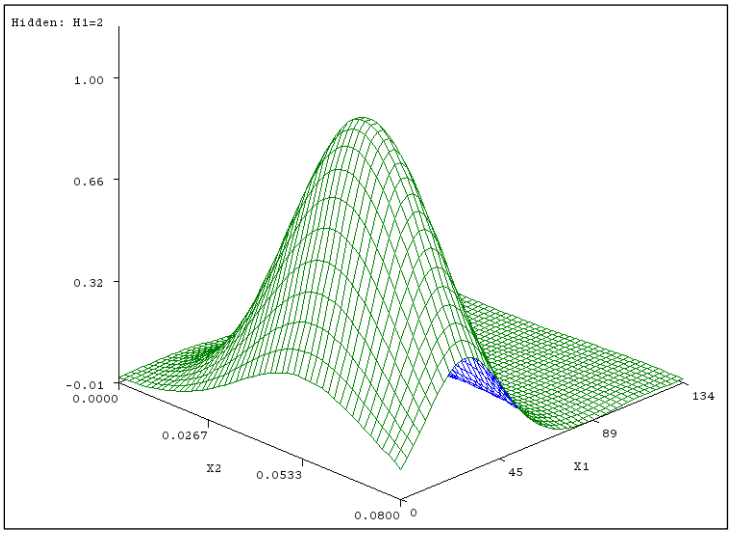
A substantial amount of the target distribution was left behind as residual information (error). This is particularly true when the value of **x1** is large and the value of **x2** is small.

25. Click the right arrow again to reveal the contribution of the first hidden unit (**H11**). The hidden unit appears to be tasked with generating the target distribution ridge at small magnitudes of **x1** and **x2**.



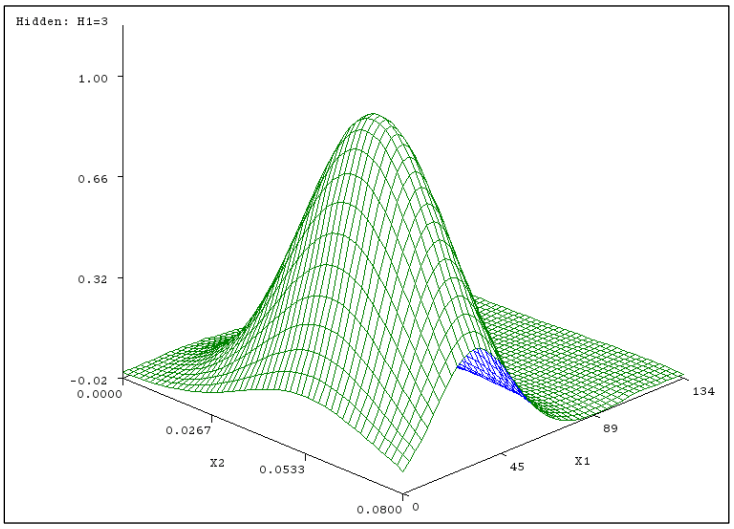
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

26. To see the activation plot of the second hidden unit (**H12**), click the right arrow.



The Gaussian generated by the second hidden unit (**H12**) is very clear. It seems to be tasked with fitting the middle portion of the ridge of the target distribution.

27. Clicking the right arrow again reveals the activation of the third hidden unit (**H13**). The output activation of **H13** appears similar to that of the first hidden unit (**H12**).

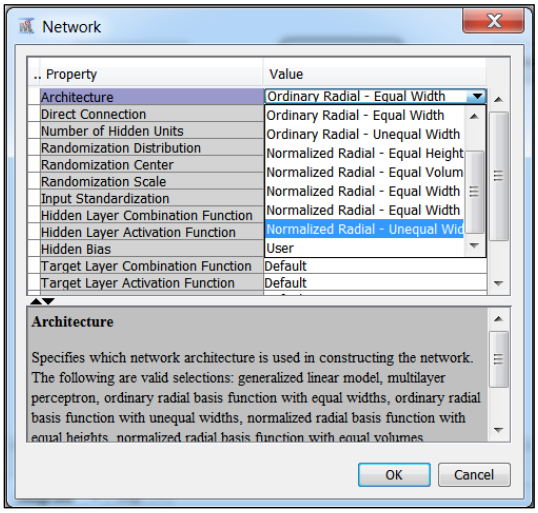


28. Close the Results window and the SAS Code node.

29. Activate the Neural Network properties panel, and click the ellipsis (**...**) next to **Network**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

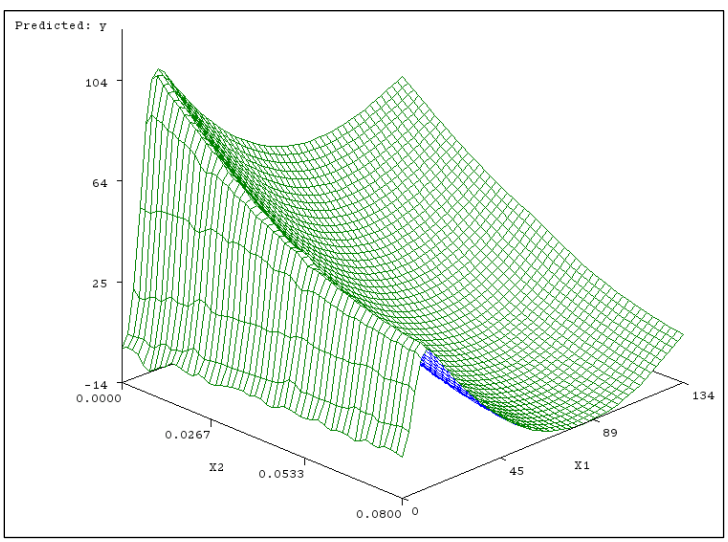
30. Change the architecture to **Normalized Radial – Unequal Width and Height**. This is the most flexible normalized radial basis function architecture.



31. Click **OK** to save the change.

32. Right-click the **SAS Code** node and rerun it. When the run is completed, examine the results.

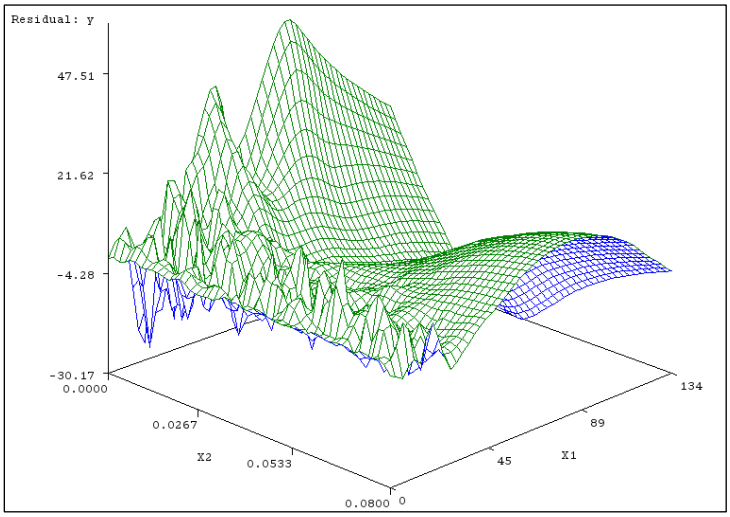
33. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** and maximize to see the requested plots. The first plot shown is the predicted fit of the Normalized Radial – Unequal Width and Height network.



The fit to the target distribution is much better. This is confirmed by the residual plot.

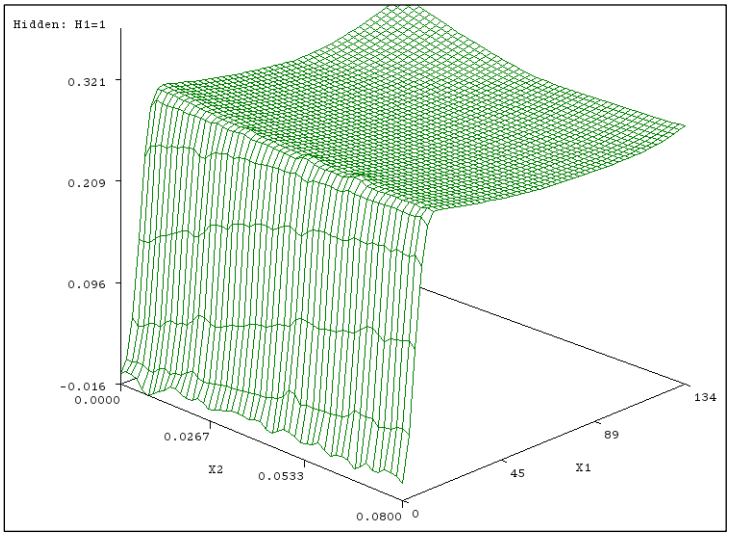
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

34. Click the right arrow to see the residual plot.



There are still minor fluctuations in the fit across all values of **x2**, at small values of **x1**. And there is also still substantial error when **x1** is small and **x2** is large, although less than the amount of error that was produced by the ORBF variant. Otherwise, the residual is fairly flat around the zero value.

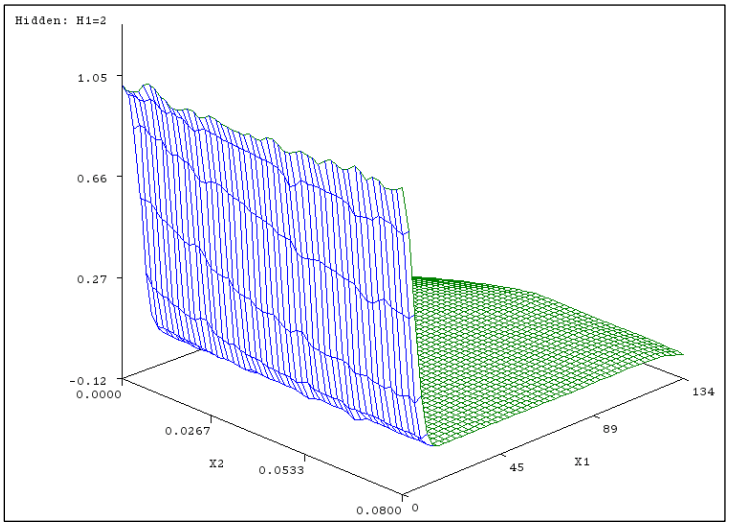
35. Click the right arrow to see the output activation of the first hidden unit (**H11**).



The basis function is clearly no longer a Gaussian. The first hidden unit (**H11**) appears to have captured much of the rise across **x2** at low values of **x1**. There is also a small activation upturn in the upper right corner, i.e. in the region defined by large values of **x1** and small values of **x2**.

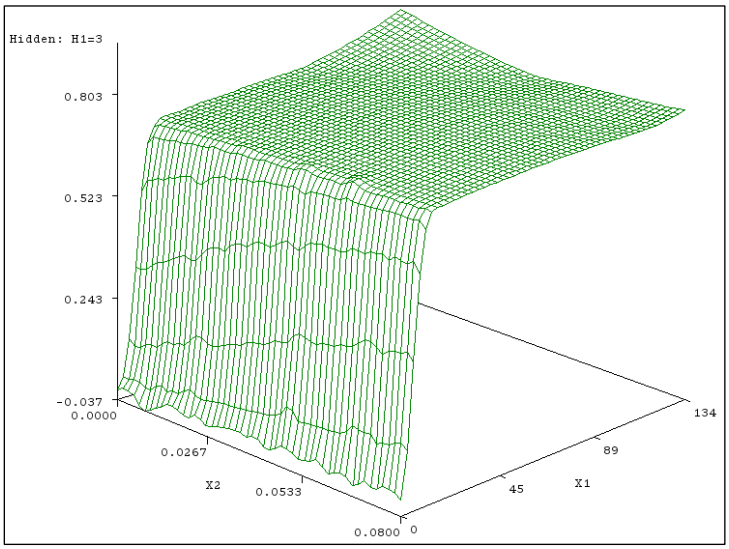
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

36. Clicking the right arrow again reveals the contribution of the second hidden unit (**H12**).



This hidden unit seems to be responsible from lowering the magnitude of the response to the left of the ridge. Remember, the basis functions are added together to produce the final fit. Thus, the small activation magnitudes reduce the high activation generated by the first hidden unit (**H11**).

37. Click the right arrow to see the third hidden unit's (**H13**) output activation.



The activation generated by the third hidden unit (**H13**) appears to be identical to that output by **H11**.

38. Close the Results windows and the SAS Code node.

39. Close the "basis functions" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.03 Poll

Radial basis functions train much faster and are less likely to fall into local minima than a multilayer perceptron.

- ❍ True
- ❍ False

35

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Exercises

2. Comparing Architectures

The **TITANIC** data set, used in this exercise, consists of several variables that describe the passengers on board the ill-fated RMS Titanic, which sank on its maiden voyage in April 1912.

The variables in the **TITANIC** data source are listed below.

Variable	Description	Role	Level
Age	Age of the passenger in years	Input	Interval
Class	Ticket class (1, 2, or 3)	Input	Ordinal
Fare	Ticket fare	Input	Interval
Gender	Gender of the passenger (male, female)	Input	Binary
Name	Passenger's name	ID	Nominal
Survived	Passenger's survival status (1=survived, 0= died)	Target	Binary

a. Right-click **Diagrams** in the project panel and select **Create Diagram**.

b. Enter **exercise 2.2** in the **Diagram Name** field, and click **OK**.

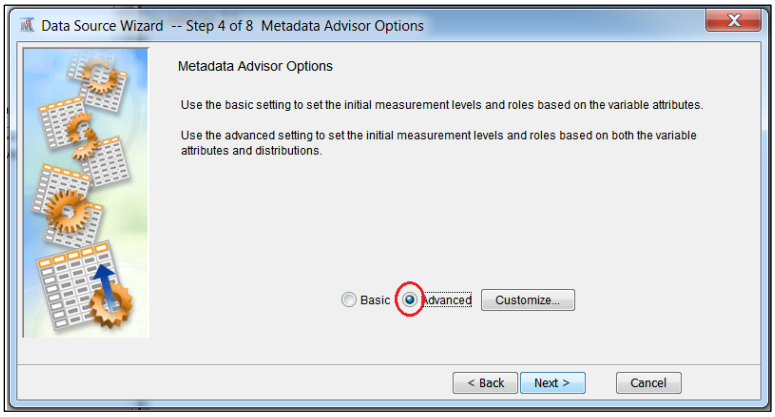
c. Right-click **Data Sources** in the project panel (upper left corner) and select **Create Data Source**. Click **Next** to use a SAS table as the metadata source.

d. Click **Browse** and select **TITANIC** from the **dmnn41** library. Click **OK**.

e. Click **Next**. You see that there are 6 variables and 1309 observations in the **TITANIC** data set.

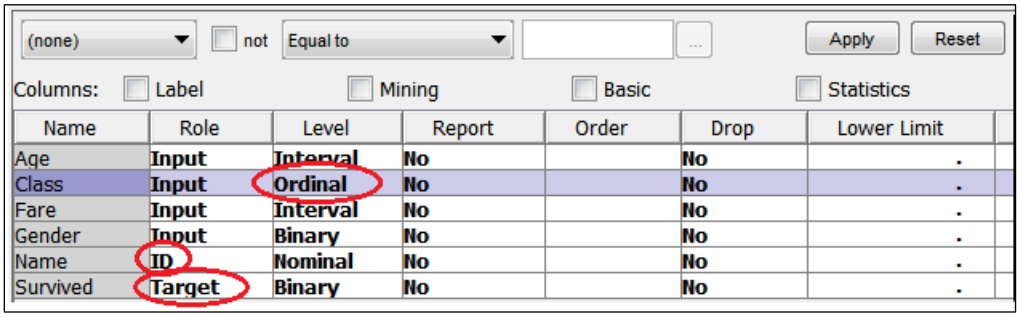
f. Click **Next** again to start the metadata definition process.

g. Select the **Advanced** radio button and then click **Next** to proceed.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

h. Change the role of **Survived** to **Target**. Also change the role of **Name** to **ID**. Finally, change the level of the **Class** variable to **Ordinal**.



Name	Role	Level	Report	Order	Drop	Lower Limit
Age	Input	Interval	No		No	.
Class	Input	Ordinal	No		No	.
Fare	Input	Interval	No		No	.
Gender	Input	Binary	No		No	.
Name	ID	Nominal	No		No	.
Survived	Target	Binary	No		No	.

i. Click **Next** four times and then click **Finish** to complete the metadata definition process.

j. Drag and drop the **TITANIC** data source onto the diagram.

k. Add a **Neural Network** node (Model tab) and connect it to the **TITANIC** data source.

l. Click the ellipsis (**…**) next to **Network**.

m. Change the architecture to **Generalized Linear Model**.

n. Click **OK** to save the change.

o. Right-click the **Neural Network** node and select **Rename**. Name the node **GLM**.

p. Add another **Neural Network** node and connect it to the **TITANIC** data source.

q. Rename the new node **MLP**.

Because MLP is the Enterprise Miner Neural Network node's default architecture, no changes need be made.

r. Add a **Neural Network** node and connect it to the **TITANIC** data source. Rename it **ORBFEQ**.

s. Click the ellipsis (**…**) next to **Network**.

t. Change the architecture to **Ordinary Radial-Equal Width**.

u. Click **OK** to save the change.

v. Add a **Neural Network** node and connect it to the **TITANIC** data source. Rename it **ORBFUN**.

w. Click the ellipsis (**…**) next to **Network** in the new node.

x. Change the architecture to **Ordinary Radial-Unequal Width**.

y. Click **OK** to save the change.

z. Add a **Neural Network** node and connect it to the **TITANIC** data source. Rename it **NRBFEH**.

aa. Click the ellipsis (**…**) next to **Network** in the new node.

bb. Change the architecture to **Normalized Radial–Equal Height**.

cc. Click **OK** to save the change.

dd. Add a **Neural Network** node and connect it to the **TITANIC** data source. Rename it **NRBFEV**.

ee. Click the ellipsis (**…**) next to **Network** in the new node.

ff. Change the architecture to **Normalized Radial–Equal Volumes**.

gg. Click **OK** to save the change.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

hh. Add a **Neural Network** node and connect it to the **TITANIC** data source. Rename it **NRBFEW**.

ii. Click the ellipsis (**…**) next to **Network** in the new node,

jj. Change the architecture to **Normalized Radial–Equal Width**.

kk. Click **OK** to save the change.

ll. Add a **Neural Network** node and connect it to the **TITANIC** data source. Rename it **NRBFEQ**.

mm. Click the ellipsis (**…**) next to **Network** in the new node.

nn. Change the architecture to **Normalized Radial–Equal Width and Height**.

oo. Click **OK** to save the change.

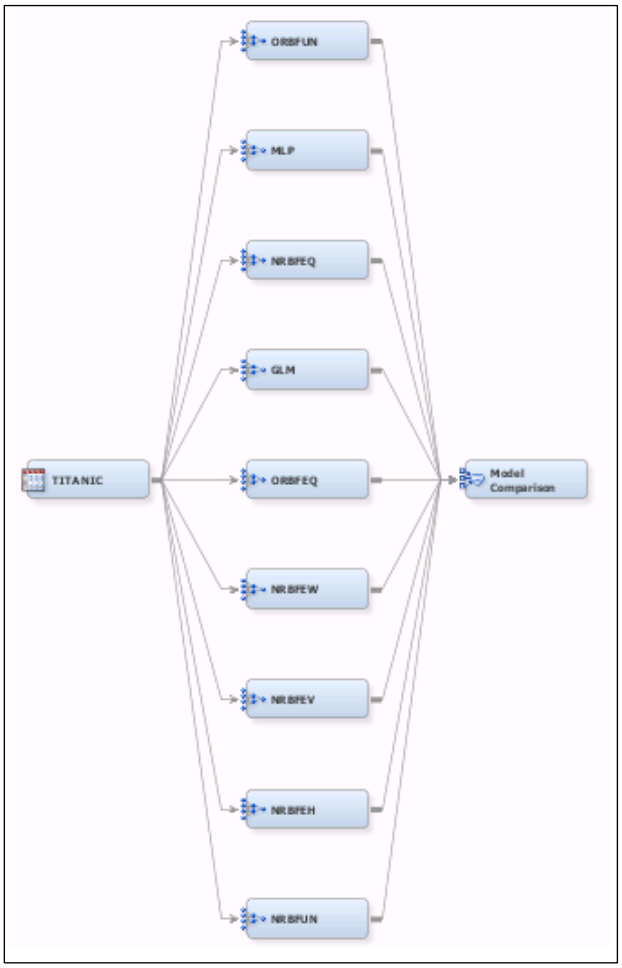
pp. Add a **Neural Network** node and connect it to the **TITANIC** data source. Rename it **NRBFUN**.

qq. Click the ellipsis (**…**) next to **Network** in the new node.

rr. Change the architecture to **Normalized Radial–Unequal Width and Height**.

ss. Click **OK** to save the change.

tt. Drag and drop a **Model Comparison** node (Assess tab) onto the diagram and feed all of the Neural Network nodes into it. Your diagram should look similar to the one below:



uu. Right-click the **Model Comparison** node and select **Run** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

vv. Confirm the run and examine the results when they are available. Complete the following table:

Architecture	Misclassification Rate	Average Squared Error
Generalized Linear Model (GLM)		
Multilayer Perceptron (MLP)		
Ordinary Radial – Equal Widths (ORBFEQ)		
Ordinary Radial – Unequal Widths (ORBFUN)		
Normalized Radial – Equal Height (NRBFEH)		
Normalized Radial – Equal Volumes (NRBFEV)		
Normalized Radial – Equal Width (NRBFEW)		
Normalized Radial – Equal Width and Height (NRBFEQ)		
Normalized Radial – Unequal Width and Height (NRBFUN)		

ww. Close the Model Comparison node's Results window.

xx. Close the "exercise 2.2" diagram.

End of Exercises

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.4 Chapter Summary

PROC NEURAL, the procedure underlying the Neural Network node in SAS Enterprise Miner, is one of the most powerful statistical procedures in SAS. A single neuron implements a generalized linear model:

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

This means that a single neuron can model many common target distributions including Poisson, gamma, binary, multinomial, and ordinal targets. Moreover, there is no need to perform target transformations; you simply fit the desired target distribution by selecting the appropriate error (deviance) function.

But the real power of a neural network is its ability to model nonlinear input-output relationships. The Neural Network node offers two architectures. The first is the multilayer perceptron architecture, in which sigmoids generated by a hidden-layer of neurons are combined to generate the desired fit. The weights and biases control both the orientation and steepness of the component sigmoid activation functions. The linear and nonlinear paradigms can be combined into what is known as a *skip-layer* network. In general, however, adding direct connections from the input layer to the output layer does little to improve a multilayer perceptron's fit. Multilayer perceptrons are already universal approximators; adding direct connections does not make them any more universal.

The second neural network paradigm available in the Neural Network node in SAS Enterprise Miner is the radial basis function architecture. Instead of using sigmoids to inject the essential nonlinearity, radial basis functions use Gaussian activation functions. The hidden unit's weights and biases control the height, width, and orientation of the bell, as well as its location in the input space. There are two variants of radial basis function: ordinary and normalized. Each hidden unit in an ordinary radial basis functions generates a Gaussian centered at the location defined by the hidden unit's weights. Because the bells have limited widths, the hidden units in the ordinary radial basis function architecture have a localized effect. This means that many hidden units might be required to produce a smooth fit. A normalized radial basis function overcomes this problem making each hidden unit's activation a function of the activation of all hidden units (using softmax).

Although the multilayer perceptron is the better known architecture, a radial basis function network offers a number of advantages over its more famous relative. For example, radial basis functions usually train much faster, often an order of magnitude faster than a multilayer perceptron. Radial basis functions also exhibit less of a tendency to fall into local minima.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.5 Solutions

Solutions to Exercises

1. Constructing a Multilayer Perceptron Using PROC NLIN

The complete MODEL statement is given below.

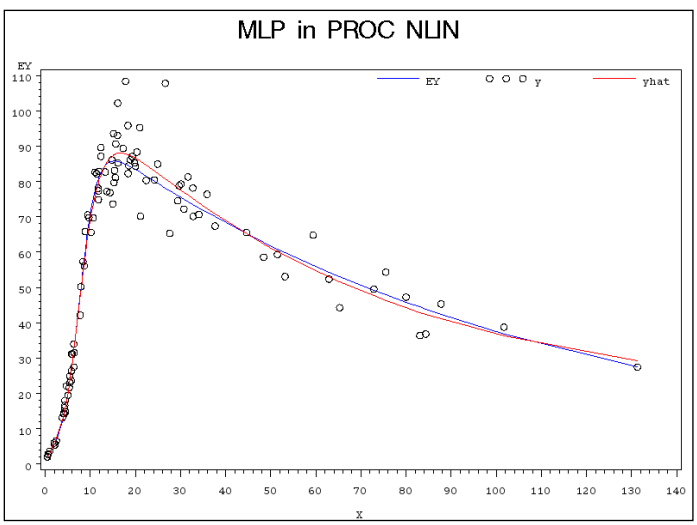
```
model
   y=w0+w1*tanh(w01+w11*x)+w2*tanh(w02+w12*x)+w3*tanh(w03+w13*x);
```

To build a network with three hidden units, three TANH sigmoids are needed. The activation arriving at each hidden unit combines a bias term (**w0i**) with the weighted input (**w1i**) arriving from the single input unit (**x**). The outputs of the sigmoids are weighted (**wi**) and added together. An output bias (**w0**) is then added, giving the predicted value (**y**).

The Summary Statistics information at line **184** shows that the three hidden unit MLP model written using PROC NLIN is significant at the alpha=0.05 level ($p < 0.0001$).

Source	DF	Sum of Squares	Mean Square	F Value	Approx Pr > F
Model	9	85409.0	9489.9	238.24	<.0001
Error	90	3585.0	39.8333		
Corrected Total	99	88993.9			

The scatter plot (reproduced below) confirms the quality of the NLIN procedure's fit.



The prediction of the three hidden unit MLP created using the NLIN procedure (the red line) appears to capture the actual equation (the blue line) quite closely.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2. Comparing Architectures

The completed table is shown below.

Architecture	Misclassification Rate	Average Squared Error
Generalized Linear Model (GLM)	0.236058	0.176292
Multilayer Perceptron (MLP)	0.215432	0.151688
Ordinary Radial – Equal Widths (ORBFEQ)	0.201681	0.152976
Ordinary Radial – Unequal Widths (ORBFUN)	0.213904	0.152300
Normalized Radial – Equal Height (NRBFEH)	0.210848	0.151206
Normalized Radial – Equal Volumes (NRBFEV)	0.209320	0.152304
Normalized Radial – Equal Width (NRBFEW)	0.206264	0.152333
Normalized Radial – Equal Width and Height (NRBFEQ)	0.205500	0.155024
Normalized Radial – Unequal Width and Height (NRBFUN)	0.204736	0.154161

On the **TITANIC** data set, ***all*** of the radial basis function architectures outperformed the multilayer perceptron when misclassification rate is the assessment criterion! This lends credence to the claim that radial basis functions are as powerful as their more famous cousin. The ORBFEQ architecture had the smallest misclassification rate. The GLM (logistic regression) had the largest.

Misclassification rate is the appropriate statistic when the objective is to make a decision. It is also the default comparison criterion when the target variable is binary (as here).

Interestingly, a radial basis function (the Normalized Radial – Equal Height architecture) also had the smallest average square error value. This time the default MLP came second. GLM still generated the worst average squared error performance.

End of Solutions

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Solutions to Student Activities (Polls and Quizzes)

2.01 Poll – Correct Answer

A linear perceptron is a nonlinear model.

- ❍ True
- ❍ False

9

2.02 Poll – Correct Answer

The addition of direct connections between the input and output layers (a skip-layer network) is usually beneficial.

- ❍ True
- ❍ False

22

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2.03 Poll – Correct Answer

Radial basis functions train much faster and are less likely to fall into local minima than a multilayer perceptron.

- ❍ True
- ❍ False

36

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Chapter 3 Learning

3.1 Avoiding Overfitting and Inferior Local Minima 3-3

Demonstration: Early Stopping 3-10

Exercises 3-19

3.2 Parameter Estimation 3-21

Demonstration: Robust Networks 3-28

Exercises 3-32

3.3 Numerical Optimization Methods 3-35

Demonstration: Training Tracks 3-46

3.4 Chapter Summary 3-53

3.5 Solutions 3-54

Solutions to Exercises 3-54

Solutions to Student Activities (Polls and Quizzes) 3-57

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

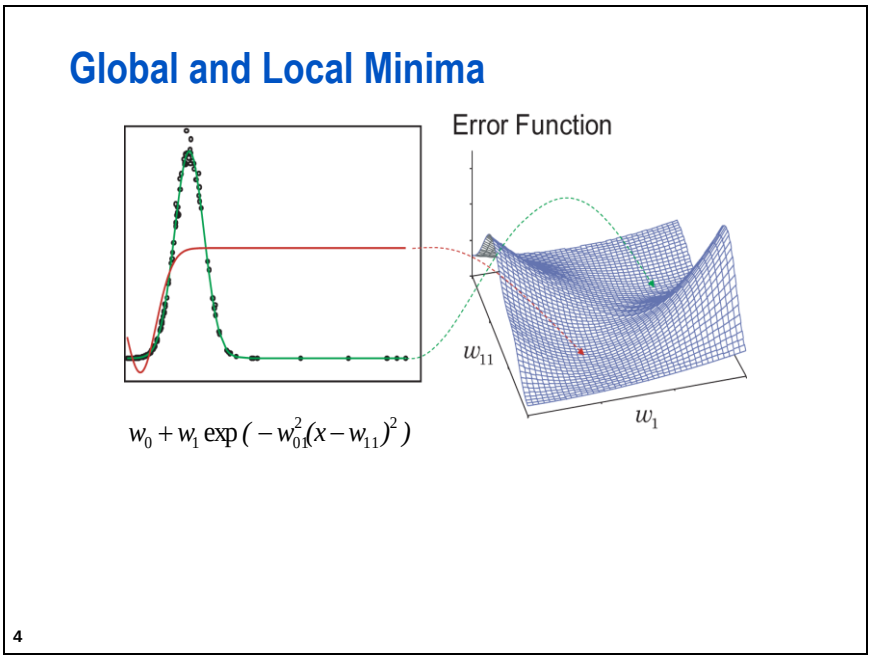
3.1 Avoiding Overfitting and Inferior Local Minima

Objectives

- Describe the problem of local minima.
- Explain the rationale behind the initialization settings.
- Outline how preliminary training and early stopping can be used to help avoid the worst local minima.

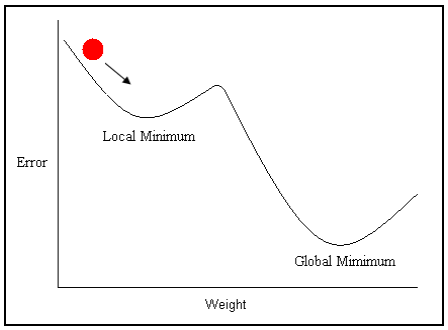
3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



A *global minimum* is a set of weights generating the smallest amount of error. A simple strategy to ensure that a global minimum has been attained is a brute-force search of the parameter space. Unfortunately, the curse of dimensionality quickly makes this method infeasible.

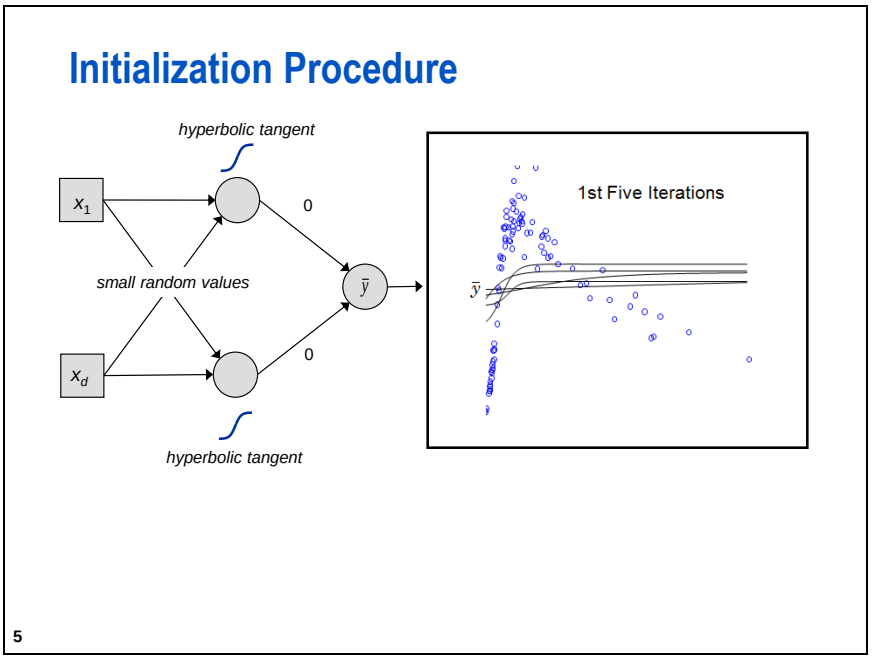
None of the algorithms outlined so far are guaranteed to converge to a global error minimum. Rather, the search optimization algorithms are *heuristic* methods that key on ***local*** features of the error surface when making their decisions. This makes them vulnerable to *local minima*, that is, areas of the error surface generating non-optimal error values. (See the diagram below.)



From the optimization algorithm's perspective, when a local minimum is reached, any movement away from the bottom leads to an increase in error. Because this is unacceptable, the search stops.

Unlike the parabolic error surface of a generalized linear model fit using least squares (which has no local minima), the error surface of a nonlinear model is plagued with local minima. Fortunately, many of these local minima have nearly the global error value. It is only the worst of them that must be avoided.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



One way to avoid the worst local minima is to start with good weight values. In the Neural Network node, this is accomplished by means of the following five-step initialization process:

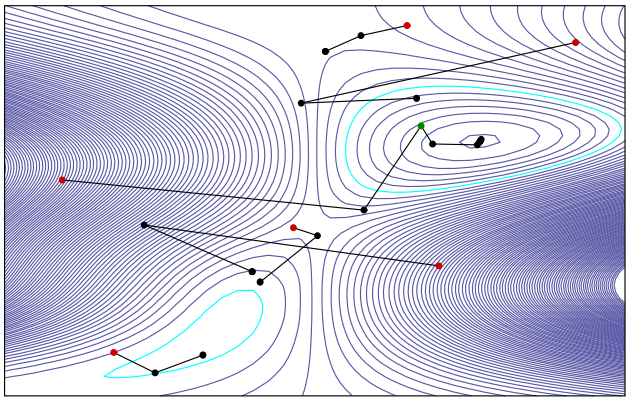
1. Standardize the inputs by subtracting the mean and dividing by the standard deviation (z-score).
2. Set the input-to-hidden weights to a small random number drawn from a specified distribution. Three distributions are available:
   - Normal — All weights are initialized to values that are drawn from the normal distribution with a mean of zero and a standard deviation of 0.1 (default).
   - Uniform — All weights are initialized to values that are drawn from the uniform distribution.
   - Cauchy — All of the weights are initialized to values drawn from a Cauchy distribution.
3. Use the hyperbolic tangent activation function so that the inflection point is at zero. The Elliott or arctangent functions are acceptable alternatives because they also have a zero inflection point.
4. Set the hidden-to-output connection weights to zero.
5. Because the hidden-to-output weights are set to zero in step 4, the output activation is given solely by the output bias on the first iteration. Therefore, the output bias is initialized to the mean of the target.

These initializations help prevent the optimization algorithms from stepping into treacherous regions of the parameter space, that is, regions of the parameter space with many bad local minima.

The mean and standard deviation of input-to-hidden weight random number distribution is given in the **Randomization Center** and **Randomization Scale** fields in the Network menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Preliminary Training



6

Preliminary training is a compromise between an exhaustive search of the parameter space and doing nothing at all. It can substantially improve model fit. Regardless of the training technique that you specify, ***preliminary training should almost always be used*** to help avoid the worst local minima.

The preliminary training algorithm is straightforward:

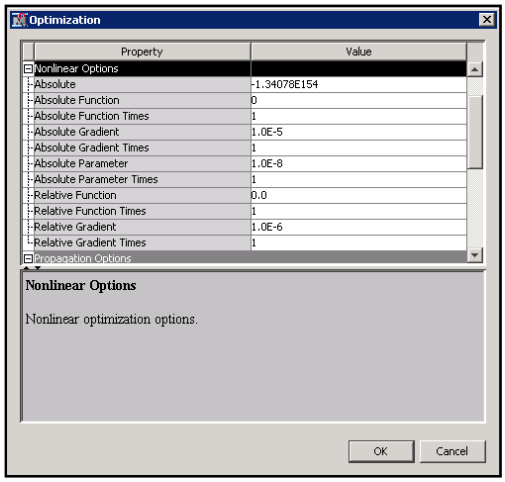
1. Start at a small number (5 by default) of randomly chosen locations in the parameter space and take a few training steps (10 by default) from each, recording each starting location's final error value.

   The number of starting locations is specified by the **Number of Runs** field in the Preliminary Training Options section of the Optimization menu in the Neural Network node's properties panel. The number of steps from each location is specified in the **Maximum Iterations** field.

2. Use the parameter set corresponding to the smallest error value as the starting weights for training.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Convergence Criteria



7

There are three conditions under which convergence is declared:

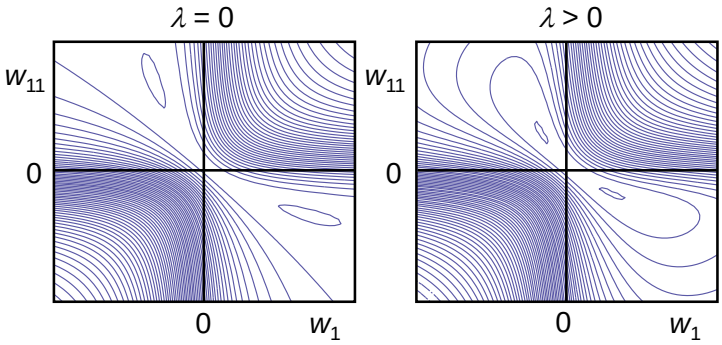
1. When the specified error function stops improving.
2. If the gradient has no slope (implying that a minimum has been reached).
3. If the magnitude of the parameters stops changing substantially.

Because the values have been preset to perform well on a wide range of problems, it is recommended that you do not change the default criteria.

To see the default convergence criteria, click the ellipsis (**…**) next to **Optimization**.

Weight Decay

$$\text{objective function} = \frac{1}{n}\sum_{i=1}^{n}\left(\text{error function}_i + \lambda\|\mathbf{w}_i\|^2\right)$$



8

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Bartlett (1997) demonstrated that generalization depends more on the magnitude of the weights than on the number of weights. Very large magnitude weights tend to generate an irregular fit to the data, as the model adapts to noise (random variation). In other words, large weights are responsible for overfitting.

Weight decay is one way to keep the weights from growing too large. It penalizes large weights by adding a term that is proportional to the sum of squared weights (Bishop 1995). That is,

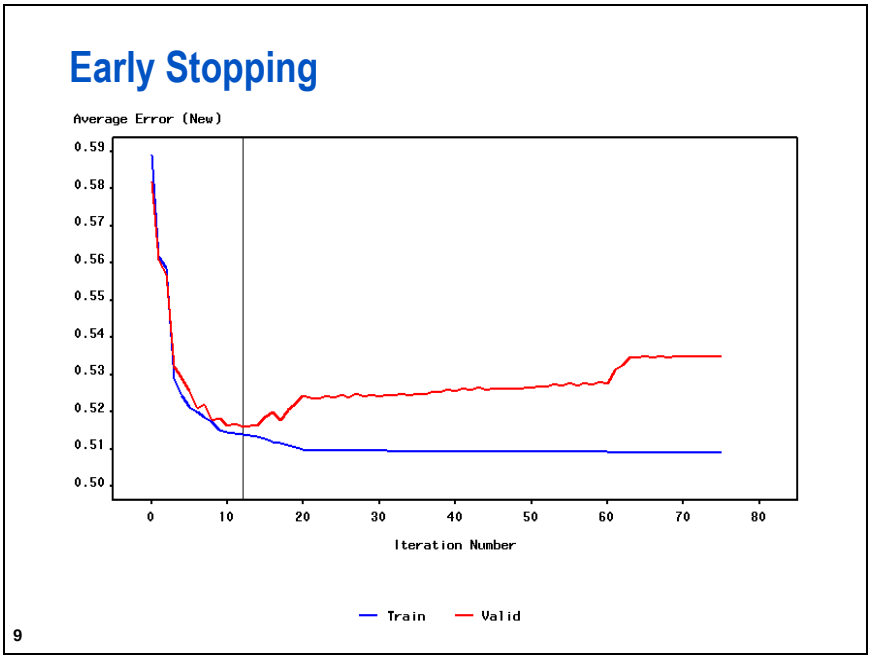
$$\text{objective function} = \frac{1}{n}\sum_{i=1}^{n}\left(\text{error function}_i + \lambda\|\mathbf{w}_i\|^2\right)$$

The decay parameter λ, which can range from zero to one, controls the relative importance of the penalty term. Specifying too large of a penalty term risks the model under-fitting the data. However, as Ripley (1996) points out, the advantages of weight decay far outweigh its risks [emphasis added]:

> Weight decay helps the optimization in several ways. When weight decay terms are included, it is normal to find fewer local minima, and as the objective function is more nearly quadratic, the quasi-Newton and conjugate gradient methods exhibit super-linear convergence in many fewer iterations. ***There seems no reason to ever exclude a regularizer such as weight decay***. (pp. 159-160)

In any event, the influence of the penalty term is usually kept extremely small. In many published studies, the magnitude of the decay parameter (λ) is on the order of 0.000001.

To set the Weight Decay parameter (**default=0**), click the ellipsis (**...**) next to **Network** in the Neural Network node's properties panel, and scroll to the bottom of the window.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Early stopping is another way to keep the weights from growing too large. It is closely related to ridge regression (Sarle 1995). In SAS Enterprise Miner, early stopping is invoked automatically if validation data is available. The procedure is outlined below.

1. Split the data into a training set and a validation set. Often a small (even 10%) validation data table is sufficient to guide early stopping (Potts, W. 2005 – personal communication).

2. Choose an overly complex architecture. For interval variables, a reasonable guess at the number of hidden units required to slightly over fit the data is given by the following equation:

$$h \geq \frac{n-m}{m(k+2)}$$

   Here k is the number of inputs, m is the cases per hidden unit, and n is the overall sample size. With binary targets, the effective sample size is closer to the number in the least frequent class than it is to the overall sample size, n. The correct equation depends on the relative number of event 0 (n_0) and event 1 (n_1) observations (Harrell et al. 1996), that is:

$$h \geq \frac{\min(n_0, n_1) - m}{m(k+2)}$$

3. Start with parameter estimates that are close to zero. (This is handled by the initialization procedure.)

4. Train the neural network. An optimization algorithm that takes small, but efficient, steps is preferred to one that converges quickly. The double dogleg or back propagation algorithms are good choices.

5. Select the parameter estimates that have the best performance on the validation data.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Early Stopping

c3s1d1

This demonstration illustrates the early stopping process. ***Early stopping is one of the easiest ways to build a successful neural network, with little or no parameter tuning. Consequently, it should be one of the first approaches that you consider.***

The **BANKING** data set, which consists of 2000 customers who had an opportunity to acquire a new investment product, is used. The variables in the **BANKING** data set are as follows:

Variable	Description	Role	Level
acquire	Acquired the banking product? (1=Yes, 0=No)	Target	Binary
adbdda	The average daily balance in checking account	Input	Interval
atmct	The number of ATM transactions per month	Input	Interval
atres	Length of time at current residence	Input	Interval
ddadep	Total amount of checking deposits	Input	Interval
ddatot	The total amount of checking transactions	Input	Interval
income	Monthly income	Input	Interval
invest	Amount of investments	Input	Interval
savbal	Savings account balance	Input	Interval

An equal number of responders and non-responders are represented in the sample. However, the true response rate in the population is really only 4%. Unless the scored observations are only to be ranked, prior probability adjustment must be performed.

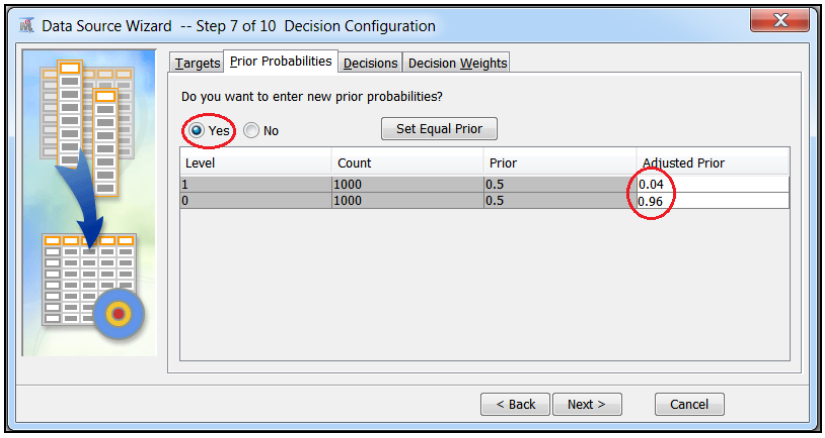
1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **early stopping** in the **Diagram Name** field and click **OK**.
3. Right-click **Data Sources** in the project panel and select **Create Data Source**.
4. Click **Next** to use a SAS table and then click **Browse** to locate the table.
5. Select the **dmnn41** library, and select the **BANKING** data table. Click **OK**.
6. Click **Next** to display selected information about the **BANKING** data. For example, there are 9 variables and 2000 observations in the **BANKING** table.
7. Again, click **Next** to start the metadata definition process.
8. Select the **Advanced** radio button and then click **Next** to set up the variables.
9. Change the role of **ACQUIRE** to **Target**, and click **Next** to proceed.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

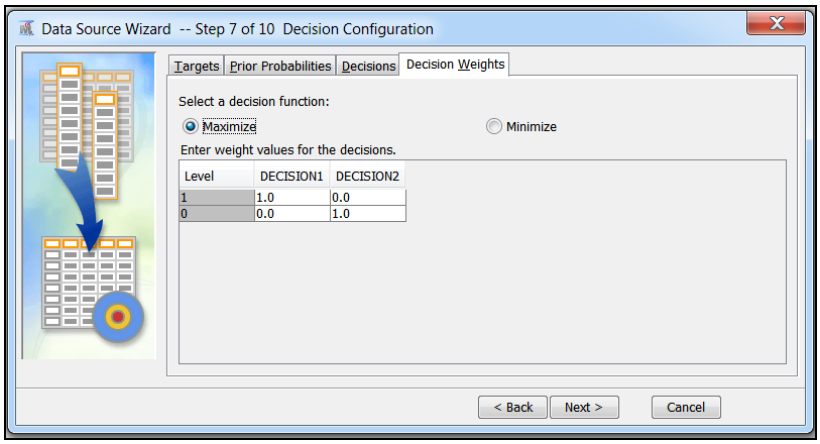
10. Because the real response rate was only 4%, the probabilities must be adjusted for oversampling. Select the **Yes** radio button and then click **Next** to begin defining the decision processing information.

11. Click the **Prior Probabilities** tab and select the **Yes** radio button in answer to the question "Do you want to enter prior probabilities?" A new column appears at the end of the prior probabilities table.

12. Enter the value **0.04** for the Adjusted Prior value of class=1, and enter **0.96** for the class=0 value. This change corrects the probability estimates of ***all*** subsequent models in the diagram.



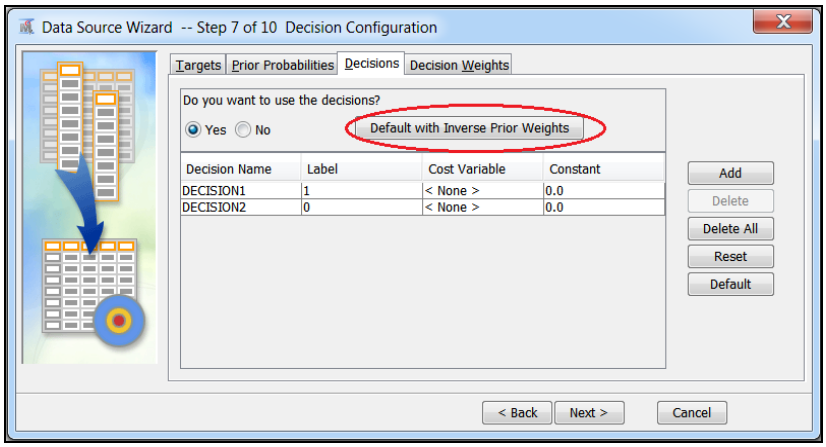
13. Click the **Decision Weights** tab and examine the default profit matrix. (See below.)



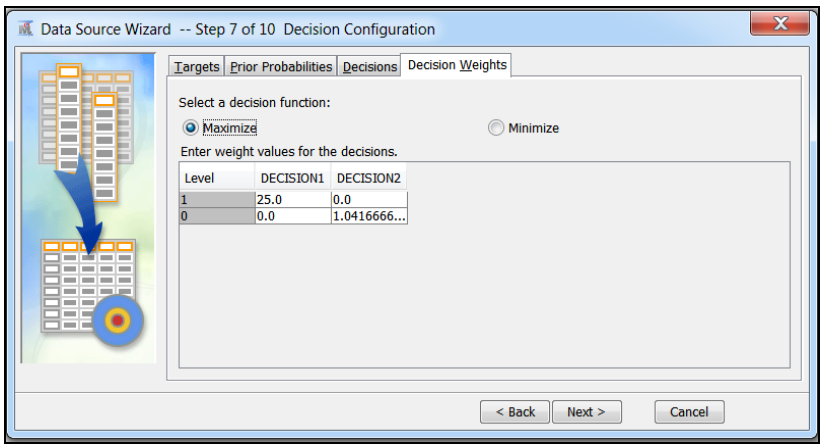
This profit matrix implies that only observations generating a probability greater than 0.5 would be classified as responders. Given that the actual prior probability of responding is only 0.04, few if any cases will be so classified. To correct this problem, the *conforming profit matrix* is used.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

14. Click the **Decisions** tab, and click the **Default with Inverse Prior Weights** button.



15. To see the resulting conforming profit matrix, click the **Decision Weights** tab.



Based on the specified prior probabilities, the correct value for a true positive is 1/0.04=25 and for a true negative is 1/0.96 = 1.04167, so the Bayesian cutoff for the conforming profit matrix is as follows:

$$p > \frac{1}{\left(1+\frac{\delta_{TP}-\delta_{FN}}{\delta_{TN}-\delta_{FP}}\right)} = \frac{1}{\left(1+\frac{25-0}{1.04167-0}\right)} = 0.04$$

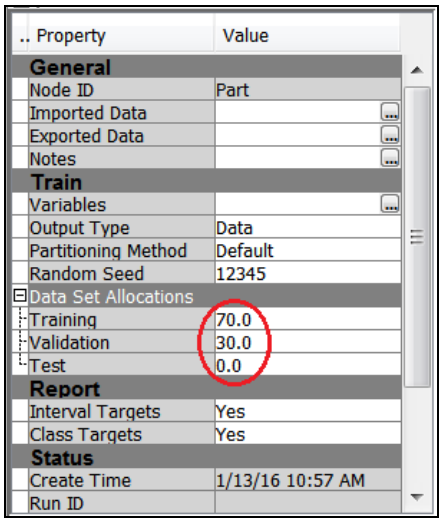
16. Click **Next** three times and then click **Finish** to complete the data source definition.

17. Drag and drop the new **BANKING** data source onto the diagram.

18. Add a **Data Partition** node (Sample tab), and connect it to the **BANKING** data source node.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

19. In the Data Partition node's properties panel, change the split percentages to **70.0** (70%) for Training, **30.0** (30%) for Validation, and **0.0** (0 %) for Test.



20. Add a **SAS Code** node (Utilities tab) to the diagram, and connect it to the **Data Partition** node.

21. Activate the SAS Code node's properties panel, and click the ellipsis (**…**) next to **Code Editor**.

22. In the (default) Training Code editor, right-click in the white space and select **Open** from the menu.

23. Select **c3s1d1.sas** (in d:\workshop\winsas\dmnn41). The program estimates the number of hidden units required to slightly over-fit a data set when the target is either interval-level or binary.

```
%macro nhidden(dsn=,m=10);
   data _null_;
      call symputx('k',length("%cmpres(%EM_INPUT)")-
         length(compress("%cmpres(%EM_INPUT)"))+1);
   run;
   %if %EM_TARGET_LEVEL=BINARY %then %do;    /* binary target */
      data result;
         set &dsn end=eof;
         if %EM_TARGET=1 then n1+1;
         else n0+1;
         if eof then do;
            if n0 < n1 then hidden = ceil((n0-&m)/(&m*(&k+2)));
            else hidden = ceil((n1-&m)/(&m*(&k+2)));
            output;
         end;
      run;
   %end;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
   %else %do;                                /* interval target */
      data result;
         set &dsn end=eof;
         if eof then do;
            hidden = ceil((_n_-&m)/(&m*(&k+2)));
            output;
         end;
      run;
   %end;
   proc print data=result label noobs;
      label hidden='Estimated Number of Hidden Units';
      format hidden 22.0;
      var hidden;
   run;
%mend nhidden;

%nhidden(dsn=&EM_IMPORT_DATA,m=25)
```

The above algorithm does ***not*** provide an estimate of the number of hidden unit to slightly over-fit the data, if the target is multinomial or ordinal.

The macro first determines how many input variables there are and then stores this value as the system variable **k**. It then checks the target type. For binary targets, the effective sample size is closer to the number in the least frequent class than it is to the overall sample size, *n*. The correct equation depends on the relative number of event 0 (n_0) and event 1 (n_1) observations (Harrell et al. 1996).

The M= argument specifies the number of cases per parameter to be used when estimating. Harrell (1996) recommended that you have at least 10 cases for each parameter to be estimated. In this demonstration, 25 cases will be used to estimate each parameter.

24. Run the diagram in the SAS Code node. Save the program, when asked.

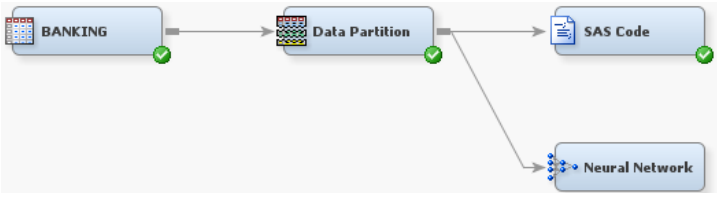
25. When the program has finished, examine the results. Scroll to line **25**.

```
      Estimated Number
       of Hidden Units

                     3
```

The results suggest that three hidden units should slightly over-fit the **BANKING** training data. Because three hidden units is the system default, there is no need to make a change.

26. Close the Results window. Also close the Training Code Editor window.

27. Go to the Model tab and drag and drop a **Neural Network** node onto the diagram. Connect it to the **Data Partition** node.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

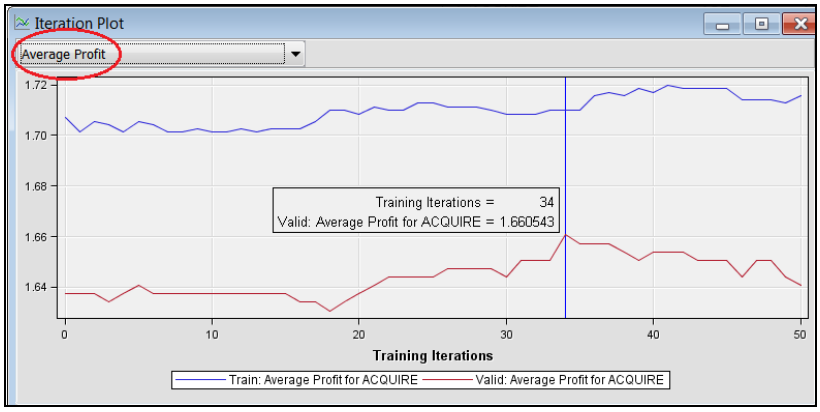
28. Make sure that the Neural Network properties panel is active, and click the ellipsis (**...**) next to the **Optimization** field (in the Train section).

29. Change the Training Technique property to **DBLDog** (double dogleg). Click **OK** to save your change.

    A training technique that produces gradual improvements over many steps, like double dogleg or back propagation, is preferable to one that converges quickly.

30. Run the diagram from the Neural Network node. Examine the results when training is complete.

31. Maximize the Iteration Plot window (lower left corner). To see the Profit/Loss plot, click the down arrow and select **Average Profit**.



The selected iteration (34) corresponds to the maximum average validation profit ($1.66).

32. Minimize the Iteration Plot window, and maximize the Fit Statistics panel (upper right corner). A number of useful statistics appear in this table. Two measures of particular note are listed below.

Fit Statistics

Target	Target Label	Fit Statistics	Statistics Label	Train	Validation	Test
ACQUIRE		DFT	Total Degrees of Freedom	1399	.	.
ACQUIRE		DFE	Degrees of Freedom for Error	1368	.	.
ACQUIRE		DFM	Model Degrees of Freedom	31	.	.
ACQUIRE		NW	Number of Estimated Weights	31	.	.
ACQUIRE		AIC	Akaike's Information Criterion	1034.638	.	.
ACQUIRE		SBC	Schwarz's Bayesian Criterion	1197.187	.	.
ACQUIRE		ASE	Average Squared Error	0.108557	0.136975	.
ACQUIRE		MAX	Maximum Absolute Error	0.975412	0.999699	.
ACQUIRE		DIV	Divisor for ASE	2798	1202	.
ACQUIRE		NOBS	Sum of Frequencies	1399	601	.
ACQUIRE		RASE	Root Average Squared Error	0.32948	0.370102	.
ACQUIRE		SSE	Sum of Squared Errors	303.7432	164.6443	.
ACQUIRE		SUMW	Sum of Case Weights Times Freq	2798	1202	.
ACQUIRE		FPE	Final Prediction Error	0.113477	.	.
ACQUIRE		MSE	Mean Squared Error	0.111017	0.136975	.
ACQUIRE		RFPE	Root Final Prediction Error	0.336864	.	.
ACQUIRE		RMSE	Root Mean Squared Error	0.333193	0.370102	.
ACQUIRE		AVERR	Average Error Function	0.347619	0.469736	.
ACQUIRE		ERR	Error Function	972.6384	564.623	.
ACQUIRE		MISC	Misclassification Rate	0.316655	0.351082	.
ACQUIRE		WRONG	Number of Wrong Classifications	443	211	.
ACQUIRE		PROF	Total Profit for ACQUIRE	2392.007	998.046	.

The network produced by early stopping generated an average squared error on the validation data of just under 14%. The validation misclassification rate was about 35%.

33. Close the Neural Network node's Results window.

34. To assess how well the network performs against novel data, create a test data source by right-clicking **Data Sources** and selecting **Create Data Source**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

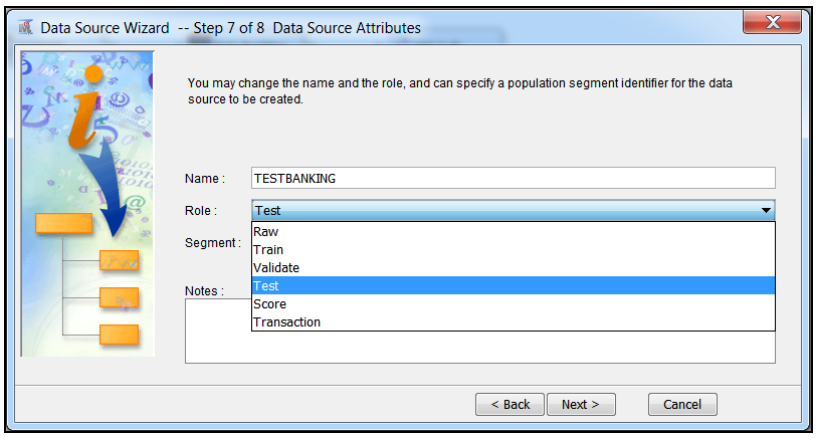
35. Click **Next** to use a SAS table as the metadata source, and click **Browse** to choose a SAS table from the available libraries.

36. Select the **dmnn41** library, and select the **TESTBANKING** table.

37. Click **Next** to display information about the table. It has 9 variables, the same variables that were described when the **BANKING** data source was defined, and 1000 observations.

38. Click **Next** three times to proceed to the final step in the metadata definition process.

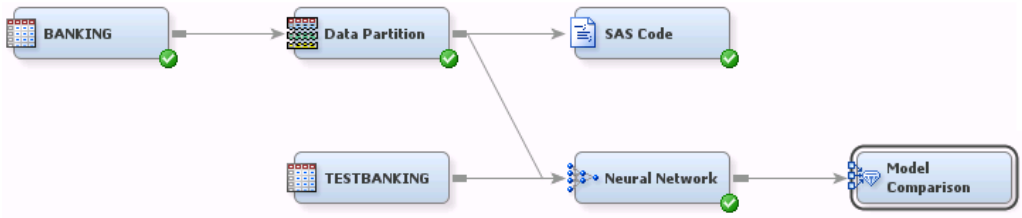
39. Change the role of the **TESTBANKING** data source to **Test**.



40. Click **Next** and then **Finish** to complete the metadata definition process.

41. Drag the **TESTBANKING** data source onto the diagram. Connect it to the **Neural Network** node.

42. Finally, add a **Model Comparison** node to the diagram. Your flow should now look similar to the one shown below.



43. Right-click the **Model Comparison** node, and select **Run** from the pop-up menu.

44. Confirm the run. When the assessment is complete, click **Results**.

45. Maximize the Output window (lower right corner) and scroll to line **131** to see the fit information.

```
                                   Data Role=Test

Statistics                                                        Neural

Test:  Kolmogorov-Smirnov Statistic                                 0.71
Test: Average Profit for ACQUIRE                                    1.70
Test: Average Squared Error                                         0.11
Test:  Roc Index                                                    0.92
Test: Average Error Function                                        0.38
Test:  Bin-Based Two-Way Kolmogorov-Smirnov Probability Cutoff      0.07
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
Test: Cumulative Percent Captured Response                          75.20
Test: Percent Captured Response                                     20.40
Test: Divisor for TASE                                            2000.00
Test: Error Function                                               755.07
Test: Gain                                                         651.10
Test:  Gini Coefficient                                              0.84
Test:  Bin-Based Two-Way Kolmogorov-Smirnov Statistic                0.71
Test:  Kolmogorov-Smirnov Probability Cutoff                         0.05
Test: Cumulative Lift                                                7.51
Test: Lift                                                           4.13
Test: Maximum Absolute Error                                         1.00
Test: Misclassification Rate                                         0.32
Test: Lower 95% Conf. Limit for TMISC                                 .
Test: Upper 95% Conf. Limit for TMISC                                 .
Test: Mean Squared Error                                             0.11
Test: Sum of Frequencies                                          1000.00
Test: Total Profit for ACQUIRE                                    1700.00
Test: Root Average Squared Error                                     0.33
Test: Cumulative Percent Response                                   30.04
Test: Percent Response                                              16.52
Test: Root Mean Squared Error                                        0.33
Test: Sum of Squared Errors                                        217.82
Test: Sum of Case Weights Times Freq                              2000.00
Test: Number of Wrong Classifications                              324.00
```

The average squared error on the test data is only 11%, and the misclassification rate is 32%.

Several other important statistics are displayed. For example, the area under the ROC curve is known as the *ROC index* (C statistic). It can be interpreted as the probability that a randomly chosen class=1 case has a greater posterior probability than a randomly chosen class=0 case. Because a perfect model generates an ROC Index value of one, the 0.92 value generated by the Neural Network node using early stopping is rather impressive.

46. Close the Model Comparison node's Results window.

47. Close the "early stopping" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.01 Poll

The purpose of weight decay and early stopping is to keep the network from falling into a bad local minimum.

- ❍ True
- ❍ False

11

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Exercises

1. Network Initialization

In this exercise, you generate and examine the values that the Neural Network node uses to initialize the network. The **FITNESS** data set, described in Chapter 2, will again be used.

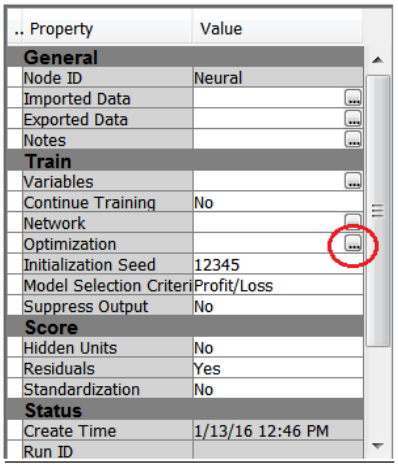
a. Right-click **Diagrams** in the project panel, and select **Create Diagram** from the pop-up menu.

b. Enter **exercise 3.1** in the **Diagram Name** field, and click **OK**.

c. Click **Data Sources** and drag and drop the **FITNESS** data onto the diagram.

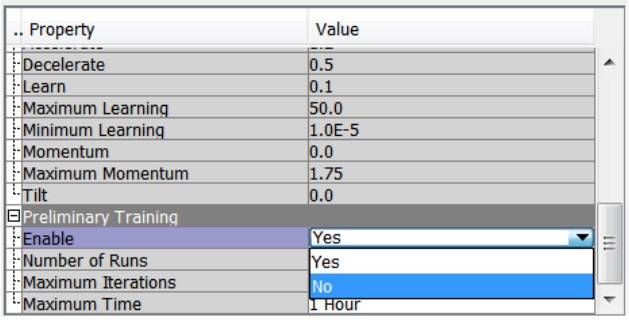
d. Add a **Neural Network** node (Model tab) and connect it to the **FITNESS** data source.

e. Because preliminary training would overwrite the original initialization values, it must be turned off. To turn off preliminary training, click the ellipsis (**…**) next to **Optimization**.



.. Property	Value
General	
Node ID	Neural
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Continue Training	No
Network	
Optimization	
Initialization Seed	12345
Model Selection Criteri	Profit/Loss
Suppress Output	No
Score	
Hidden Units	No
Residuals	Yes
Standardization	No
Status	
Create Time	1/13/16 12:46 PM
Run ID	

f. Scroll down to the Preliminary Training section and change **Enable** to **No**.



.. Property	Value
Decelerate	0.5
Learn	0.1
Maximum Learning	50.0
Minimum Learning	1.0E-5
Momentum	0.0
Maximum Momentum	1.75
Tilt	0.0
Preliminary Training	
Enable	Yes
Number of Runs	
Maximum Iterations	
Maximum Time	1 Hour

Yes
No

g. Click **OK** to save the change.

h. Right-click the **Neural Network** node and select **Run** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

i. Confirm the run when asked, and open the results when training is complete.

j. Maximize the Output window (bottom right corner) and examine the starting weights (line **39**).

1) How are the values of the input-to-hidden unit connections (e.g., **Performance_H11**) and the hidden unit biases (e.g., **BIAS_H11**) determined?
2) Several weights (e.g., **H11_Oxygen_Consumption**) are initialized to zero. What do these weights represent, and why were they set to zero?
3) What is the value of the output unit bias (i.e., **BIAS_Oxygen_Consumption**), and how is its value determined?

k. Close the Neural Network node's Results window.

l. Close the "exercise 3.1" diagram.

End of Exercises

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.2 Parameter Estimation

Objectives

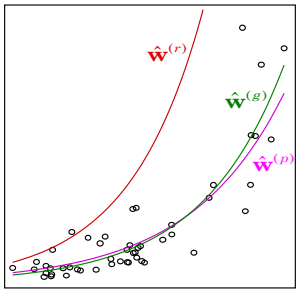
- Describe the parameter estimation methods.
- List the assortment of error functions that are available in the Neural Network node.

16

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Ordinary Least Squares Estimation

$$Q(\mathbf{w}) = (y - \mu(\mathbf{w}))^2$$



$$Q(\hat{\mathbf{w}}^{(r)}) = 1128$$

$$Q(\hat{\mathbf{w}}^{(p)}) = 259$$

$$Q(\hat{\mathbf{w}}^{(g)}) = 236$$

17

Estimating the unknown parameters in a neural network is referred to as *learning*. Each point in the *n*-dimensional parameter space is a possible solution. Unfortunately, the large number of parameters to be estimated caused the early neural network learning algorithms to have serious performance issues. Now numerical optimization methods efficiently guide the search process through vast parameter spaces.

Before an optimization method can be applied, however, a criterion for choosing among the different sets of parameters must be specified. This criterion is known as an *error function*. Many error functions exist. Ordinary least squares (below) is probably the best known.

$$Q(\mathbf{w}) = \sum_{i=1}^{d} \left(y_i - \mu_i(\mathbf{w})\right)^2$$

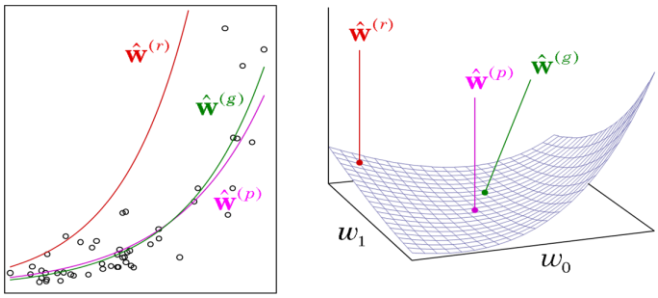
Here y_i is the ith target value, and $\mu_i(\mathbf{w})$ is the model's estimate of the ith target, given weight vector **w**. The choice of the appropriate error function primarily depends on statistical considerations, such as the target distribution. Applying different error functions will typically give different parameter estimates. The goal is to find the parameter set that minimizes the specified error function.

To see the available error functions, click the ellipsis (**...**) next to **Network**, and expand the **Target Layer Error Function** field.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Maximum Likelihood Estimation

$$Q(\mathbf{w}) = 0.5\left[\left(\frac{y-\mu(\mathbf{w})}{\vartheta}\right)^2 + \ln(2\pi)\right] + \ln(\vartheta)$$



18

Many error functions are motivated by the method of *maximum likelihood*. Discovered by Fisher in 1922, maximum likelihood estimates have a number of very desirable properties, such as asymptotic efficiency and consistency. The error function minimized is the negative log likelihood. To use maximum likelihood estimation, the distribution of the target needs to be specified. Assuming that the cases are independent, the likelihood is given by taking the product of the individual probability density functions. Thus, if y is the target value and $\mu(\mathbf{w})$ is its expected target value, then the likelihood is given by the joint probability of the observed data expressed as a function of the parameters:

$$Q(\mathbf{w}) = -\ln\left(\prod_{i=1}^{n} pdf\,(y_i, \mu(\mathbf{w}))\right)$$

The maximum likelihood estimates are located at the mode of this distribution. Consequently, they are the parameter values with the highest probability of having generated the observed data.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Deviance

$$Q(\mathbf{w}) = 2\phi\left[\ln\left(l_{staturated}\right) - \ln(l(\mathbf{w}))\right]$$

Distribution	Deviance Measure
Normal	$Q(\mathbf{w}) = \sum (y - \mu(\mathbf{w}))^2$
Poisson	$Q(\mathbf{w}) = 2\sum \left[y \ln(y / \mu(\mathbf{w})) - (y - \mu(\mathbf{w}))\right]$
Gamma	$Q(\mathbf{w}) = 2\sum \left[-\ln(y / \mu(\mathbf{w})) + (y - \mu(\mathbf{w})) / \mu(\mathbf{w})\right]$
Bernoulli	$Q(\mathbf{w}) = -2\sum \left[y \ln(\mu(\mathbf{w})) + (1 - y) \ln(1 - \mu(\mathbf{w}))\right]$

19

If the probability distribution is a member of the exponential family, minimizing deviance is equivalent to maximizing likelihood. But deviance offers the advantage that it does not require a probability density function, which makes the calculation of deviance more efficient. Deviance also offers other numerical advantages. For example, the deviance measures are automatically scaled.

The default method for fitting an interval target is a *normal* distribution. The deviance measure used to fit a normal distribution is the familiar ordinary least squares equation (below):

$$Q(\mathbf{w}) = \sum (y - \mu(\mathbf{w}))^2$$

A *Poisson* distribution is usually thought of as the appropriate distribution for count data. Because the variance is proportional to the mean, the deviance function for a Poisson distribution is the following:

$$Q(\mathbf{w}) = 2\sum [y \ln(y / \mu(\mathbf{w})) - (y - \mu(\mathbf{w}))]$$

In a *gamma* distribution the variance is proportional to the square of the mean. It is often used when the target represents an amount. The gamma deviance function is given by the following:

$$Q(\mathbf{w}) = 2\sum \left[-\ln(y / \mu(\mathbf{w})) + (y - \mu(\mathbf{w})) / \mu(\mathbf{w})\right]$$

The default method for fitting a binary target (logistic model) is *Bernoulli* deviance:

$$Q(\mathbf{w}) = -2\sum [y \ln(\mu(\mathbf{w})) + (1 - y) \ln(1 - \mu(\mathbf{w}))]$$

It is the same as a binomial distribution with one trial.

Both Poisson and gamma distributions require that the random variables are nonnegative. Consequently, these deviance functions must be coupled with an activation function that constrains the expected target to be nonnegative (e.g., exponential activation).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Other Available Error Functions (Self Study)

Cauchy (Lorenz) Distribution

Cauchy is most often used when you want to predict the approximate conditional mode instead of the conditional mean. The distributed weights favor the most frequent values in the Cauchy distribution. If the target is Cauchy distributed, then the following deviance estimate is appropriate:

$$Q(\mathbf{w}) = \ln\left\{\pi\sigma\left[\left(\frac{y-\mu(\mathbf{w})}{\sigma}\right)^2 + 1\right]\right\}$$

Entropy

Cross or relative entropy is for independent interval targets with values between zero and one (inclusive). Identical to the Bernoulli distribution if the target is binary, it offers some advantages over the Bernoulli distribution when the data are proportions. The entropy deviance estimate is given by the following:

$$Q(\mathbf{w}) = 2\left[y\ln\left(\frac{y}{\mu(\mathbf{w})}\right) + (1-y)/\ln\left(\frac{1-y}{1-\mu(\mathbf{w})}\right)\right]$$

Multinomial Distribution

The multinomial distribution is a generalization of the binomial distribution where each trial results in one of a fixed number of k possible outcomes with probabilities $p_1, p_2, \ldots, p_k$. The corresponding multinomial distribution deviance function is defined as follows:

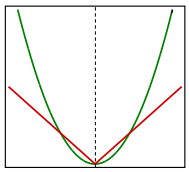
$$Q(\mathbf{w}) = 2y\ln\left(\frac{y}{\mu(\mathbf{w})\,t}\right)$$

Notice that the estimate of the target, $\mu(\mathbf{w})$, is first multiplied by the number of trials, t, before dividing.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

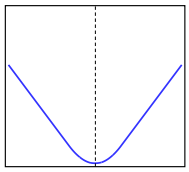
Robust Estimation

$$Q(\mathbf{w}) = \sum_{i=1}^{n} \psi\left(\frac{y_i - \mu_i(\mathbf{w})}{\sigma_y}\right) = \sum_{i=1}^{n} \psi(z_i)$$



$\psi(z) = 0.5z^2$ Normal

$\psi(z) = |z|$ Laplace



$\psi(z) = 0.5z^2$ if $|z| < 1$

$\psi(z) = |z| - 0.5$ if $|z| \geq 1$

Huber's

20

Robust estimation methods are relatively insensitive to extreme values (outliers). There are three robust error estimation techniques available in the Neural Network node:

- Huber's M-estimation (**HUBER**)
- Tukey's bi-weight (**BIWEIGHT**)
- Andrew's wave (**WAVE**)

Although there are merits to each approach, Huber's (1964) M-estimation method is often preferred. Not only is it nearly as efficient as least squares estimation when the data is normally distributed, it is more efficient than least squares when the distribution has relatively more mass concentrated in the tails.

Huber's M-estimate can be thought of as a special case of maximum likelihood estimation where the probability density function is proportional to $\exp(-\psi(z))$. Here z is the standardized residual.

$$z = \frac{y_i - \mu(\mathbf{w})}{\sigma_i}$$

In a normal distribution, the ψ function is the square of z. In a Laplace distribution, the ψ function is the absolute value of z. Huber's M-estimate behaves like least squares when the residuals are small, and like least absolute deviation when the residuals are large.

Although Huber's M-estimator is most often applied to unbounded interval targets with outliers, or to a symmetric distribution with a moderate degree of inequality in the conditional variance, it can also be used with categorical targets when you want to predict the mode rather than the posterior probability.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Activation and Error Function Combinations

Target	Link Function	Activation Function	Error Function
Amounts	Identity	Identity	Normal
	Identity	Identity	Huber
	Log	Exponential	Poisson
	Log	Exponential	Gamma
Names or Grades	Logit	Logistic	Bernoulli
	Generalized Logit	Softmax	MBernoulli
	Cumulative Logit	Logistic (See note.)	MBernoulli
Proportions	Logit	Logistic	Entropy
	Generalized Logit	Softmax	MEntropy

The inverse cumulative logit has no name, so it was called *Logistic*. This is ***not*** the inverse logit-link function.

21

The slide above summarizes the appropriate activation and error function combinations.

If the target is amount (interval) data, there are four possible activation and error combinations, depending on the target distribution. If the target is nominal or ordinal, there are three activation and error function options. If the target constitutes proportions, two activation-error function combinations are available.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Robust Networks

c3s2d1

This demonstration explores the impact of applying robust estimation deviance measures when the data contains severe outliers. The exponential decay data (**DECAY**) is again used. Although the Neural Network node offers several robust error functions, only Huber's M-estimator is explored here.

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **robust networks** in the **Diagram Name** field and click **OK**.
3. Because the **DECAY** data source has already been defined, just drag and drop it onto the diagram.
4. Add a **SAS Code** node (Utility tab) to the diagram. Connect it to the **DECAY** data source.

5. In the SAS Code node's properties panel, click the ellipsis (**…**) next to the **Code Editor** field.
6. In the (default) Training Code editor, right-click in the white space and select **Open** from the menu.
7. Select the program **c3s2d1a.sas** (in d:\workshop\winsas\dmnn41). The program turns observations number 45 and 85 into outliers, and then sorts the data so that it will plot correctly.

```
options nodate nonumber;

data &EM_EXPORT_TRAIN;
   set &EM_IMPORT_DATA;
   if _n_=45 then do;
      x=10*x;
      ey=.;
   end;
   if _n_=85 then do;
      y=y/10;
      ey=.;
   end;
run;

proc sort data=&EM_EXPORT_TRAIN;
   by x;
run;
```

8. Close the SAS Code node and save your changes when asked.
9. Add a **Neural Network** node (Model tab) to the diagram, and connect it to the **SAS Code** node.
10. Add another **SAS Code** node (Utility tab). Connect it to the **Neural Network** node, as shown below.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

11. In the SAS Code (2) node's properties panel, click the ellipsis (**…**) next to **Code Editor**.

12. Select the Training Code editor, right-click in the white space, and select **Open** from the menu.

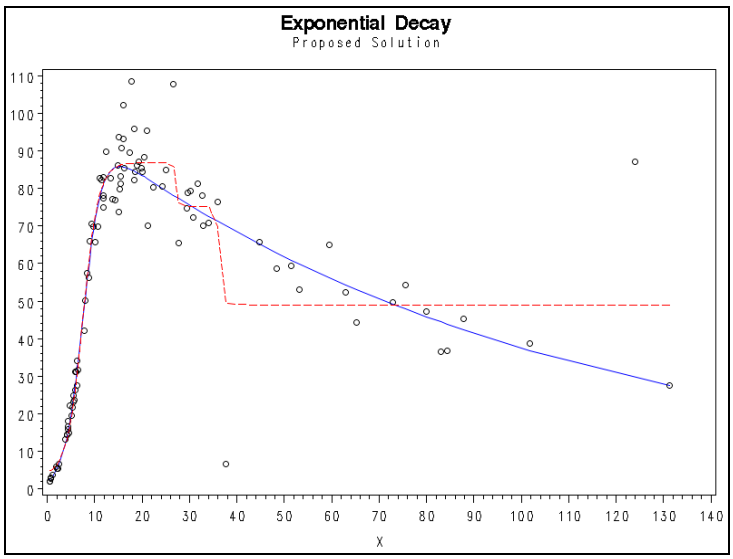
13. Select **c3s2d1b.sas** (in d:\workshop\winsas\dmnn41). The program plots the proposed solution.

```
options nodate nonumber;
goptions reset=all device=actximg;

axis1 label=none;

symbol1 c=b i=join v=none;
symbol2 c=bl i=none v=circle
symbol3 c=r i=join v=none l=3;

title1 h=1.5 f=swissb 'Exponential Decay';
title2 'Proposed Solution';
proc gplot data=&EM_IMPORT_DATA;
   plot ey*x=1 y*x=2 p_y*x=3/ overlay frame vaxis=axis1;
run;
```

14. Run the diagram from within the SAS Code (2) node. Confirm the run when asked.

15. Click **Results** when training is complete.

16. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** to display the proposed solution. Maximize the plot.



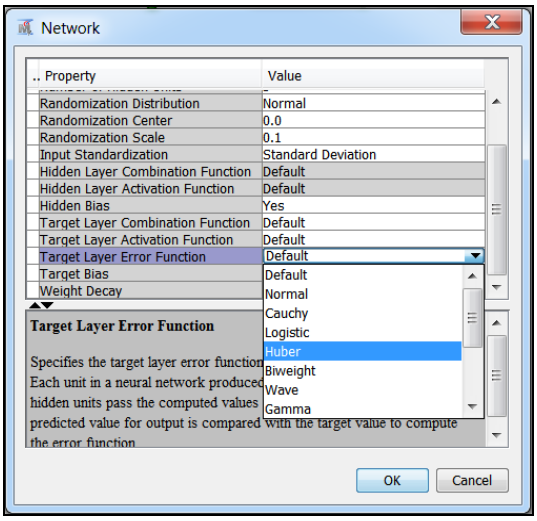
The solution is ***not*** a good one. The outliers have drawn the nonlinear regression line toward them, making the predicted model (the dashed line) deviate badly from the actual model (the solid line).

17. Close the Results window and the SAS Code node.

18. Activate the Neural Network node's properties panel, and click the ellipsis (**…**) next to **Network**.

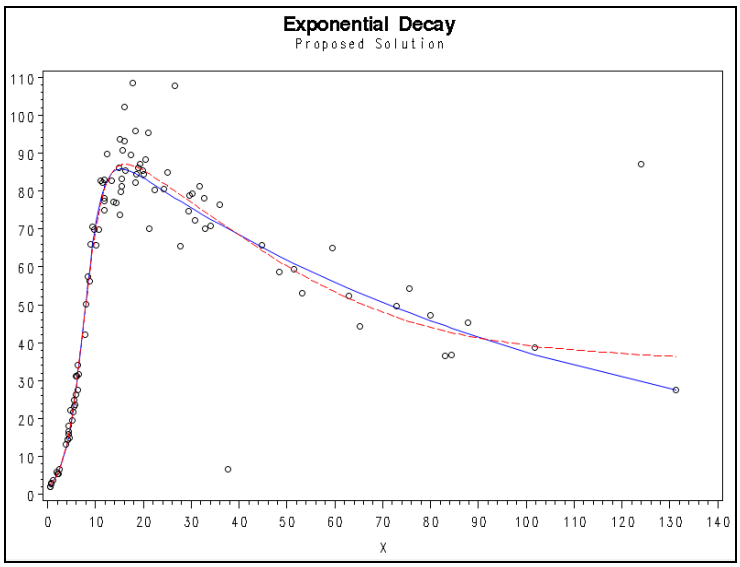
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

19. Change the Target Layer Error Function property to **Huber**, and click **OK** to save the change.



20. Right-click the **SAS Code (2)** node, and select **Run** from the pop-up menu. Confirm the run.

21. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs,** and maximize the plot



Although there is still slight leverage exerted by the rightmost outlier, applying Huber's M-estimator has greatly improved the fit achieved by the (default) normal deviance error function.

22. Close the Results window and the SAS Code (2) node.

23. Close the "robust networks" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.02 Poll

Minimizing deviance is equivalent to maximizing likelihood when the target is a member of the exponential family.

- ❍ True
- ❍ False

23

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Exercises

2. Modeling Count Data

The **CRAB** data set is used in this exercise. It contains data from a study that was conducted on the mating habits of female horseshoe crabs. The study investigated factors affecting whether the female horseshoe crab had any other males, called satellites, residing nearby. The target variable is for each female crab is her number of satellites. The variables in the **CRAB** data set are listed below.

Variable	Description	Role	Level
Color	Female horseshoe crab's color (light medium, medium, dark medium, dark)	Input	Nominal
Satellites	The number of satellites (male horseshoe crabs) residing nearby	Target	Interval
Spine	Spine condition (both good, one worn, both worn)	Input	Nominal
Weight	Weight in kilograms	Input	Interval
Width	Carapace width in centimeters	Input	Interval

a. Right-click **Diagrams** in the project panel, and select **Create Diagram**.

b. Enter **exercise 3.2** in the **Diagram Name** field, and click **OK**.

c. Right-click **Data Sources** in the project panel (upper left corner) and select **Create Data Source**.

d. Click **Next** to progress to the second step of the metadata definition process.

e. Click **Browse** to choose a SAS table from the available libraries.

f. Open the **dmnn41** library and select the **CRAB** data table. Click **OK** to continue.

g. Click **Next** to display information about the **CRAB** table, such as the number of variables (5) and the number of observations (173) it contains.

h. Again, click **Next** to begin modifying the metadata.

i. Select the **Advanced** setting and click **Next** to proceed.

j. The variables **Color** and **Spine** are correctly set to **Nominal**. However, **Satellites** was incorrectly set to **Nominal**. Change it back to **Interval**. Also change the role of **Satellites** to **Target**.

Name	Role	Level	Report	Order	Drop	Lower Limit
Color	Input	Nominal	No		No	.
Satellites	Target	Interval	No		No	.
Spine	Input	Nominal	No		No	.
Weight	Input	Interval	No		No	.
Width	Input	Interval	No		No	.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

k. Click **Next** three times and then click **Finish** to complete the metadata definition process.

l. Drag-and-drop the **CRAB** data source onto the diagram.

m. Add a **Neural Network** node (Model tab) to the diagram and connect it to the **CRAB** data.

n. Use a linear perceptron (generalized linear model) to model the **CRAB** data.

Remember that the **CRAB** target is count data, which is Poisson distributed, so you will need to change your activation and error functions accordingly.

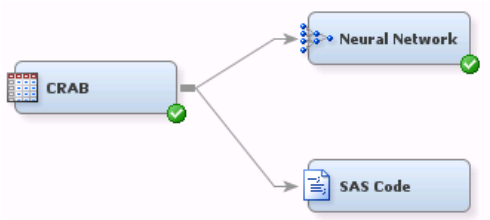
o. Run the diagram from the Neural Network node by right-clicking the **Neural Network** node and selecting **Run** from the pop-up menu. Examine the results when training is complete.

p. Maximize the Output window (bottom right).

q. Scroll to line **132** (Optimization Results), and complete the following table:

Parameter	Estimate
BIAS_Satellites	
Weight_Satellites	
Width_Satellites	
Color1_Satellites	
Color2_Satellites	
Color3_Satellites	
Spline1_Satellites	
Spline2_Satellites	

r. Add a **SAS Code** node to the flow and attach it to the **CRAB** data source. Your flow should now look similar to the following diagram:



s. Right-click in the (default) Training Code editor window, and select **Open** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

t. Read in the program **exercise 3.2** (in d:\workshop\winsas\dmnn41). The program is shown below.

```
title1 'Poisson Model';
proc genmod data=&EM_IMPORT_DATA;
   class color spine / param=effect;
   model satellites=width weight color spine/link=log dist=poisson;
run;
```

The CLASS statement "dummy-codes" the two discrete inputs: **Color** and **Spine**. Effect coding, rather than reference coding, must be specified to match the Neural Network node's results. The DIST option in the MODEL statement specifies the target distribution to be modeled. Because a Poisson distribution is specified, a log-link (exponential activation) function must be applied to keep the argument to the Poisson distribution positive.

u. Run the program from within the SAS Code node by clicking the running man icon at the top of the window. Confirm your run and save your changes when asked.

v. When PROC GENMOD has completed its model, click **Results** to continue.

w. Go to line **71** and complete the table below.

Parameter	Estimate
Intercept	
Weight	
Width	
Color1	
Color2	
Color3	
Spline1	
Spline2	

x. Compare these results to the values generated by the linear perceptron. Did the Neural Network node and the PROC GENMOD results match (within rounding error)?

y. Close the SAS Code node's Results window.

z. Close the "exercise 3.2" diagram.

End of Exercises

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.3 Numerical Optimization Methods

Objectives

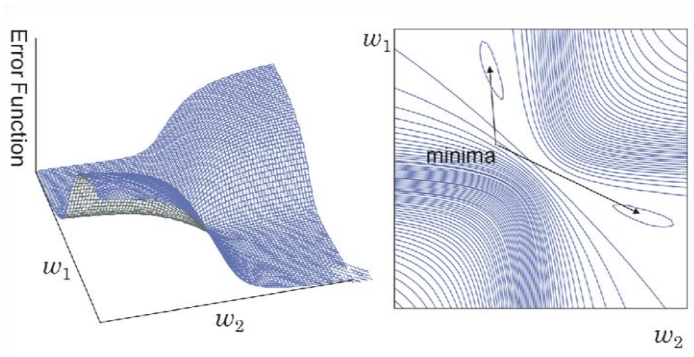
- Outline the optimization (training) techniques that are available in the Neural Network node.

28

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Iterative Updating

$$\mathrm{w}^{(t+1)} = \mathrm{w}^{(t)} + \delta^{(t)}$$



29

The error function defines a surface in the parameter space. If it is a linear model fit by least squares, the error surface is a parabola. In a nonlinear model, however, this error surface is often a complex landscape consisting of numerous deep valleys and steep cliffs.

To efficiently search this landscape for an error minimum, optimization must be used. The optimization methods use local features of the error surface to guide their descent. Specifically, the weights associated with a given error minimum are located using the following procedure:

1. Initialize the weight vector to small random values, $\mathbf{w}^{(0)}$.
2. Use an optimization method to determine the update vector, $\boldsymbol{\delta}^{(t)}$.
3. Add the update vector to the weight values from the previous iteration to generate new estimates:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \boldsymbol{\delta}^{(t)}$$

4. If none of the specified convergence criteria have been achieved, then go to step 2.

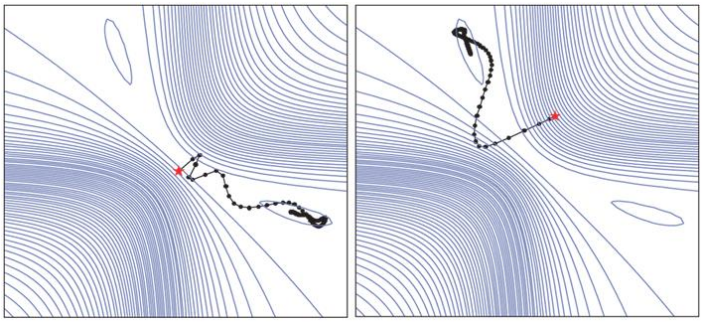
In this chapter, you examine various ways of determining the update vector $\boldsymbol{\delta}^{(t)}$ used in step 2 above, beginning with the most famous update (optimization) algorithm of all, *back propagation*.

To see the available optimization functions, click the ellipsis (**...**) next to **Optimization** and expand the **Training Technique** field.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Back Propagation

$$\delta^{(t)} = -\eta \nabla g^{(t)} + \alpha \delta^{(t-1)}$$



87 iterations
(η = 0.5, α = 0.9)

285 iterations
(η = 0.1, α = 0.9)

30

Re-invented several times, the back propagation (*backprop*) algorithm initially used only *gradient descent* to determine an appropriate set of weights. The gradient, $\nabla \mathbf{g}^{(t)}$, is the vector of partial derivatives of the error function with respect to the weights. It points in the steepest direction uphill. By negating the step size (i.e., *learning rate*) parameter, η, a step is made in the direction that is locally steepest downhill.

$$\boldsymbol{\delta}^{(t)} = -\eta \nabla \mathbf{g}^{(t)}$$

Unfortunately as gradient descent approaches the desired weights, it exhibits numerous back-and-forth movements known as *hemstitching*. To control the training iterations wasted in this hemstitching, later versions of back propagation included a momentum term, yielding the modern update rule:

$$\boldsymbol{\delta}^{(t)} = -\eta \nabla \mathbf{g}^{(t)} + \alpha \boldsymbol{\delta}^{(t-1)}$$

The momentum term retains the last update vector, $\boldsymbol{\delta}^{(t-1)}$, using this information to "dampen" potentially oscillating search paths. The cost is an extra learning rate parameter ($0 \le \alpha \le 1$) that must be set.

The default value of α is 0. This means that, by default, backprop performs gradient descent.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Determining the Optimal Learning Rate (Self-Study)

The gradient descent method requires that you specify a learning rate value. This raises the question of what value should be assigned. Unfortunately, the size of the optimal learning rate is problem specific.

Although the answer to the question of the size of the optimal learning rate is still largely a matter of experimentation, a few guidelines have emerged. For example, Le Cun et al. (1993) showed that the largest value of the learning rate parameter is given by the following:

$$\eta_{\max} = \frac{2}{\lambda_{\max}}$$

where $\lambda_{\max}$ is the largest eigenvalue of the Hessian matrix.

Principe et al. (2000) came to much the same conclusion when they suggested that, if the inputs are centered, the maximum learning rate is a function of the autocorrelation matrix, **R** (defined below).

$$R_{kj} = \frac{1}{n}\sum_i x_{ik} x_{ij}$$

Here x_{ij} is the jth input variable in the ith observation of n. Specifically, they found that the learning rate is related to the size of the maximum eigenvalue ($\lambda_{\max}$) of **R**.

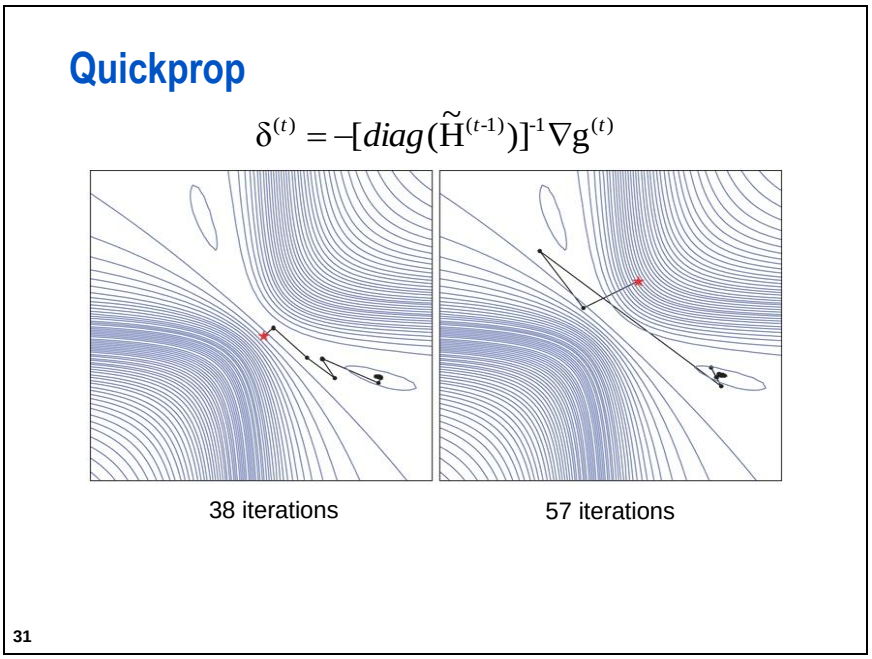
$$\eta_{\max} = \frac{2}{\lambda_{\max}}$$

The eigenvalues of **R** define the curvature of the error surface, and the largest eigenvalue corresponds to the steepest curvature direction of the error surface. In the early stages of adaptation, convergence is primarily along the direction of the largest eigenvalue, corresponding to the steepest curvature direction of the error surface. However, toward the end of adaptation, the algorithm will mainly adapt only the weight with the smallest eigenvalue. They suggest, therefore, that it is usually a good idea to include a safety factor. For example, if a safety factor of 10 is used, the maximum rate is given by the following:

$$\eta_{\max} = 0.1\left(\frac{2}{\lambda_{\max}}\right)$$

Because large learning rate values risk divergence whereas a learning rate that is too small merely risks increased training time, a useful rule of thumb is to err on the side of smaller values.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



QPROP (Quick Back Propagation)

The QPROP algorithm (Fahlman 1988) offers the advantages that it is much faster and more reliable than standard batch back propagation. It also requires no tuning. In order to understand how QPROP works, consider the following. If **w*** is the set of weights yielding the minimum error value, and **H** is the Hessian (second derivative) matrix, the gradient at any point **w** is given by the following:

$$\nabla \mathbf{g} = \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

Re-arranging the terms in this equation yields this formula:

$$\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1}\nabla \mathbf{g}$$

In other words, if you subtract $\mathbf{H}^{-1}\nabla \mathbf{g}$ (known the *Newton step*) from the current weight values, ***in a single iteration you get the weights that minimize the specified error value***.

Unfortunately, calculating the Hessian matrix for the Newton step is a computationally intensive task, so QPROP approximates the error surface by a parabola, and then uses two successive evaluations of the error function and its gradient to determine the coefficients of the parabola (Bishop 1995). This is known as *finite differencing*. The diagonal elements in the approximated Hessian matrix are then extracted, yielding the QPROP update rule:

$$\boldsymbol{\delta}^{(t)} = -\left(diag(\tilde{\mathbf{H}}^{(t-1)})\right)^{-1}\nabla \mathbf{g}^{(t)} = \left(\frac{\frac{\partial}{\partial \mathbf{w}_j}\mathbf{Q}^{(t)}}{\frac{\partial}{\partial \mathbf{w}_j}\mathbf{Q}^{(t-1)} - \frac{\partial}{\partial \mathbf{w}_j}\mathbf{Q}^{(t)}}\right)\boldsymbol{\delta}_j^{(t-1)}$$

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

RPROP (Resilient Back Propagation)

An alternative to QPROP is RPROP (Martin Riedmiller and Heinrich Braun 1993). RPROP does not use the gradient's magnitude in its estimate. Instead, the value of each weight and bias is increased by a factor of δ_{ji} whenever the derivative of the error function with respect to the weights has the same sign (same direction) on two successive iterations. Conversely, the value is decreased whenever the derivative changes sign from iteration to iteration. If the derivative is zero, then no change is made. This yields the following set of update equations:

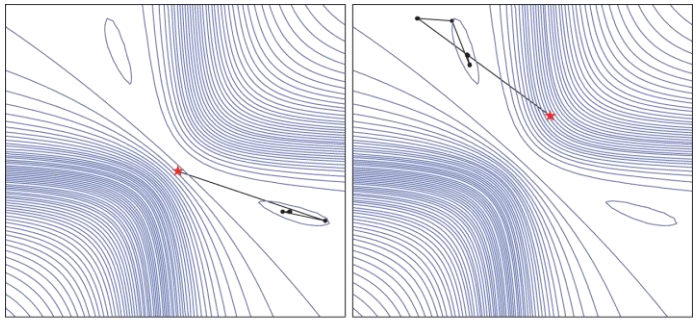
$$\delta_{ji}^{(t)} = \begin{cases} \eta\delta_{ji}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ji}}(t-1)\frac{\partial E}{\partial w_{ji}}(t) > 0 \\ \mu\delta_{ji}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ji}}(t-1)\frac{\partial E}{\partial w_{ji}}(t) < 0 \\ \delta_{ji}^{(t-1)} & \text{otherwise} \end{cases} \qquad \Delta w_{ji}^{(t)} = \begin{cases} -\delta_{ji}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ji}}(t) > 0 \\ +\delta_{ji}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ji}}(t) < 0 \\ 0 & \text{otherwise} \end{cases}$$

RPROP is one of the most stable of all the "prop" techniques. It achieves training speeds many times faster than the original back propagation algorithm.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Levenberg-Marquardt

$$\delta^{(t)} = -(\mathrm{J}^{(t)\prime}\mathrm{J}^{(t)} + \lambda^{(t)}\mathrm{I})^{-1}\mathrm{J}^{(t)\prime}\mathrm{r}^{(t)}$$



32

If the error function uses deviance (maximum likelihood), the task of minimizing the deviance can be cast as a least squares problem using residuals (McCullagh and Nelder 1989).

$$Q(\mathbf{w}) = \| \mathbf{r} \|^2 = \sum_{i=1}^{d} r^2$$

Only a Jacobian (first derivative) matrix, **J**, is needed to calculate the update vector. More specifically, if $\mathbf{J} = \partial\mathbf{r} / \partial\mathbf{w}'$ is a Jacobian matrix of partial residuals with respect to the weights, then the *Levenberg-Marquardt* update rule is given by the following:

$$\boldsymbol{\delta}^{(t)} = -\left(\mathbf{J}^{(t)\prime}\mathbf{J}^{(t)} + \lambda^{(t)}\mathbf{I}\right)^{-1}\mathbf{J}^{(t)\prime}\mathbf{r}^{(t)}$$

The Hessian in the Newton step is approximated by the formula:

$$\mathbf{J}^{(t)\prime}\mathbf{J}^{(t)} + \lambda^{(t)}\mathbf{I}$$

Notice that the matrix $\mathbf{J}^{(t)\prime}\mathbf{J}^{(t)}$ is augmented by adding a multiple of the identity matrix (**I**). This keeps the diagonal positive, creating a hyper-elliptical neighborhood where the quadratic is adequate. In a similar fashion, the Newton step gradient is approximated by the formula:

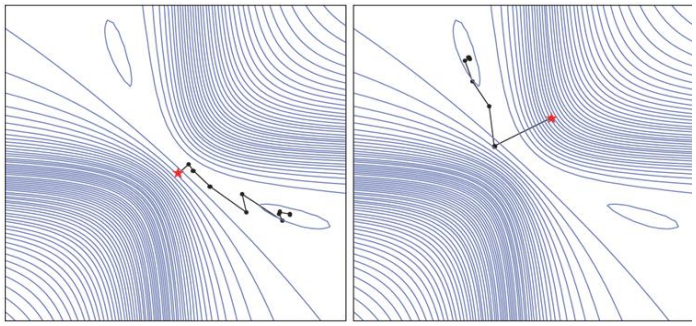
$$\mathbf{J}^{(t)\prime}\mathbf{r}^{(t)}$$

The Levenberg-Marquardt method is the default for problems with less than 100 weights.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Quasi-Newton

$$\delta^{(t)} = -\eta^{(t)}[\mathrm{B}^{(t-1)} + \mathrm{E}^{(t-1)}]^{-1}\nabla g^{(t)}$$



11 iterations 8 iterations

33

Modified Newton methods, such as quasi-Newton, are among the fastest and most reliable algorithms for unconstrained optimization of smooth objective functions. Fletcher (1987) notes the following:

> "When compared with the results for the quasi-Newton methods … it is clear that conjugate gradient methods are both less efficient and less robust, and therefore should not be preferred in normal circumstances." (p. 85)

For efficiency reasons the quasi-Newton update rule (below) approximates the Hessian matrix in the Newton step using the first derivative matrices **B** and **E**.

$$\boldsymbol{\delta}^{(t)} = -\eta^{(t)}\left(\mathbf{B}^{(t-1)} + \mathbf{E}^{(t-1)}\right)^{-1}\nabla\mathbf{g}^{(t)}$$

Matrix **B** is usually initialized to the identity matrix. Matrix **E** can be determined a number of ways. One of the most popular is the BFGS method, suggested independently by Broyden, Fletcher, Goldfarb, and Shanno in 1970. A relatively imprecise line search is used to determine the step size, $\eta^{(t)}$.

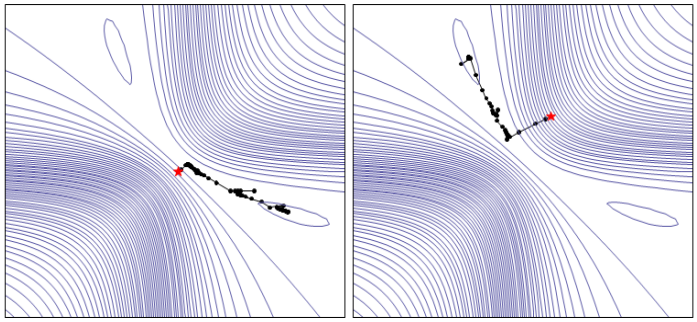
The HPNEURAL procedure uses a variant of the BFGS method known as *limited memory BFGS*. Like the original BFGS, the limited memory BFGS (L-BFGS) uses an estimation of the inverse Hessian to steer the search. But, whereas BFGS stores an *n* by *n* approximation to the Hessian (where *n* is the number of variables), the L-BFGS variant stores only a few vectors that represent the approximation implicitly. Thus, L-BFGS is well suited for optimization problems with a large number of variables. (See "Limited-memory BFGS" in Wikipedia for more details.)

The quasi-Newton method is the default method for problems with between 100 and 500 weights.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Conjugate Gradient

$$\delta^{(t)} = -\eta^{(t)}[-\nabla g^{(t)} + \beta^{(t-1)}\delta^{(t-1)}]$$



66 iterations 45 iterations

34

As was noted previously, when a simple gradient is used to guide the search process, successive steps are necessarily perpendicular to each other (Hertz, Krough, and Palmer 1991), which means that the approach to the error minimum follows a zigzag path, often greatly slowing convergence. The *conjugate gradient* optimization method minimizes this hemstitching by choosing each new search direction so as to "spoil", as little as possible, the direction set in the previous steps.

The first update vector is generated using a *steepest* (not gradient) *descent* step.

$$\boldsymbol{\delta}^{(t)} = -\eta^{(t)}\nabla\mathbf{g}^{(t)}$$

Subsequent vectors are determined so that the updates are mutually conjugate with respect to the Hessian matrix of the objective function.

$$\boldsymbol{\delta}^{(t)} = -\eta^{(t)}(-\nabla\mathbf{g}^{(t)} + \boldsymbol{\beta}^{(t-1)}\boldsymbol{\delta}^{(t-1)})$$

Remarkably, this is accomplished without actually having to calculate the Hessian. For example, the Polak-Ribière rule calculates the $\boldsymbol{\beta}$ value using only gradient information:

$$\boldsymbol{\beta} = \frac{\left(\nabla\mathbf{g}^{new} - \nabla\mathbf{g}^{old}\right).\nabla\mathbf{g}^{new}}{\left(\nabla\mathbf{g}^{old}\right)^2}$$

In this equation, $\nabla\mathbf{g}^{new}$ is the new gradient direction and $\nabla\mathbf{g}^{old}$ is the old gradient direction.

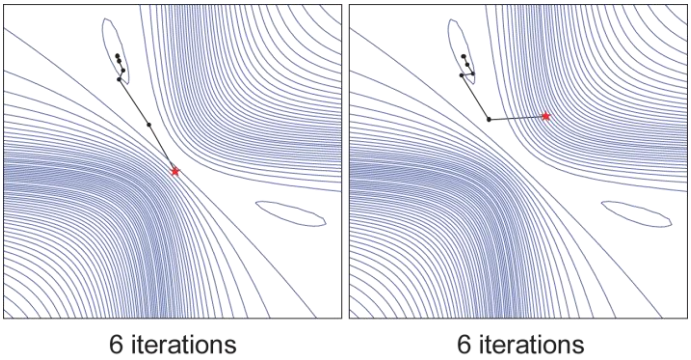
On a strictly quadratic surface in *n* dimensions, a conjugate gradient search reaches an error minimum in *n* steps. Moreover, conjugate gradient requires very little storage, which led Fletcher (1987) to conclude that conjugate gradient methods might be the only methods applicable to large problems.

Conjugate gradient is the default update method for problems with more than 500 weights.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Trust Region

$$\delta^{(t)} = -(\mathrm{H}^{(t)} + \lambda^{(t)}\mathrm{I})^{-1}\nabla \mathrm{g}^{(t)}$$



35

If the error function is highly non-quadratic, as happens often in neural networks, the Newton step might be either too long or too short. The modified Newton *Trust Region* method restricts its steps to a hyper-elliptical neighborhood, known as a "trust region" where the quadratic approximation is adequate. As in Levenberg-Marquardt, this is done by adding $\lambda^{(t)}$ to the diagonal elements of the Hessian, **H**, matrix.

$$\delta^{(t)} = -(\mathbf{H}^{(t)} + \lambda^{(t)}\mathbf{I})^{-1}\nabla \mathbf{g}^{(t)}$$

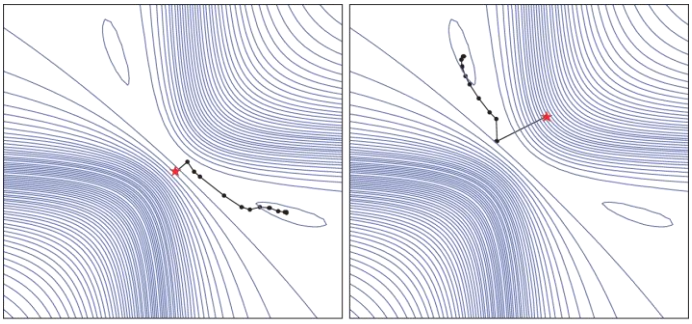
The constant $\lambda^{(t)}$ controls the size of the trust region. It alters the length and direction of the step. The quality of the quadratic approximation is tested each iteration, and $\lambda^{(t)}$ is changed accordingly.

The Trust Region method is applicable to problems with up to 40 weights.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Double-Dogleg

$$\delta^{(t)} = -\alpha_1 s^{(t)}_{Steepest\,Descent} + \alpha_2 s^{(t)}_{Quasi\,Newton}$$



12 iterations 11 iterations

36

The *double dogleg* optimization method tries to “hedge its bets” by combining two other optimization methods when estimating its search direction. The double dogleg rule computes a linear combination of the steepest descent direction and the quasi-Newton direction.

Because the step size is restricted to fall within the trust region, no line search is needed.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Training Tracks

c3s3d1

The optimization algorithms leave distinctive tracks across the parameter space. An interesting low-dimensional perspective can be seen by plotting the parameter values against the iteration number. Because of its relative simplicity, the **FITNESS** data (described in Chapter 2) is again used.

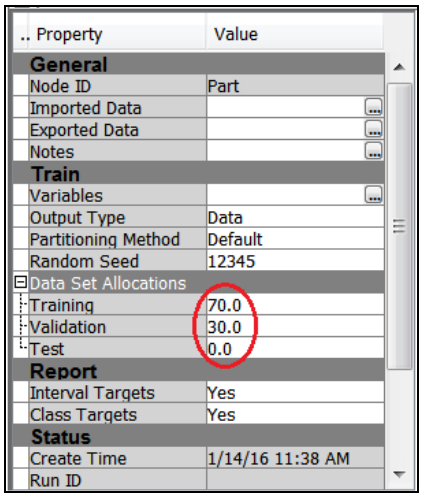
1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **training tracks** in the **Diagram Name** field, and click **OK**.
3. Drag and drop the **FITNESS** data source onto the diagram.
4. Click the ellipsis (**…**) next to **Variables** and change the role of **Age**, **Maximum_Pulse**, **RunTime**, **Run_Pulse**, and **Weight** from **Rejected** to **Input**. Also change **Performance** to **Rejected**.

Name	Role	Level	Report	Order	Drop	Lower Limit	Upper Limit
Age	Input	Interval	No		No	.	.
Gender	Rejected	Nominal	No		No	.	.
Maximum_Pulse	Input	Interval	No		No	.	.
Name	Rejected	Nominal	No		No	.	.
Oxygen_Consumption	Target	Interval	No		No	.	.
Performance	Rejected	Interval	No		No	.	.
Rest_Pulse	Rejected	Interval	No		No	.	.
RunTime	Input	Interval	No		No	.	.
Run_Pulse	Input	Interval	No		No	.	.
Weight	Input	Interval	No		No	.	.

The inputs specified in this demonstration are the variables recommended by the Hocking criterion when linear regression is used to model the relationship.

5. To use early stopping, add a **Data Partition** node to the flow. Connect it to the **FITNESS** data source.

6. Set the data set allocations to **Train 70.0**, **Validate 30.0**, and **Test 0.0**.

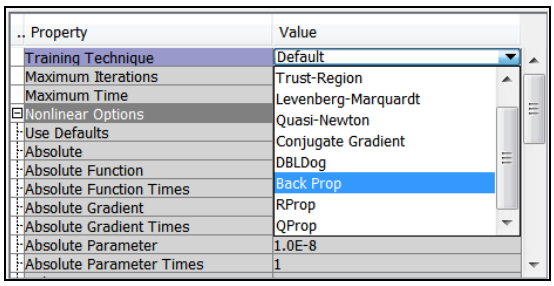


Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

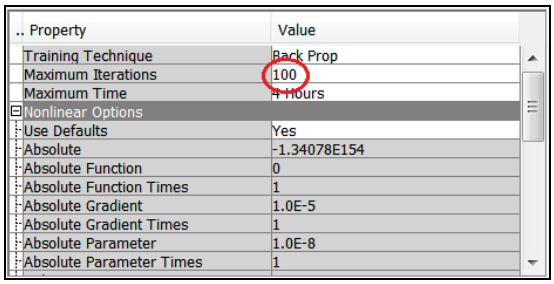
7. Add a **Neural Network** node, and connect it to the **Data Partition** node.

8. In the Neural Network node's properties panel, click the ellipsis (**…**) next to **Optimization**.

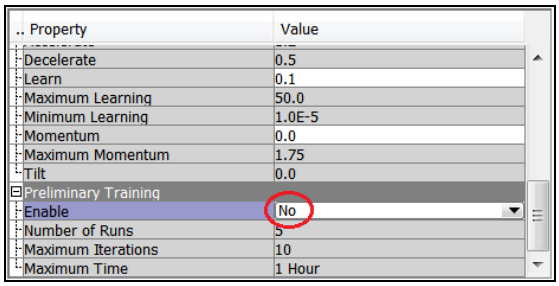
9. Change the Training Technique property to **Back Prop**, the original learning technique.



10. Also increase the Maximum Iterations property to **100**.



11. Scroll down to the Preliminary Training section and set the Enable option to **No**.



12. Click **OK** to save your changes.

13. Drag and drop a **SAS Code** node onto the diagram. Connect it to the **Neural Network** node. Your final diagram should look like the one shown below.

14. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.

15. In the (default) Training Code editor, right-click in the white space and select **Open**.

16. Select the program **c3s3d1.sas** (in d:\workshop\winsas\dmnn41). The program, which plots the changing weight values as the network learns, is listed below.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

First, the global system and graphics options are initialized. Then a symbol and two axes are defined.

```
options nodate nonumber;
goptions reset=all device=actximg;

symbol1 c=b v=none i=join mode=include;

axis1 label=(c=bl a=-90 r=90 'Parameter Values');
axis2 label=(c=bl 'Iteration Number') order=(0 to 100 by 25);
```

Next, the list of connections (weights) to be plotted is retrieved from the file containing the parameter estimates data generated by the Neural Network node (**%EM_IMPORT_ESTIMATE**).

```
proc contents data=&EM_IMPORT_ESTIMATE noprint out=temp(keep=name);
run;

proc sql noprint;
   select name
      into :input_hidden separated by ' '
      from temp
      where substr(name,1,1) ne '_'
      and substr(name,1,2) ne 'H1'
      and substr(name,1,4) ne 'BIAS';
quit;

proc sql noprint;
   select name
      into :hidden_output separated by ' '
      from temp
      where substr(name,1,2) = 'H1';
quit;
```

The CONTENTS procedure pulls the names of the connections from **%EM_IMPORT_ESTIMATE** and writes this information to **TEMP**. The first call to PROC SQL retrieves the names of the input-to-hidden unit connections and stores them in the macro variable **input_hidden**. The second call to SQL retrieves the hidden-to-output connection names, and stores them in **hidden_output**.

One more macro variable is created. (See below.) This macro variable (**error**) stores the final error value generated by the neural network optimization function that is being used.

```
data _null_;
   set &EM_IMPORT_ESTIMATE(where=(_NAME_='_LAST_'));
   call symputx('error',left(round(_AVERR_,.0001)));
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The GPLOT procedure is then called to plots the training tracks. The stored error is also reported.

```
title1 h=1.5 f=swissb 'Input to Hidden Layer Weights';
proc gplot data=&EM_IMPORT_ESTIMATE(where=(_TYPE_='PARMS'));
   plot (&input_hidden)*_iter_=1/overlay vaxis=axis1 haxis=axis2;
   footnote j=right h=1.5 "Error: &error";
run;

title1 h=1.5 f=swissb 'Hidden to Output Layer Weights';
proc gplot data=&EM_IMPORT_ESTIMATE(where=(_TYPE_='PARMS'));
   plot (&hidden_output)*_iter_=1/overlay vaxis=axis1 haxis=axis2;
   footnote j=right h=1.5 "Error: &error";
run;
quit;
```

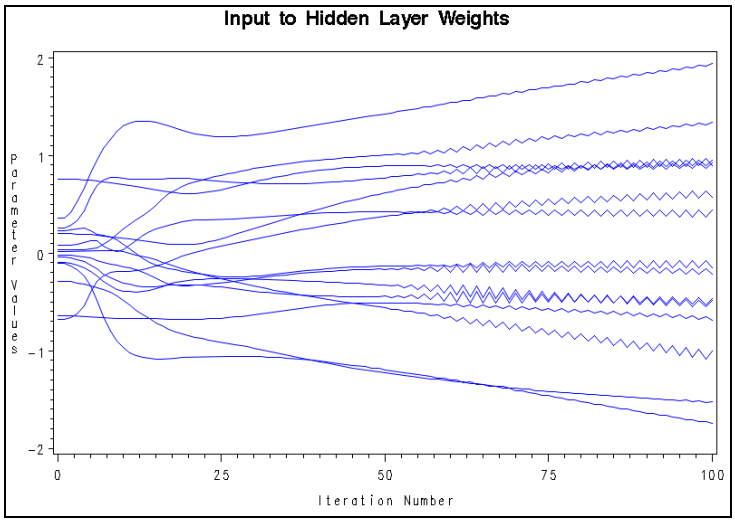
The first call to the GPLOT procedure displays the input layer to hidden layer weights. The second call to PROC GPLOT displays the hidden layer to output layer weights.

17. Click the running man icon at the top of the page to run the program. Confirm the run.

18. Examine the results when they become available.

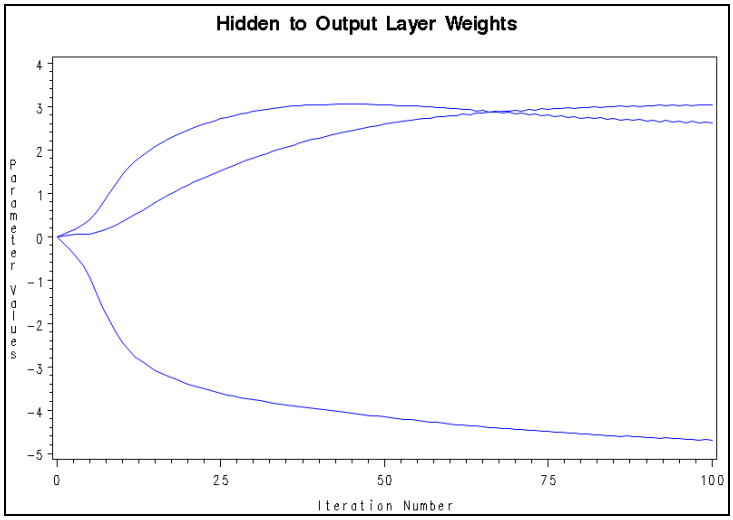
19. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** and maximize the input-to-hidden layer plot.

    The first plot, which displays the magnitude changes across iterations for the input-to-hidden weights, shows that there is substantial hemstitching occurring.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

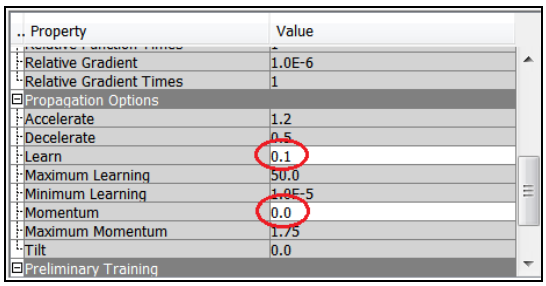
20. Click the right arrow at the bottom of the page to see the plot of the hidden-to-output weights. Hemstitching is also present in the hidden-to-output connections. (See below.)



The final error value, after the 100 iterations has been completed, is 1.2652.

21. Again click the ellipsis (**…**) next to **Optimization**.

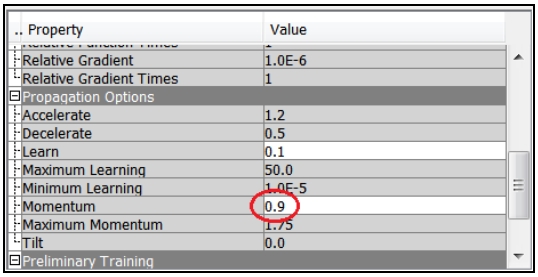
22. Scroll down to the bottom of the menu and examine the default learning rate (Learn) and momentum (Momentum) values in the Propagation Options section of the properties panel (shown below).



.. Property	Value
Relative Gradient	1.0E-6
Relative Gradient Times	1
Propagation Options	
Accelerate	1.2
Decelerate	0.5
Learn	0.1
Maximum Learning	50.0
Minimum Learning	1.0E-5
Momentum	0.0
Maximum Momentum	1.75
Tilt	0.0
Preliminary Training	

Notice that the default learning rate is 0.1 and the default momentum value is 0.0. In other words, ***the default settings perform simple gradient descent*** (which is known to be prone to hemstitching).

23. Change the Momentum learning rate value to **0.9**.



.. Property	Value
Relative Gradient	1.0E-6
Relative Gradient Times	1
Propagation Options	
Accelerate	1.2
Decelerate	0.5
Learn	0.1
Maximum Learning	50.0
Minimum Learning	1.0E-5
Momentum	0.9
Maximum Momentum	1.75
Tilt	0.0
Preliminary Training	

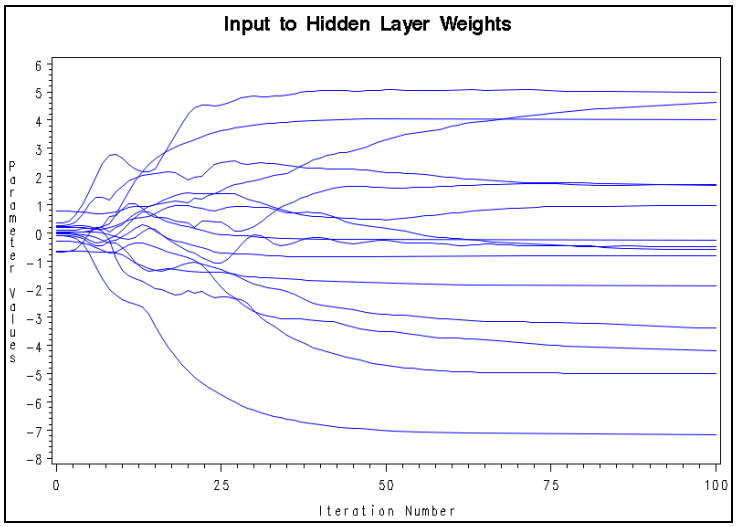
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

24. Click **OK**.

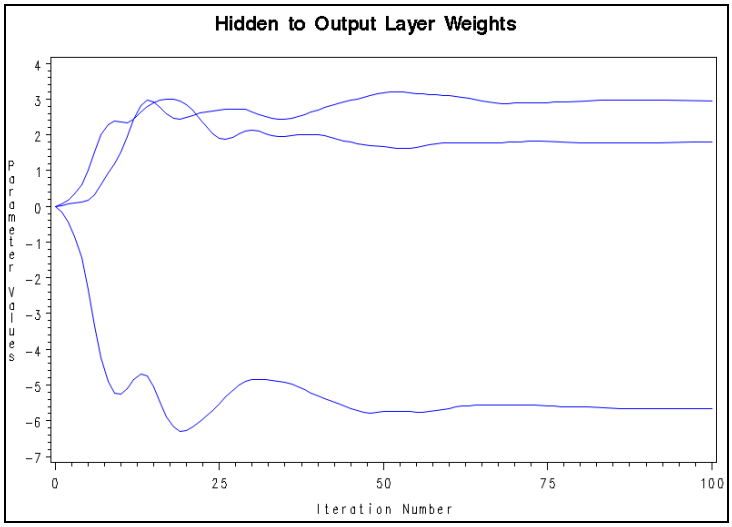
25. Right-click the **SAS Code** node and select **Run** from the pop-up menu.

26. Examine the results when they become available.

27. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** and maximize the input-to-hidden layer plot. Notice that adding a momentum term has eliminated the hemstitching previously observed



28. Click the right arrow at the bottom of the page to see the plot of the hidden-to-output weights.



The second plot shows that the hidden-to-output hemstitching has also completely disappeared. Notice, too, that adding a momentum term reduced the final error value from 1.2652 to 0.8009.

29. Close the SAS Code node Training Graphs window.

30. Close the “training tracks” diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.03 Poll

Numerical optimization methods guarantee that the neural network will eventually find the optimal solution.

- ❍ True
- ❍ False

38

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.4 Chapter Summary

Estimating the unknown parameters in a neural network is referred to as *learning*. Each of the points in the *n*-dimensional parameter space is a possible solution. The number of parameters grows exponentially with the number of dimensions. This made early neural network learning algorithms very slow to train. Now numerical optimization algorithms efficiently search the parameter space.

The SAS Enterprise Miner Neural Network node offers a rich set of optimization methods. Conjugate gradient is the default method when there are more than 500 weights. Quasi-Newton optimization is the default for networks with between 100 and 500 weights. And Levenberg-Marquardt is the default optimization method for networks with less than 100 weights.

Before an optimization method can be used, however, a criterion for choosing among the different sets of parameter estimates needs to be specified. This criterion is known as an *error function*. Again the Neural Network node offers wide range of options. Applying different error functions typically yields different parameter estimates. The choice of which error function to apply depends on statistical considerations.

Error functions define a surface to be searched. These error surfaces are often plagued with local minima. In order to prevent the optimization algorithm from falling into bad local minima, the Neural Network node initializes its hidden-to-output weights to zero. Only the output biases are assigned a value; they are set to the (possibly link transformed) mean of their respective target. This means that on the first iteration, the estimated output will be each targets' (link-transformed) mean. To further assist in avoiding the worst local minima, by default the Neural Network node also implements "preliminary training," a process that seeks out the best starting location from among a number of candidate starting locations.

Finally, because that large magnitude weights are known to be largely responsible for overfitting, the Neural Network node offers two ways to help keep the weights from growing too quickly. The first method, known as weight decay penalizes the weights in proportion to their magnitude. Alternatively, early stopping can be used. Early stopping keeps the weights small by cutting them back to an earlier point, determined by the neural network's performance on validation data, in the training history.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.5 Solutions

Solutions to Exercises

1. Network Initialization

The listing of the initial parameter values is reproduced below.

```
                    Optimization Start
                    Parameter Estimates
                                                  Gradient
                                                 Objective
  N Parameter                      Estimate       Function

  1 Performance_H11               -0.042981              0
  2 Performance_H12               -0.099988              0
  3 Performance_H13               -0.243493              0
  4 BIAS_H11                      -0.222261              0
  5 BIAS_H12                       0.073533              0
  6 BIAS_H13                       0.499365              0
  7 H11_Oxygen_Consumption                0       0.327439
  8 H12_Oxygen_Consumption                0       0.792486
  9 H13_Oxygen_Consumption                0       1.542352
 10 BIAS_Oxygen_Consumption       47.375806     1.23772E-14

Value of Objective Function = 27.469501769
```

1) The weights connecting the input-to-hidden units are as follows:

   - **Performance_H11**
   - **Performance_H12**
   - **Performance_H13**

   And the hidden unit biases are as follows:

   - **BIAS_H11**
   - **BIAS_H12**
   - **BIAS_H13**

   Both the input-to-hidden weights and the hidden unit biases are set to small random values drawn, by default, from a normal distribution with a mean of zero and a standard deviation of 0.1.

2) The weight values that are initialized to zero are the following:

   - **H11_Oxygen_Consumption**
   - **H12_Oxygen_Consumption**
   - **H13_Oxygen_Consumption**

   These are the hidden-to-output connections. The reason that these values are set to zero is so that the initial estimate will be the value contained in output unit bias

3) The value of the output unit bias (**BIAS_Oxygen_Consumption**) is 47.375806. This is the mean of target variable (**Oxygen_Consumption**)

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

2. Modeling Count Data

Because the target is Poisson distributed, you need to change the Target Layer Error Function property to **Poisson**. You also need to change the Target Layer Activation Function property to **Exponential** (log-link).

When the linear perceptron is run with these settings, the following estimates are generated by the generalized linear model in the Neural Network node (PROC NEURAL).

```
                  Optimization Results
                  Parameter Estimates
                                              Gradient
                                             Objective
  N Parameter                Estimate         Function

  1 Weight_Satellites        0.496469      0.000001950
  2 Width_Satellites         0.016749      0.000040214
  3 Color1_Satellites        0.327353     -0.000001211
  4 Color2_Satellites        0.062503     -0.000000466
  5 Color3_Satellites       -0.186351     -4.569501E-8
  6 Spine1_Satellites        0.021030     -0.000001784
  7 Spine2_Satellites       -0.129341     -0.000000936
  8 BIAS_Satellites         -0.710181      0.000001841

Value of Objective Function = 3.1767953048
```

Therefore, the completed table is as follows:

Parameter	Estimate
BIAS_Satellites	-0.710181
Weight_Satellites	0.496469
Width_Satellites	0.016749
Color1_Satellites	0.327353
Color2_Satellites	0.062503
Color3_Satellites	-0.186351
Spline1_Satellites	0.021030
Spline2_Satellites	-0.129341

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The GENMOD procedure results at line 71 are shown below.

Analysis Of Maximum Likelihood Parameter Estimates

Parameter		DF	Estimate	Standard Error	Wald 95% Confidence Limits		Wald Chi-Square	Pr > ChiSq
Intercept		1	-0.7102	0.9339	-2.5407	1.1203	0.58	0.4470
Width		1	0.0167	0.0489	-0.0791	0.1126	0.12	0.7321
Weight		1	0.4965	0.1663	0.1706	0.8223	8.92	0.0028
Color	1	1	0.3274	0.1322	0.0683	0.5864	6.13	0.0133
Color	2	1	0.0625	0.0737	-0.0820	0.2070	0.72	0.3966
Color	3	1	-0.1864	0.0964	-0.3752	0.0025	3.74	0.0532
Spine	1	1	0.0210	0.0947	-0.1647	0.2067	0.05	0.8243
Spine	2	1	-0.1293	0.1313	-0.3868	0.1281	0.97	0.3248
Scale		0	1.0000	0.0000	1.0000	1.0000		

Thus, the completed PROC GENMOD table is as follows:

Parameter	Estimate
Intercept	-0.7102
Weight	0.4965
Width	0.0167
Color1	0.3274
Color2	0.0625
Color3	-0.1864
Spline1	0.0210
Spline2	-0.1293

A comparison of the values in the two tables reveals that the parameters generated by the Neural Network node's Linear Perceptron, and the Poisson regression model produced by the GENMOD procedure match (to within rounding error).

End of Solutions

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Solutions to Student Activities (Polls and Quizzes)

3.01 Poll – Correct Answer

The purpose of weight decay and early stopping is to keep the network from falling into a bad local minimum.

- ❍ True
- ❍ False

12

The purpose of weight decay and early stopping is to prevent the weights in the network from getting too large, thereby increasing the chance of over fitting. Preliminary training is used to avoid bad minima.

3.02 Poll – Correct Answer

Minimizing deviance is equivalent to maximizing likelihood when the target is a member of the exponential family.

- ❍ True
- ❍ False

24

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3.03 Poll – Correct Answer

Numerical optimization methods guarantee that the neural network will eventually find the optimal solution.

- ❍ True
- ❍ False

39

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Chapter 4 The NEURAL Procedure

4.1 Overview of PROC NEURAL 4-3

Demonstration: Linear Perceptrons Using PROC NEURAL 4-9

4.2 Input Selection Using PROC NEURAL 4-15

Demonstration: Variable Importance 4-18

Demonstration: Sensitivity-Based Pruning 4-26

Exercises 4-32

4.3 Sequential Network Construction 4-33

Demonstration: Sequential Network Construction 4-35

Exercises 4-46

4.4 Generalized Additive Neural Networks (Self-Study) 4-47

Demonstration: Generalized Additive Neural Network 4-54

4.5 Stochastic Gradient Descent 4-71

Demonstration: Mini-Batch Gradient Descent 4-75

4.6 Chapter Summary 4-84

4.7 Solutions 4-85

Solutions to Exercises 4-85

Solutions to Student Activities (Polls and Quizzes) 4-88

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.1 Overview of PROC NEURAL

Objectives

- Construct custom neural networks.
- Outline the basic syntax of PROC NEURAL.

3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

NEURAL Procedure

```
PROC NEURAL DATA=<SAS-data-set> DMDBCAT=<catalog>;
    INPUT <variable(s)> / LEVEL=<level> ID=<name>;
    HIDDEN <integer> / ID=<name>;
    TARGET <targets> / LEVEL=<level> ID=<name>;
    CONNECT <name> … <name>;
    PRELIM <integer> MAXITER=<integer>;
    TRAIN;
RUN;
```

- PROC NEURAL enables you to construct custom feed-forward neural network architectures.

4

The NEURAL procedure underlies the functionality of the Neural Network node. Although network construction is usually easier through the GUI, the NEURAL procedure can be used to create custom network architectures. With it, you can construct virtually any desired feed-forward network.

To enable advanced users to augment the functionality available in the Neural Network node, the complete PROC DMDB and PROC NEURAL subsections of the *SAS® Enterprise Miner™ User's Guide* are provided. (See Appendix A.) ***It is provided with the understanding that the DMDB and NEURAL procedures are not covered by SAS Technical Support at this time***.

Selected NEURAL procedure statements:

ARCHI constructs a network with at most one hidden layer, sets the hidden-unit ACT= and COMB= options, and sets default values for other options. Only one ARCHI statement is allowed.

Example

```
archi mlp hidden=3 direct;
```

CONNECT explicitly defines one or more network connections. If no CONNECT statements are provided, the network is connected as follows. First, all input units are connected to the first hidden layer. Then each hidden layer except the last is connected to the next hidden layer. Finally, the last hidden layer is connected to all output units. This is known as a *fully connected* neural network.

Example

```
connect in h1 out;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

HIDDEN defines a hidden layer. You can specify as many HIDDEN statements as you want, up to the limits imposed by computer memory. The hidden layers can be connected in any feed-forward pattern using CONNECT statements.

Example

```
hidden 3/bias comb=linear act=tanh id=h1;
```

INPUT enables you to group input variables having common levels (i.e., nominal, ordinal, interval) and standardizations. You can specify as many INPUT statements as you want, up to the limits imposed by computer memory. The input layers can be connected to hidden or output layers using CONNECT statements.

Example

```
input %EM_INTERVAL_INPUT/level=int std=std id=in;
```

PRELIM performs preliminary training to reduce the risk of bad local optima. The final weights and biases in a trained network depend on the initial values. The PRELIM statement repeatedly trains a network for a small number of iterations using different initializations. The final weights of the best network are then used to initialize a subsequent TRAIN statement.

Example

```
prelim 5 maxiter=10;
```

TARGET constructs the target layer. All of the dependent variables in a given TARGET statement must be the same level (i.e., nominal, ordinal, or interval).

Example

```
target %EM_TARGET/level=int act=exp error=poisson id=out;
```

TRAIN trains the network. You can specify the optimization method to be applied. Otherwise, the default optimization technique is used: Levenberg-Marquardt (up to 100 weights), quasi-Newton (between 100 and 500 weights), or conjugate gradient (500 or more weights)

Example

```
train tech=bprop learn=0.4 mom=0.9;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

DMDB Procedure

```
PROC DMDB DATA=<SAS-data-set> DMDBCAT=<catalog>;
      CLASS <discrete variables>;
      VAR <continuous variables>;
RUN;
```

- PROC DMDB converts the specified SAS data set into a data mining database and catalogs it.

5

SAS/Enterprise Miner architecture is based on the creation of a data mining database (DMDB) that is a snapshot of the original data. A data mining database optimizes network performance by reducing the number of passes through the data. The catalog name specified in the DMDB procedure is used by the NEURAL procedure when referencing the cataloged SAS data set.

PROC DMDB creates this DMDB from the input data source. Each data set needs to be cataloged only once. It also compiles and computes metadata information about the input data based on variable roles and stores it in a metadata catalog. The DMDB and the associated metadata catalog facilitate subsequent data mining activities. They are both designed for processing and storage efficiencies.

The DMDBCAT= argument is required for the following procedures: ASSOC, DMDB, DMINE, DMSPLIT, SPLIT, and SEQ. It is optional in the NEURAL and DMREG procedures. It is not a valid argument in the RULEGEN and STDIZE procedures.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Defining the Architecture

```
PROC NEURAL DATA=<SAS data set> DMDBCAT=<catalog>;
      INPUT <inputs> / LEVEL=<level>;
      TARGET <targets> / LEVEL=<level>;
      ARCHI <architecture-name> <HIDDEN=integer> <DIRECT>;
      PRELIM <integer> MAXITER=<integer>;
      TRAIN;
RUN;
```

- An ARCHI statement can be used to define a network with, at most, one hidden layer.
- It is the simplest way to define an architecture.

6

The architecture of a neural network can be specified in either of two ways.

The simplest method is via the ARCHI statement, which enables you to construct a network with at most one hidden layer. The *architecture-name* parameter specifies the network architecture that you want to apply. The following architectures are available:

- Generalized Linear Model (GLIM)
- Multilayer Perceptron (MLP)
- Ordinary Radial Basis Function with Equal Widths (ORBFEQ)
- Ordinary Radial Basis Function with Unequal Widths (ORBFUN)
- Normalized Radial Basis Function with Equal Widths and Heights (NRBFEQ)
- Normalized Radial Basis Function with Equal Heights and Unequal Widths (NRBFEH)
- Normalized Radial Basis Function with Unequal Heights and Equal Widths (NRBFEW)
- Normalized Radial Basis Function with Equal Volumes (NRBFEV)
- Normalized Radial Basis Function with Unequal Heights and Unequal Widths (NRBFUN)

The ARCHI statement has only two options:

- HIDDEN=*n* specifies the number, *n*, of hidden units to be used.
- DIRECT requests that a skip-layer be added.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Defining the Architecture

```
PROC NEURAL DATA=<SAS data set> DMDBCAT=<catalog>;
      INPUT <inputs> / LEVEL=<level> ID=<name>;
      HIDDEN <integer> / ID=<name>;
      TARGET <targets> / LEVEL=<level> ID=<name>;
      CONNECT <ID-list>;
      PRELIM <integer> MAXITER=<integer>;
      TRAIN;
RUN;
```

- CONNECT statements enable you define any feed-forward neural network architecture.

7

You can also specify an architecture using CONNECT statements. The ID list contains the identifiers of the layers to be connected. The identifiers in the ID list must have previously been defined using the ID= option in an INPUT, HIDDEN, or TARGET statement.

Although the use of CONNECT statements is more complicated than the ARCHI statement, the statements enable you to construct virtually any feed forward design. For example, the following PROC NEURAL syntax creates a network with two hidden layers.

```
proc neural data=<dataset name> dmdbcat=<DMDB catalog name>;
   input <inputs> / level=<input level> id=in;
   hidden <number of units in first hidden layer> / id=h1;
   hidden <number of units in second hidden layer> / id=h2;
   target <targets> / level=<target level> id=out;
   connect in h1 h2 out;
   train;
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Linear Perceptrons Using PROC NEURAL

c4s1d1

In this demonstration, you learn how to create a linear perceptron (generalized linear model) using PROC NEURAL coding. The basic syntax for a linear perceptron is given below.

```
PROC NEURAL DATA=<data-set-name> DMDBCAT=<dmdb-catalog>;
    INPUT <independent-variables> / LEVEL=<input-level>;
    TARGET <dependent-variables> / LEVEL=<target-level>;
    ARCHI GLIM;
    TRAIN;
RUN;
```

The **FITNESS** data set, described in Chapter 2, is again used. The results are compared to the parameters that were generated using the Neural Network node.

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **linear perceptron** in the **Diagram Name** field and click **OK**.
3. Drag and drop the **FITNESS** data source onto the diagram and add a **SAS Code** node (Utility tab) to the diagram. Connect it to the **FITNESS** data.

4. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.
5. In the (default) Training Code editor, right-click in the white space and select **Open**.
6. Select the program **c4s1d1.sas** (in d:\workshop\winsas\dmnn41). The program creates a three-valued discrete target (**target_d**) from the continuous target (**oxygen_consumption**).

```
proc rank data=&EM_IMPORT_DATA groups=3 out=&EM_EXPORT_TRAIN;
   var oxygen_consumption;
   ranks target_d;
run;
```

Because the RANK procedure was used to discretize the continuous variable, the resulting classes will all have approximately the same number of observations.

Next, the augmented input data source is converted into a DMDB and cataloged.

```
proc dmdb data=&EM_EXPORT_TRAIN dmdbcat=cat;
   class target_d(desc);
   var performance oxygen_consumption;
run;
```

Notice that the discrete target variable **target_d** had to be changed to descending order (**desc**). This is because the Neural Network node and PROC NEURAL differ in the internal ordering used.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Modeling Continuous Targets

First, the three continuous target distributions are modeled. The PROC NEURAL syntax to match the GENMOD procedure's parameter estimates under the assumption of a ***normal*** target is shown below.

```
title1 'Normally Distributed Target';
proc neural data=&EM_EXPORT_TRAIN dmdbcat=cat;
   input performance / level=int;
   target oxygen_consumption / level=int;
   archi glim;
   train;
run;
```

Notice that no activation or error function needs to be specified in the TARGET statement. Normal is the default distribution assumption when the target is continuous.

The NEURAL procedure syntax for a ***Poisson***-distributed target is listed next.

```
title1 'Poisson Distributed Target';
proc neural data=&EM_EXPORT_TRAIN dmdbcat=cat;
   input performance / level=int;
   target oxygen_consumption / level=int act=exp error=poisson;
   archi glim;
   train;
run;
```

The PROC NEURAL syntax when the target is ***gamma*** distributed follows:

```
title1 'Gamma Distributed Target';
proc neural data=&EM_EXPORT_TRAIN dmdbcat=cat;
   input performance / level=int;
   target oxygen_consumption / level=int act=exp error=gamma;
   archi glim;
   train;
run;
```

Modeling Discrete Targets

The syntax to model a ***multinomial*** target is given below.

```
title1 'Multinomial Target (Generalized Logit)';
proc neural data=&EM_EXPORT_TRAIN dmdbcat=cat;
   input performance / level=int;
   target target_d / level=nom;
   archi glim;
   train;
run;
```

Now **group** is the target variable. To fit this new target as a multinomial variable, the level in the TARGET statement must be changed to **nom** (nominal).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Finally, the syntax required to fit **target_d** under the assumption that it is ordinal is below. Only the level of the target changes (from **level=nom** to **level=ord**).

```
title1 'Ordinal Target (Cumulative Logit)';
proc neural data=&EM_EXPORT_TRAIN dmdbcat=cat;
   input performance / level=int;
   target target_d / level=ord;
   archi glim;
   train;
run;
```

7. Run the program from within the SAS Code node by clicking the running man icon at the top of the window. Confirm the run and save your changes when asked.

8. Examine the results when training is complete.

9. To see PROC NEURAL's estimates when the target has a ***normal*** distribution, go to line **101**.

```
Normally Distributed Target

The NEURAL Procedure

                       Optimization Results
                        Parameter Estimates
                                                    Gradient
                                                   Objective
  N Parameter                             Estimate    Function

  1 Performance_Oxygen_Consumption        0.226446   2.842171E-13
  2 BIAS_Oxygen_Consumption              34.548745   3.208903E-15
```

The values correspond both to PROC GENMOD's estimates and the Neural Network node results when the target distribution is normal.

10. To see PROC NEURAL's estimates when the target is ***Poisson*** distributed, scroll to line **197**.

```
Poisson Distributed Target

The NEURAL Procedure

                       Optimization Results
                        Parameter Estimates
                                                    Gradient
                                                   Objective
  N Parameter                             Estimate    Function

  1 Performance_Oxygen_Consumption        0.004730    0.000005225
  2 BIAS_Oxygen_Consumption               3.586540   4.3404534E-9
```

Again, the values match both to PROC GENMOD's estimates and the Neural Network node results when the target variable is Poisson distributed.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

11. To see PROC NEURAL's estimates when the target is ***gamma*** distributed, scroll to line **293**.

```
Gamma Distributed Target

The NEURAL Procedure

                       Optimization Results
                        Parameter Estimates
                                                        Gradient
                                                       Objective
  N Parameter                              Estimate      Function

  1 Performance_Oxygen_Consumption         0.004766    -0.000000143
  2 BIAS_Oxygen_Consumption                3.584453    6.669134E-11
```

For a third time, the values correspond (within rounding error) to both PROC GENMOD's estimates and the Neural Network node results when the target distribution is gamma.

12. To see PROC NEURAL's estimates when the target is ***multinomial*** (nominal), scroll to line **408**. The values correspond to PROC LOGISTIC's generalized logit link function estimates.

```
Multinomial Target (Generalized Logit)

The NEURAL Procedure

                   Optimization Results
                   Parameter Estimates
                                                 Gradient
                                                Objective
  N Parameter                     Estimate       Function

  1 Performance_target_d2         0.130942       0.002486
  2 Performance_target_d1         0.053029       0.001823
  3 BIAS_target_d2               -7.461726    0.000048673
  4 BIAS_target_d1               -2.525934    0.000044849
```

13. To see PROC NEURAL's estimates when the target is ***ordinal***, scroll to line **528**. The values correspond to both PROC LOGISTIC's cumulative logit link function estimates.

```
Ordinal Target (Cumulative Logit)

The NEURAL Procedure

                   Optimization Results
                   Parameter Estimates
                                                 Gradient
                                                Objective
  N Parameter                     Estimate       Function

  1 Performance_target_d2         0.095330      -0.019915
  2 BIAS_target_d2               -6.377485      -0.000153
  3 BIAS_target_d1               -4.269437      -0.000104
```

14. Close the Results window and the SAS Code node.

15. Close the "linear perceptrons" diagram.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Summary

Given that PROC NEURAL underlies the Neural Network node's functionality, and that the Neural Network node was already seen (in Chapter 2) to produce estimates that correspond to the values output by the GENMOD and LOGISTIC procedures, it is perhaps not surprising that the NEURAL procedure's parameter estimates also closely correspond to those generated by the conventional statistical methods.

The following table summarizes the results.

Distribution	Parameter Identifiers		Parameter Estimates	
	Standard Method	PROC NEURAL	Standard Method	PROC NEURAL
Normal	Intercept	BIAS_Oxygen_Consumption	34.5487	34.548745
	Performance	Performance_Oxygen_Consumption	0.2264	0.226446
Poisson	Intercept	BIAS_Oxygen_Consumption	3.5865	3.586540
	Performance	Performance_Oxygen_Consumption	0.0047	0.004730
Gamma	Intercept	BIAS_Oxygen_Consumption	3.5845	3.584453
	Performance	Performance_Oxygen_Consumption	0.0048	0.004766
Multinomial	Intercept 1	BIAS_target_d1	-2.5276	-2.525934
	Intercept 2	BIAS_target_d2	-7.4648	-7.461726
	Performance 1	Performance_target_d1	0.0530	0.053029
	Performance 2	Performance_target_d2	0.1310	0.130942
Ordinal	Intercept 1	BIAS_target_d1	-6.3897	-6.377485
	Intercept 2	BIAS_target_d2	-4.2804	-4.269437
	Performance	Performance_target_d2	0.0956	0.095330

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.01 Poll

PROC DMDB must be called to catalog the input to PROC NEURAL every time that the input data set is used.

- ❍ True
- ❍ False

9

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

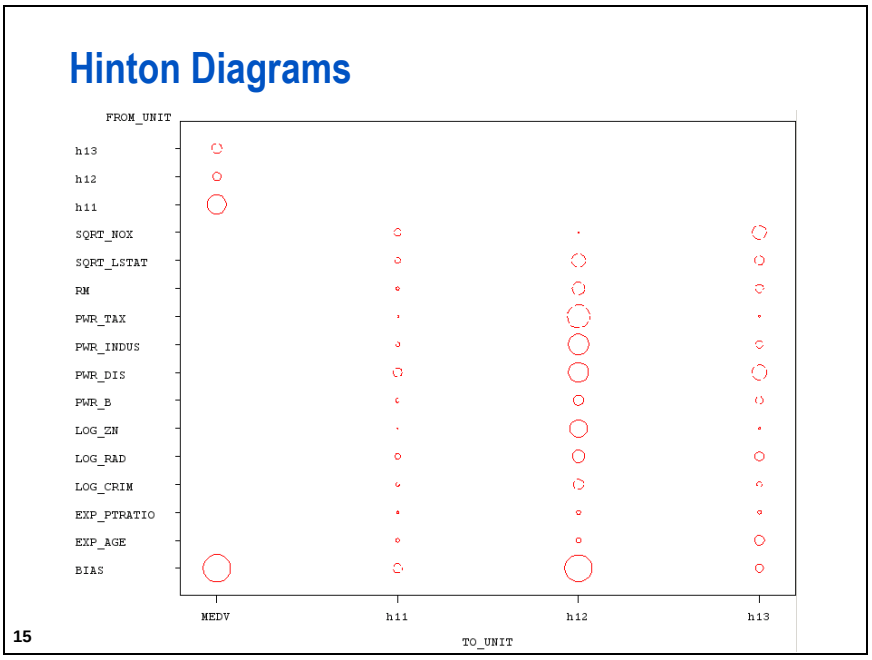
4.2 Input Selection Using PROC NEURAL

Objectives

- Outline the difficulties associated with using weight interpretation to select relevant input variables.
- Illustrate the use of sensitivity-based pruning (SBP) as a means of selecting relevant input variables.

14

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



One approach to assessing which inputs are relevant is to borrow a page from regression modeling and examine the relative magnitude of the parameters associated with each input. In the context of a neural network, this means examining the input-to-hidden unit weights.

This can conveniently be done though a Hinton diagram, which displays the final weight values as a plot. The magnitude of each weight is usually represented as an area or radius, with larger weights displayed as either larger areas or larger radii. Hinton diagrams facilitate the following input pruning strategy:

1. Train the neural network.
2. Generate a Hinton diagram.
3. Prune the input with the input-to-hidden weight closest to zero.
4. Re-train the neural network.
5. If the predictive power increases, then repeat. If not, then reconnect the input and stop.

Unfortunately, in addition to being time consuming, this strategy can lead to inappropriate pruning. An input might only marginally activate a given hidden unit, suggesting the input can be pruned but, if the input connects with many hidden units, it could still yield a substantial overall effect.

Conversely, a large weight does not necessarily mean that the input is important. The hidden unit to which it connects might have only a small weight linking it to the output neurons. A given input might even strongly activate two hidden units that connect the output units in opposing ways.

The weight interpretation problem becomes even more complicated when you realize that there are a number of networks that perform equivalently, each emphasizing different features in the data.

A variation of the Hinton diagram is directly available in the Neural Network node. It can be seen by selecting **View** ⇨ **Model** ⇨ **Weights-Final** when displaying the neural network's results.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Sum of Products

$$\text{importance}(x_i) = \sum_{j=1}^{n} \left| w(x_i, \mathbf{H}_j) * w(\mathbf{H}_j, y) \right|$$

- This formula combines the input-to-hidden weights, $w(x_i, \mathbf{H}_j)$, and hidden-to-output weights, $w(\mathbf{H}_j, y)$.
- If the importance value is small, the input can usually be pruned with impunity.
- However, a large importance value does not imply that the input is important.

16

Because the output of a single-layer network depends on both the input-to-hidden weights, $w(x_i, \mathbf{H}_j)$, and the hidden-to-output weights, $w(\mathbf{H}_j, y)$, it makes sense to try to combine these two factors into a single measure of importance. Here is one formula that has been suggested by several authors:

$$\text{importance}(x_i) = \sum_{j=1}^{n} \left| w(x_i, \mathbf{H}_j) * w(\mathbf{H}_j, y) \right|$$

This formula defines the importance of variable x_i as the sum of the absolute value of the product of the input-to-hidden weights and the hidden-to-output weights, across all n hidden units to which it connects. The formula yields an upper bound for the change in output y corresponding to a given change in input x_i. Thus, ***when the sum of products value of input x_i is a small number, you can validly conclude that x_i is unimportant***. A large number, however, does ***not*** necessarily mean that the input is important.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Variable Importance

c4s2d1

In this demonstration, you calculate input importance using the sum of products methodology. Input variables generating relatively small sum of products values are prime candidates for pruning.

The famous Boston **HOUSING** data is used. In the Boston housing study, Harrison and Rubinfeld (1976) used housing market data to measure the willingness to pay for clean air. A common presumption is that individuals will pay more for a housing unit located in an area with good air quality than they will for an otherwise identical unit located in an area of poor air quality.

The variables in the Boston **HOUSING** data set are listed below.

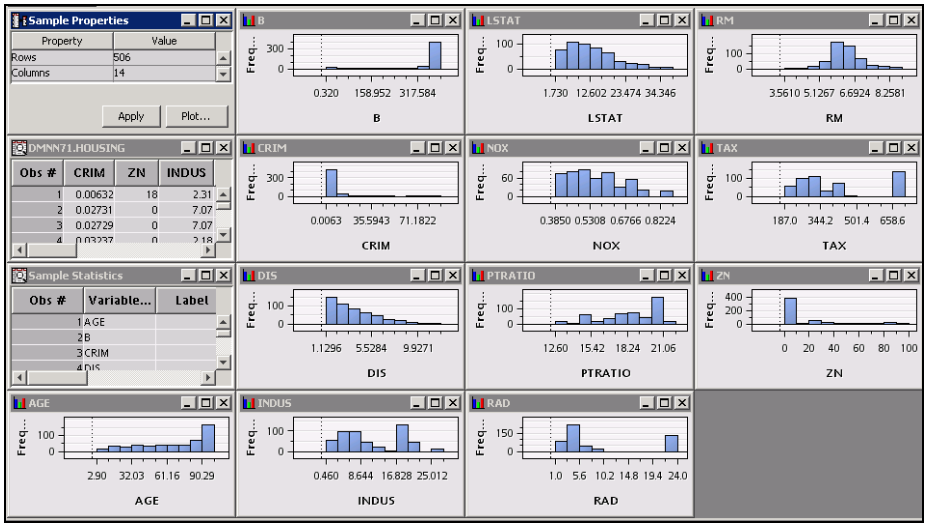
Variable	Description	Role	Level
AGE	Proportion of owner-occupied units built before 1940	Input	Interval
B	$1000(B-0.63)^2$ where B is the proportion of African-American citizens, by town	Input	Interval
CHAS	Charles River indicator (tract bounds river=1, otherwise=0)	Input	Binary
CRIM	Per capita crime rate by town	Input	Interval
DIS	Weighted distances to five Boston employment centers	Input	Interval
INDUS	Proportion of non-residential business acres per town	Input	Interval
LSTAT	Percentage of the population that is lower status	Input	Interval
MEDV	Median value of owner-occupied homes in USD 1K	Target	Interval
NOX	Nitric oxide concentration (parts per 10 million)	Input	Interval
PTRATIO	Pupil-teacher ratio by town school district	Input	Interval
RAD	Index of accessibility to radial highways	Input	Interval
RM	Average number of rooms per dwelling	Input	Interval
TAX	Full-value property tax rate per USD 10K	Input	Interval
ZN	Proportion of residential land zoned for lots more than 25K sq. ft.	Input	Interval

The target variable is **MEDV**, the median value of owner occupied homes (in USD). As in the original study, **MEDV** is assumed to be Poisson distributed.

1. Right-click **Diagrams** in the project panel and select **Create Diagram** from the pop-up menu.
2. Enter **input selection** in the **Diagram Name** field, and click **OK**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3. Right-click **Data Sources** in the project panel and select **Create Data Source**.
4. Click **Next** to use a SAS table as the metadata source.
5. Click **Browse** and select the **HOUSING** data table from the **dmnn41** library.
6. Click **Next**. You see that there are 14 variables and 506 observations in **HOUSING**.
7. Click **Next** again to start the metadata definition process.
8. The **Basic** setting suffices, so click **Next** to proceed.
9. Change the role of **MEDV** to **Target**.
10. Also change the level of **CHAS** to **Binary**.
11. Click **Next** three times and then click **Finish** to complete the metadata definition process.
12. Drag and drop the **HOUSING** data source onto the diagram.
13. In the **HOUSING** data source's properties panel, click the ellipsis (**…**) next to **Variables**.
14. Select all of the input variables (except the binary **CHAS** variable) and click **Explore**. As you can see, many inputs are badly skewed and should be transformed.



15. Close the Explore window.
16. Add a **Transform Variables** node (Modify tab) and connect it to the **HOUSING** data source.
17. In the Default Methods section, change the default method for Interval Inputs to **Maximum Normal** (maximize normality).

Default Methods	
Interval Inputs	Maximum Normal
Interval Targets	None
Class Inputs	None
Class Targets	None
Treat Missing as Level	No

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

18. Right-click the **Transform Variables** node and select **Run** from the pop-up window.

19. Confirm the run and examine the results when the run finishes.

20. Maximize the Output window and scroll to line **23** to see the selected transformations.

```
Computed Transformations
(maximum 500 observations printed)

Input           Input
Name    Role    Level    Name          Level      Formula

AGE     INPUT  INTERVAL  EXP_AGE      INTERVAL    exp(max(AGE-2.9, 0.0)/97.1)
B       INPUT  INTERVAL  PWR_B        INTERVAL    (max(B-0.32, 0.0)/396.58)**4
CRIM    INPUT  INTERVAL  LOG_CRIM     INTERVAL    log(max(CRIM-0.00632, 0.0)/88.96988  + 1)
DIS     INPUT  INTERVAL  PWR_DIS      INTERVAL    (max(DIS-1.1296, 0.0)/10.9969)**0.25
INDUS   INPUT  INTERVAL  PWR_INDUS    INTERVAL    (max(INDUS-0.46, 0.0)/27.28)**0.25
LSTAT   INPUT  INTERVAL  SQRT_LSTAT   INTERVAL    sqrt(max(LSTAT-1.73, 0.0)/36.24)
NOX     INPUT  INTERVAL  SQRT_NOX     INTERVAL    sqrt(max(NOX-0.385, 0.0)/0.486)
PTRATIO INPUT  INTERVAL  EXP_PTRATIO  INTERVAL    exp(max(PTRATIO-12.6, 0.0)/9.4)
RAD     INPUT  INTERVAL  LOG_RAD      INTERVAL    log(max(RAD-1, 0.0)/23  + 1)
TAX     INPUT  INTERVAL  PWR_TAX      INTERVAL    (max(TAX-187, 0.0)/524)**0.25
ZN      INPUT  INTERVAL  LOG_ZN       INTERVAL    log(max(ZN-0, 0.0)/100  + 1)
```

Many of the inputs differ with respect to the transformation that maximizes their normality. Often a log transform sufficed, whereas in other cases, a power transformation was required. Also notice that the **RM** (number of rooms) variable did not need to be transformed.

21. Add a **SAS Code** node (Utility tab) to the diagram.

22. Connect it to the **Transform Variables** node

23. Right-click the **SAS Code** node and select **Rename** from the pop-up menu.

24. Rename the node **Sum of Products**. Your diagram should look similar to the one below.

25. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.

26. In the (default) Training Code editor, right-click in the white space and select **Open**.

27. Select the program **c4s2d1.sas** (in d:\workshop\winsas\dmnn41).

First, the input data source (**HOUSING**) must be converted and cataloged.

```
proc dmdb data=&EM_IMPORT_DATA dmdbcat=cat;
   var %EM_INTERVAL_INPUT %EM_TARGET;
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Then the network to be assessed is constructed.

```
title1 'Sum of Products';
proc neural data=&EM_IMPORT_DATA ranscale=0.1 dmdbcat=cat;
   input %EM_INTERVAL_INPUT / level=int std=std id=in;
   target %EM_TARGET / level=int act=exp error=poisson id=out;
   hidden 3 / id=h1;
   connect in h1 out;
   prelim 5 maxiter=10;
   train maxiter=50 estiter=0 outest=weights outfit=stats;
run;
```

This is the same three-hidden-unit MLP network created in the last demonstration (c4s2d1), only this time it is created using CONNECT statements rather than an ARCHI statement.

A request to print selected goodness-of-fit information follows network construction.

```
title2 'Selected Fit Statistics';
proc print data=stats(firstobs=2) label noobs;
   var _dfm_ _ase_ _aic_ _sbc_;
run;
```

Next, the network's final weight values are transposed and written to the file **transposed**.

```
proc transpose data=weights(where=(_name_='_LAST_')) prefix=_w
           out=transposed;
run;
```

As was done in the previous demonstration, the "from" and "to" units are identified by locating the arrow separator in the **_LABEL_** field. The only difference is that this DATA step outputs the input-to-hidden weights to one file (**OUT1**) and the hidden-to-output weights to another file (**OUT2**).

```
data out1(rename=(to_unit=hidden_unit))
     out2(rename=(from_unit=hidden_unit _w1=_w2));
   set transposed(where=(substr(_name_,1,1) NE '_'));
   from_unit=trim(scan(_label_,1,' -> '));
   to_unit=trim(scan(_label_,2,' -> '));
   if upcase(substr(from_unit,1,2))='H1' then output out2;
   else output out1;
run;
```

After sorting **out1**, the files are then combined using **hidden_unit** in a one-to-many merge and output to **results**. This puts each variable's input-to-hidden weight (**_w1**) and its hidden-to-output weight (**_w2**) in the same observation, thereby facilitating their upcoming multiplication.

```
proc sort data=out1;
   by hidden_unit;
run;

data results(drop=_name_ _label_ hidden_unit to_unit);
   merge out1 out2;
   by hidden_unit;
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The sum or products importance of each input is then calculated directly from the merged file.

```
proc sort data=results;
   by from_unit;
run;

data results;
   set results(where=(from_unit ne 'BIAS'));
   by from_unit;
   if first.from_unit then sum=0;
   sum + abs(_w1*_w2);
   if last.from_unit then output;
run;
```

The resulting sum of products values is sorted into ascending order of importance and displayed.

```
proc sort data=results;
   by sum;
run;

proc print data=results label noobs;
   label from_unit='Variable' sum='Importance';
   var from_unit sum;
run;
```

28. Run the diagram from within the SAS Code node. Confirm the run and save your changes.

29. Examine the results. The final parameter estimates are displayed at line **183**. A subset of the weights are shown below. These values were selected in order to illustrate the calculation of importance value for “the number of rooms” (**RM**) variable.

```
                 Optimization Results
                Parameter Estimates
                                          Gradient
                                         Objective
  N Parameter              Estimate        Function

  1 EXP_AGE_h11            0.312293     0.000023685
  2 EXP_PTRATIO_h11       -0.117188    -0.000085079
 ...
 10 RM_h11                -0.038967     0.000075503

 ...

 22 RM_h12                 0.504743     0.000055755

 ...

 34 RM_h13                -0.080467    -0.000091259

 ...

 40 h11_MEDV               1.813181     0.000002518
 41 h12_MEDV               0.390360    -0.000003790
 42 h13_MEDV              -1.425276     0.000004192
 43 BIAS_MEDV              3.476402     0.000000685

Value of Objective Function = 0.2677418478
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The importance of the **RM** variable would be calculated as follows:

|RM_H11*H11_MEDV| + |RM_H12*H12_MEDV| + |RM_H13*H13_MEDV|

Substituting the corresponding parameter values from the listing above yields the following:

| -0.038967 * 1.813181 | + | 0.504743* 0.39036 | + | -0.080467 * -1.425276 |

= 0.070654224027 + 0.19703147748 + 0.114687683892

= 0.382373385399

30. Scroll to line **250** in the output listing to examine the calculated importance values. Notice that the importance value for the **RM** variable is 0.38237, exactly as calculated above.

```
Sum of Products

Variable       Importance

LOG_ZN           0.36974
EXP_PTRATIO      0.38236
RM               0.38237
PWR_B            0.40214
LOG_RAD          0.44586
PWR_TAX          0.85001
SQRT_LSTAT       0.91767
PWR_INDUS        1.12586
EXP_AGE          1.26118
LOG_CRIM         1.26432
SQRT_NOX         1.43757
PWR_DIS          1.83571
```

The results suggest that the **LOG_ZN** (log transformed proportion of residential land zoned) variable is the least important and, therefore, most dispensable input.

31. However, before deleting **LOG_ZN**, scroll back up a little (to line **238**) and examine the current model's goodness of fit information.

```
Selected Fit Statistics

                 Train:       Train:        Train:
Train: Model    Average      Akaike's     Schwarz's
 Degrees of     Squared    Information     Bayesian
   Freedom       Error      Criterion     Criterion

     43         5.56492      221.477       403.218
```

This information can be used to assess the merit of deleting **LOG_ZN**. If the fit information improves after deletion, then the deletion was beneficial, and further deletions can be considered. Conversely, if removal of **LOG_ZN** was detrimental, then the variable should be reinstated and pruning terminated.

32. Close the Results window and the SAS Code node.

33. To remove the **LOG_ZN** variable, click the ellipsis (**…**) next to **Variables** in the SAS Code node's properties panel, and set the Use status of **LOG_ZN** to **No**. Click **OK** to save the change.

34. Right-click the **SAS Code** node and select **Run** from the pop-up menu.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

35. Examine the results when training is complete.

36. Scroll to the new fit statistics information (at line **237**).

```
Selected Fit Statistics

                 Train:       Train:        Train:
Train: Model    Average      Akaike's     Schwarz's
 Degrees of     Squared    Information     Bayesian
   Freedom       Error      Criterion     Criterion

     40         5.90509      220.841       389.903
```

Although the average squared error rose slightly, both the AIC and SBC went down, implying that pruning the transformed zone variable (**LOG_ZN**) was beneficial.

37. Scroll down to line **248** to see the next candidate for deletion. It is the **EXP_PTRATIO** variable.

```
Sum of Products
Selected Fit Statistics

Variable       Importance

EXP_PTRATIO      0.12210
RM               0.37748
LOG_RAD          0.38531
PWR_B            0.50458
PWR_INDUS        0.63797
PWR_TAX          1.11563
SQRT_LSTAT       1.22001
LOG_CRIM         1.35416
EXP_AGE          1.42748
SQRT_NOX         1.84168
PWR_DIS          2.60392
```

38. Close the Results window and the SAS Code node.

39. Remove the **EXP_PTRATIO** variable as before, by clicking the ellipsis (**…**) next to **Variables** in the SAS Code node's properties panel, and setting its Use status to **No**. Click **OK** to save the change.

40. Right-click the **SAS Code** node and select **Run** from the pop-up menu.

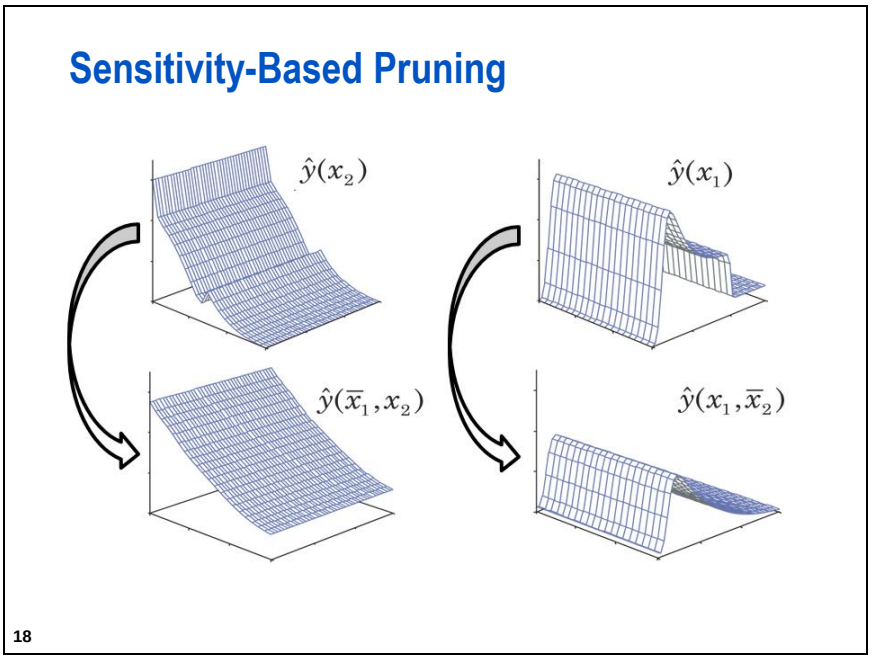
41. In this case, the fit information (line **223**) suggests that the removal was detrimental, so reset the Use status of **EXP_PTRATIO** to **Default**.

```
Sum of Products
Selected Fit Statistics

                 Train:       Train:        Train:
Train: Model    Average      Akaike's     Schwarz's
 Degrees of     Squared    Information     Bayesian
   Freedom       Error      Criterion     Criterion

     37         7.48509      248.199       404.580
```

42. Close the Results window and the SAS Code node.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Unfortunately, the sum of products approach does not take into account the squashing effect of the hidden layer activation function (Sarle 2000). The amount of squashing depends, not only on the weight of the input, but also on the values and the weights of other inputs in the network. If you take these complexities into account, you end up using a measure of importance based on the computed input-output function. One such measure is *sensitivity-based pruning* (Moody 1994).

Sensitivity-based pruning reflects the change in model fit when each input is successively held constant at a typical value, but the other inputs are allowed to vary. It is similar to a likelihood-ratio test, in that it attempts to measure the change in error caused by the removal of an input variable. However, unlike the likelihood-ratio test, sensitivity-based pruning does not require re-training the model, a task that can be computationally expensive with neural networks. The methodology is detailed below.

1. Select the ith input variable.

2. Calculate a location (mean, percentile) value for the ith input. Because the sensitivity measure might, itself, be sensitive to the location value at which the input is held constant, it is usually a good idea to assess the variable's sensitivity at several location values.

3. Merge the constant location value calculated in step 2 with the original data set.

4. Recall the scoring code containing the neural network model, and score the merged data using the calculated location rather than the original value of the ith input.

5. Determine the difference between the fit (deviance) measure when the original input value is used, and the deviance when the constant location is substituted.

Input redundancy can cause this sensitivity measure to overvalue certain inputs. Conversely, interactions can cause it to undervalue inputs.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

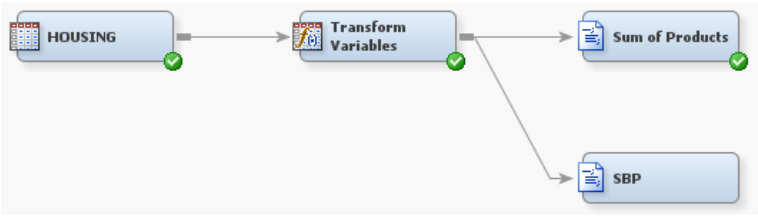
Sensitivity-Based Pruning

c4s2d2

This demonstration implements *sensitivity-based pruning* as a means of selecting a relevant set of inputs for a neural network. Sensitivity-based pruning reflects the change in the fit when each input is, in turn, held constant while the other inputs are allowed to vary. The transformed **HOUSING** data is again used.

1. Add a **SAS Code** node (Utility tab) to the diagram. Connect it to the **Transform Variables** node.

2. Right-click the **SAS Code** node and select **Rename**. Rename the node **SBP**. Your diagram should look similar to the one below.



3. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.

4. In the (default) Training Code editor, right-click in the white space and select **Open**. Select the program **c4s2d2.sas** (in d:\workshop\winsas\dmnn41). The program is described below.

   After initializing the global system options (not shown), a macro variable (**codefile**) is declared. This macro variable points to the location of the file containing the trained neural network.

```
%let codefile=d:\workshop\winsas\dmnn41\code.sas;
```

   The input data set it then converted into a DMDB and cataloged.

```
proc dmdb data=&EM_IMPORT_DATA dmdbcat=cat;
   var %EM_INTERVAL_INPUT %EM_TARGET;
run;
```

   Next, the neural network is set up and trained. **Notice that a CODE statement has been added.** It writes SAS DATA step scoring code to the file pointed to by the **codefile** macro variable.

```
title1 'Sensitivity Based Pruning';
proc neural data=&EM_IMPORT_DATA ranscale=0.1 dmdbcat=cat;
   input %EM_INTERVAL_INPUT / level=int std=std id=in;
   target %EM_TARGET / level=int act=exp error=poisson id=out;
   hidden 3 / id=h1;
   connect in h1 out;
   prelim 5 maxiter=10;
   train maxiter=50 estiter=0 outest=weights outfit=stats;
   code file="&codefile";
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Selected information reflecting the network's performance on the training data is then requested.

```
title2 'Selected Fit Statistics';
proc print data=stats(firstobs=2) label noobs;
   var _dfm_ _ase_ _aic_ _sbc_;
run;
```

The definition of the SPB (sensitivity-based pruning) macro follows network creation.

```
%macro sbp(dsn=,fit=,q=);
   data _null_;
      set &fit(firstobs=2);
      call symputx('error',_err_);
      call symputx('predicted',%str("p_%EM_TARGET"));
      call symputx('ninputs',length("%cmpres(%EM_INTERVAL_INPUT)"-
         length(compress("%cmpres(%EM_INTERVAL_INPUT)"))+1);
   run;
   proc sql;
      create table results(variable char(25),sensitivity num);
   quit;
   proc univariate data=&dsn noprint;
      var %EM_INTERVAL_INPUT;
      output &q=%EM_INTERVAL_INPUT out=quantiles;
   run;
   %do i=1 %to &ninputs;
      data _null_;
         call symputx('_x_',scan("%EM_INTERVAL_INPUT",&i));
      run;
      data temp;
         if _n_=1 then set quantiles(keep=&_x_);
         set &dsn(drop=&_x_);
         %include "&codefile";
         dev=2*(%EM_TARGET*log(%EM_TARGET/&predicted)-
            (%EM_TARGET-&predicted));     /* Poisson deviance */
      run;
      proc sql;
         insert into results(Variable,Sensitivity)
            select "&_x_",sum(dev)-&error from temp;
      quit;
   %end;
   proc sort data=results;
      by Sensitivity;
   run;
   title2 "Quantile = &q";
   proc print data=results noobs;
      var Variable Sensitivity;
   run;
%mend sbp;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The SBP macro function begins by defining three macro variables. The first, **error**, holds the neural network's final deviance. The **predicted** variable holds the estimated housing price. And **ninputs** stores the number of (interval level) input variables.

Next, the output table (**results**) is initialized. It contains two columns: **Variable**, the input variable being evaluated, and **Sensitivity**, the variable's sensitivity.

PROC UNIVARIATE is then called in order to generate each input's typical value at the specified quantile (**q**). It sends the calculated quantile output to **quantiles**.

The heart of the SBP macro is the DO loop that follows the call to PROC UNIVARIATE. A call to **SYMPUTX** stores the current variable being evaluated in a macro variable (**_x_**). Then the data set containing the quantiles is read in. Only the typical value for the considered input is retained. The input data set is also read. After dropping the variable being evaluated, the input data is merged with the quantile. This replaces the actual value of the input with its quantile. The %INCLUDE statement reads in the scoring code written out by PROC NEURAL. It also applies the scoring code to the new data set, **temp**, returning the predicted median housing value. The final component of the DO loop is a call to PROC SQL, which calculates the Poisson deviance of the new data and subtracts it from the deviance of the original model. The input variable that was assessed and its computed sensitivity value are written to **results**. The **results** data set is then sorted (ascending order) and printed. The SBP macro is invoked three times, requesting that each input's first (**Q1**), second (**MEDIAN**), and third (**Q3**) quartiles be used as the substituted constant values when assessing input sensitivity.

```
%sbp(dsn=&EM_IMPORT_DATA,fit=stats,q=Q1)
%sbp(dsn=&EM_IMPORT_DATA,fit=stats,q=MEDIAN)
%sbp(dsn=&EM_IMPORT_DATA,fit=stats,q=Q3)
```

5. Run the program from within the SAS Code node. Confirm the run and examine the results when they become available.

6. Scroll to line **240** to see the goodness of fit values.

```
Selected Fit Statistics

                 Train:       Train:        Train:
Train: Model    Average      Akaike's     Schwarz's
 Degrees of     Squared    Information     Bayesian
   Freedom       Error      Criterion     Criterion

      43         5.56492      221.477       403.218
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

7. Next scroll to line **252** to see the variable sensitivity when each input is, in turn, held constant at its first quartile (25th percentile) value.

```
Quantile = Q1

variable       sensitivity

LOG_ZN             23.18
PWR_B              39.45
EXP_PTRATIO        78.29
LOG_RAD           104.07
LOG_CRIM          186.28
PWR_INDUS         381.90
EXP_AGE           421.11
RM                473.04
PWR_TAX           583.05
PWR_DIS           661.41
SQRT_LSTAT        749.17
SQRT_NOX         1283.35
```

The **LOG_ZN** variable generated the smallest sensitivity value, implying that it is most dispensable. Interestingly, this is the same variable that was suggested by the Sum of Products. The **SQRT_NOX** variable seems to generate the largest change in output when it is held constant at its 25th percentile value, suggesting that it is the least dispensable input.

8. To see the sensitivity when each input's median (50% percentile) is used, scroll down to line **272**.

```
Quantile = MEDIAN

variable       sensitivity

LOG_ZN            23.185
PWR_B             51.079
EXP_PTRATIO       51.122
LOG_RAD           93.653
PWR_INDUS        125.833
EXP_AGE          159.933
LOG_CRIM         180.698
SQRT_NOX         294.643
RM               349.265
PWR_TAX          352.499
SQRT_LSTAT       374.288
PWR_DIS          514.822
```

Again, the transformed **ZN** variable (**LOG_ZN)** would appear to be the most dispensable input. However, this time **PWR_DIS** appears to be the least dispensable variable.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

9. The input ranking at the third quartile (75 percentile) appears at line **292**. For a third time, **LOG_ZN** generated the smallest value. **PWR_DIS** is again the least dispensable variable.

```
Quantile = Q3

variable       sensitivity

LOG_ZN            20.732
PWR_B             59.032
EXP_PTRATIO       77.429
PWR_INDUS        126.981
SQRT_NOX         135.013
EXP_AGE          171.877
PWR_TAX          199.207
RM               308.519
SQRT_LSTAT       371.514
LOG_RAD          434.690
LOG_CRIM         666.122
PWR_DIS          890.771
```

10. Close the Results window and the SAS Code node.

As before, the strategy would be to remove **LOG_ZN** by setting it to "don't use," and rerun the analysis. If the goodness of fit information improves, then consider removing the next suggested input. Otherwise, restore the **LOG_ZN** variable and stop.

11. Close the "input selection" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.02 Poll

Pruning the input with the smallest input-to-hidden weight can negatively impact the neural network's performance.

- ❍ True
- ❍ False

20

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Exercises

1. Sensitivity-Based Pruning (Bernoulli Distributed Target)

This exercise illustrates the sensitivity-based pruning input selection methodology, this time applied to a discrete target variable. The **BANKING** data set (defined in Chapter 3) is again used.

a. Right-click **Diagrams** in the project panel, and select **Create Diagram** from the pop-up menu. Enter **exercise 4.1** in the **Diagram Name** field, and click **OK**.

b. Drag and drop the **BANKING** data source onto the diagram.

c. Add a **SAS Code** node (Utility tab) to the flow, and connect it to the **BANKING** data source.

d. In the (default) Training Code editor, right-click in the white space and select **Open**. Select the program **exercise 4.1.sas** (in d:\workshop\winsas\dmnn41).

e. Complete the NEURAL procedure code by filling in the missing INPUT, TARGET, and ARCHI statements. Construct a three-hidden-unit MLP. Only interval inputs are to be considered, and they should all be Z-score standardized using the STD=STD option. The target level is nominal.

```
proc dmdb data=&EM_IMPORT_DATA dmdbcat=cat;
   class %EM_TARGET(desc);
   var %EM_INTERVAL_INPUT;
run;
```

The code to construct a macro variable (**predicted1**), which points to the variable containing probability of the 1 event (i.e., acquiring the banking product), has been defined for you.

```
call symputx('predicted1',%str("p_%EM_TARGET")||'1');
```

And because the target variable is now binary, the Poisson deviance function has been changed to the Bernoulli deviance function (below). Entropy deviance could also have been specified.

```
dev=-2*(%EM_TARGET*log(&predicted1)+
             (1-%EM_TARGET)*log(1-&predicted1));
```

f. Run the diagram from within the SAS Code node, and answer the following questions:

- When each input is held constant at the first quartile, which variable is least sensitive?
- Which variable is least sensitive when the inputs are held constant at the median?
- When the third quartile is used as the constant value, which variable is least sensitive?

g. Close the SAS Code node's results, and the "exercise 4.1" diagram.

End of Exercises

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

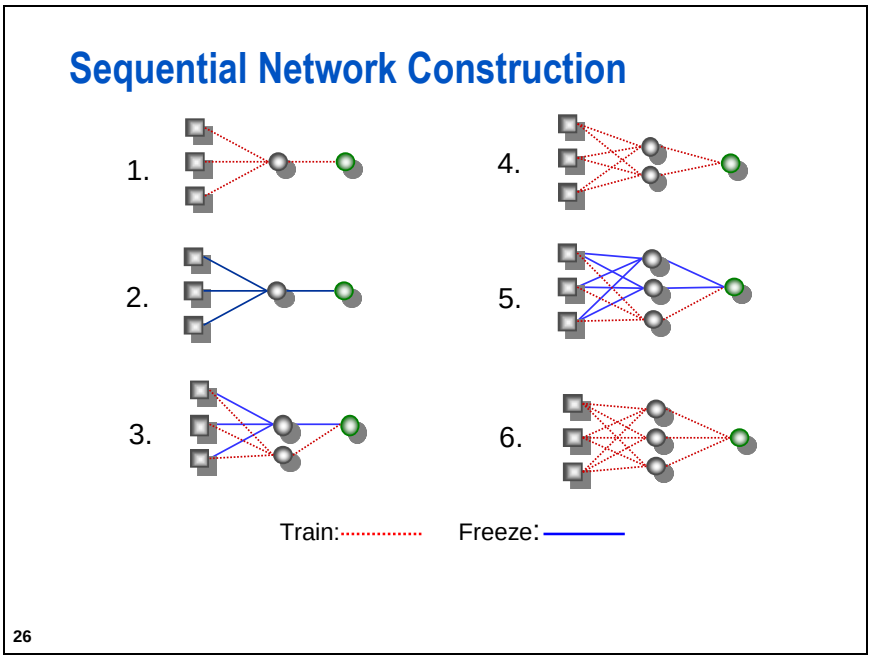
4.3 Sequential Network Construction

Objectives

- Define Sequential Network Construction (SNC).
- Illustrate the SNC paradigm.

25

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



PROC NEURAL can also be used to construct neural networks in a principled way. For example, the *sequential network construction* methodology addresses the difficult question "How many hidden units are required to adequately fit the data?" by building the hidden layer one unit at a time. The sequential network construction methodology is implemented as follows:

1. Create and train a network consisting of an input layer, a hidden layer consisting of a single hidden unit, and an output layer. Fully connect the layers.
2. Freeze the weights in the trained network, except for the bias term in each output unit.
3. Add a new hidden unit, connect it to all input and output units (but not to the other hidden units), and re-train the network. Only the output biases and the connections to the new hidden unit will change.
4. Thaw all frozen weights, and train the entire network so that all of the hidden units cooperate.
5. Determine the change in goodness of fit after the hidden unit was added. If it degenerates, remove the last hidden unit that was added, re-train the network, and stop. Otherwise, go to step 2.

Because the final parameter estimates at each step become the starting values for subsequent steps, in essence, this methodology fits each hidden unit to the residuals generated by the previous network.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Sequential Network Construction

c4s3d1

This demonstration builds an MLP using sequential network construction. In sequential network construction, neurons are added to the hidden layer one at a time. The network with the best performance has the recommended number of hidden units. In this case, the Schwarz-Bayesian criterion will be used to determine "best performance." The Boston **HOUSING** data, described previously, is again used.

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **snc** in the **Diagram Name** field and click **OK**.
3. Drag and drop the **HOUSING** data source onto the diagram.
4. As before, because many of the input variables are severely skewed, add a **Transform Variables** node (Modify tab) and connect it to the **HOUSING** data source.
5. Click in the Default Methods section, and change the Interval Inputs value to **Maximum Normal**.
6. Finally, add a **SAS Code** node (Utility tab) and connect it to the **HOUSING** data source.

7. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.
8. In the (default) Training Code editor, right-click in the white space and click **Open**. Select the program **c4s3d1.sas** (in d:\workshop\winsas\dmnn41). The program is described below.

   First, the **HOUSING** data set is converted to a data mining database (DMDB) and then cataloged.

```
proc dmdb data=&EM_IMPORT_DATA dmdbcat=cat;
   var %EM_INTERVAL_INPUT %EM_TARGET;
run;
```

Next, sequential network construction begins. A one-hidden-unit network is constructed and the fit measured. The six-step sequential network construction process (below) then continues iteratively.

a. Freeze the current network (except for the output bias).
b. Add a new hidden unit, and fully connect it to the input and output units.
c. Perform preliminary training and then training of the newly added hidden unit.
d. Thaw all weights that are currently frozen and train the network.
e. Measure the current network's performance.
f. If the performance improves, go to step a. Otherwise, remove the last hidden unit and stop.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The three-hidden-unit network generated using the SNC process is given below.

```
title1 'Sequential Network Construction';
proc neural data=&EM_IMPORT_DATA dmdbcat=cat;
   input %EM_INTERVAL_INPUT/level=int id=in;
   target %EM_INTERVAL_TARGET/level=int act=exp error=poisson
          id=out;

   title2 'One Hidden Unit Network';
   hidden 1 /id=h1;
   connect in h1 out;
   prelim 5 maxiter=10;
   train outfit=fit1(where=(_name_='OVERALL'));

   title2 'Two Hidden Unit Network';
   freeze in->h1 h1->out bias->h1;
   hidden 1 /id=h2;
   connect in h2 out;
   prelim 5 maxiter=10;
   train;
   thaw in->h1 h1->out bias->h1;
   train outfit=fit2(where=(_name_='OVERALL'));

   title2 'Three Hidden Unit Network';
   freeze in->h1 h1->out bias->h1 in->h2 h2->out bias->h2;
   hidden 1 /id=h3;
   connect in h3 out;
   prelim 5 maxiter=105;
   train;
   thaw in->h1 h1->out bias->h1 in->h2 h2->out bias->h2;
   train outfit=fit3(where=(_name_='OVERALL'));
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The final component of the program simply displays the desired goodness-of-fit information.

```
title2 'One Hidden Unit Network';
proc print data=fit1 noobs label;
   var _sbc_;
run;

title2 'Two Hidden Unit Network';
proc print data=fit2 noobs label;
   var _sbc_;
run;

title2 'Three Hidden Unit Network';
proc print data=fit3 noobs label;
   var _sbc_;
run;
```

The recommended number of hidden units is the network that yielded the best fit to the data, in this case, the network generating the best Schwarz Bayesian Criterion (**_SBC_**) value.

9. Run the diagram from within the SAS Code node. Confirm the run and save the program.

10. Examine the results when training is complete.

11. To see the results of the preliminary training runs for the one hidden unit network, go to line **23.**

```
One Hidden Unit Network

The NEURAL Procedure

Preliminary       Starting         Objective         Number
 Training          Random           Function           of        Terminating
    Run             Seed             Value         Iterations     Criteria

        1               12345    0.573782692303           10
        2           782179818    0.575181756728           10
        3          1866834797    0.577592876397           10
        4          1384285978    0.572254453434           10
        5          1784231189    0.586606042606           10
```

As requested, five preliminary training runs were conducted, each lasting 10 iterations. The fourth preliminary training run had the smallest objective function value. The parameter values determined at the end of this run serve as starting values for training.

These starting weights will not be reproduced here. They are listed in the output window, beginning at line **45**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

12. Scroll to line **155** to see the final weight values for the one-hidden-unit network.

```
                 Optimization Results
                 Parameter Estimates
                                          Gradient
                                         Objective
  N Parameter              Estimate       Function

  1 EXP_AGE_h11           -0.004993       0.001085
  2 EXP_PTRATIO_h11       -0.031611       0.002293
  3 LOG_CRIM_h11          -0.113223       0.000820
  4 LOG_RAD_h11            0.066755       0.001230
  5 LOG_ZN_h11            -0.003890      -0.001390
  6 PWR_B_h11              0.011713      -0.000713
  7 PWR_DIS_h11           -0.087211      -0.000173
  8 PWR_INDUS_h11         -0.010684       0.001827
  9 PWR_TAX_h11           -0.025711       0.001554
 10 RM_h11                 0.033846      -0.003130
 11 SQRT_LSTAT_h11        -0.097869       0.003211
 12 SQRT_NOX_h11          -0.050488       0.001436
 13 BIAS_h11              -0.529681      -0.004073
 14 h11_MEDV               2.661934       0.000664
 15 BIAS_MEDV              4.307912      -0.001820

Value of Objective Function = 0.5669432348
```

These values serve as the initial parameters for preliminary training in the next iteration.

13. Next the preliminary training associated with the added second hidden unit appears (line **183**).

```
Two Hidden Unit Network

The NEURAL Procedure

Preliminary       Starting         Objective         Number
 Training          Random           Function           of        Terminating
    Run             Seed             Value         Iterations     Criteria

        1           305220667    0.561698189561           10
        2          1548793169    0.489686461694           10
        3          1816732956    0.514342832012           10
        4          1023729404    0.546011884056           10
        5           719546802      0.5655321552           10
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

14. Because the second preliminary run generates the smallest objective function value, its weights become the starting values for training. These weights are displayed, beginning at line **204**.

```
                 Optimization Start
                Parameter Estimates
                                          Gradient
                                         Objective
  N Parameter              Estimate        Function

  1 EXP_AGE_h21            3.293338       -0.000455
  2 EXP_PTRATIO_h21       -0.948082       -0.000425
  3 LOG_CRIM_h21           2.881652       -0.002611
  4 LOG_RAD_h21            2.717189       -0.006424
  5 LOG_ZN_h21            -0.464169        0.000204
  6 PWR_B_h21             -3.214144        0.002608
  7 PWR_DIS_h21           -7.810860        0.001387
  8 PWR_INDUS_h21         -0.811495       -0.001684
  9 PWR_TAX_h21           -0.783642       -0.004477
 10 RM_h21                -6.795325        0.000629
 11 SQRT_LSTAT_h21        -8.399428       -0.002822
 12 SQRT_NOX_h21          -7.129702       -0.002260
 13 BIAS_h21               3.979852        0.001525
 14 h21_MEDV              -0.068041        0.024420
 15 BIAS_MEDV              4.330556        0.003741

Value of Objective Function = 0.4896864617
```

Notice that the frozen weights (for example, EXP_AGE_h11) do not appear in this listing. Only weights that are free to change their value appear.

15. The final weights (shown below) of the partially frozen two-hidden-unit network appear at line **325**.

```
                Optimization Results
                Parameter Estimates
                                          Gradient
                                         Objective
  N Parameter              Estimate        Function

  1 EXP_AGE_h21           29.052194        0.000181
  2 EXP_PTRATIO_h21      -11.327585     0.000074459
  3 LOG_CRIM_h21          30.753520     0.000015969
  4 LOG_RAD_h21           27.889927        0.000178
  5 LOG_ZN_h21            -5.489417       -0.000213
  6 PWR_B_h21            -29.131695     0.000015961
  7 PWR_DIS_h21          -57.301461       -0.000147
  8 PWR_INDUS_h21         -7.109840        0.000208
  9 PWR_TAX_h21           -9.421459        0.000188
 10 RM_h21               -55.328801    -0.000018258
 11 SQRT_LSTAT_h21       -68.814608        0.000128
 12 SQRT_NOX_h21         -56.688132        0.000168
 13 BIAS_h21              32.576510     0.000028133
 14 h21_MEDV              -0.064748        0.000216
 15 BIAS_MEDV              4.325385     0.000070067

Value of Objective Function = 0.4827440354
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

16. Next the frozen weights are thawed so that the two hidden units can work together. Now all of the network weights appear in the listing. The starting weight values begin at line **357**.

```
                 Optimization Start
                Parameter Estimates
                                                Gradient
                                               Objective
  N Parameter              Estimate             Function

  1 EXP_AGE_h11           -0.004993            -0.771146
  2 EXP_PTRATIO_h11       -0.031611             0.331259
  3 LOG_CRIM_h11          -0.113223            -0.579223
  4 LOG_RAD_h11            0.066755            -0.704773
  5 LOG_ZN_h11            -0.003890             0.354882
  6 PWR_B_h11              0.011713             0.962843
  7 PWR_DIS_h11           -0.087211             1.245282
  8 PWR_INDUS_h11         -0.010684            -0.489590
  9 PWR_TAX_h11           -0.025711            -0.219707
 10 RM_h11                 0.033846             0.912490
 11 SQRT_LSTAT_h11        -0.097869             0.517688
 12 SQRT_NOX_h11          -0.050488            -0.140444
 13 BIAS_h11              -0.529681            -0.043904
 14 EXP_AGE_h21           29.052194             0.000181
 15 EXP_PTRATIO_h21      -11.327585          0.000074459
 16 LOG_CRIM_h21          30.753520          0.000015969
 17 LOG_RAD_h21           27.889927             0.000178
 18 LOG_ZN_h21            -5.489417            -0.000213
 19 PWR_B_h21            -29.131695          0.000015961
 20 PWR_DIS_h21          -57.301461            -0.000147
 21 PWR_INDUS_h21         -7.109840             0.000208
 22 PWR_TAX_h21           -9.421459             0.000188
 23 RM_h21               -55.328801         -0.000018258
 24 SQRT_LSTAT_h21       -68.814608             0.000128
 25 SQRT_NOX_h21         -56.688132             0.000168
 26 BIAS_h21              32.576510          0.000028133
 27 h11_MEDV               2.661934            -0.027251
 28 h21_MEDV              -0.064748             0.000216
 29 BIAS_MEDV              4.325385          0.000070067

Value of Objective Function = 0.4827440354
```

17. A listing of the final weight values for the integrated two-hidden-unit SNC starts at line **494**.

```
                Optimization Results
                Parameter Estimates
                                                Gradient
                                               Objective
  N Parameter              Estimate             Function

  1 EXP_AGE_h11            0.048232             0.000553
  2 EXP_PTRATIO_h11       -0.079470             0.002138
  3 LOG_CRIM_h11          -0.061532             0.000312
  4 LOG_RAD_h11            0.121688             0.000619
  5 LOG_ZN_h11            -0.036177            -0.000642
  6 PWR_B_h11             -0.018916            -0.000815
  7 PWR_DIS_h11           -0.146659          0.000047840
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
  8 PWR_INDUS_h11          0.009803        0.001194
  9 PWR_TAX_h11           -0.054781        0.000428
 10 RM_h11                -0.014888       -0.002324
 11 SQRT_LSTAT_h11        -0.159659        0.001525
 12 SQRT_NOX_h11          -0.200511        0.000305
 13 BIAS_h11              -0.262718       -0.002343
 14 EXP_AGE_h21            0.638360        0.000285
 15 EXP_PTRATIO_h21       -0.374085        0.000727
 16 LOG_CRIM_h21           1.107703        0.000553
 17 LOG_RAD_h21            0.449407        0.000860
 18 LOG_ZN_h21            -0.273772       -0.001731
 19 PWR_B_h21             -0.399836       -0.000917
 20 PWR_DIS_h21           -0.926367       -0.001247
 21 PWR_INDUS_h21          0.094297        0.001778
 22 PWR_TAX_h21           -0.205174        0.000505
 23 RM_h21                -0.862313       -0.000848
 24 SQRT_LSTAT_h21        -0.949545        0.001628
 25 SQRT_NOX_h21          -1.625506        0.001214
 26 BIAS_h21               0.613682       -0.000157
 27 h11_MEDV               2.074265        0.000165
 28 h21_MEDV              -0.324932       -0.000129
 29 BIAS_MEDV              3.632104       -0.001204

Value of Objective Function = 0.3718292205
```

18. Next, the two-hidden-unit network is frozen (except for **BIAS_MEDV**) and a third hidden unit is added. Because a new hidden unit has been added, preliminary training must be performed (line **536**).

```
Three Hidden Unit Network

The NEURAL Procedure

Preliminary       Starting         Objective         Number
 Training          Random           Function           of        Terminating
    Run             Seed             Value         Iterations     Criteria

        1           781996371    0.367713435286          10
        2          1166826662    0.355769464813          10
        3          1690307879    0.371645998654          10
        4          1820157317    0.361675413488          10
        5          1989002139    0.370059315206          10
```

19. The weights at the end of the second preliminary training run become the starting values for training the partially frozen three-hidden-unit network. These parameter values appear at line **557**.

```
                 Optimization Start
                Parameter Estimates
                                          Gradient
                                         Objective
  N Parameter              Estimate        Function

  1 EXP_AGE_h31            0.037456        0.000462
  2 EXP_PTRATIO_h31       -7.944794        0.000831
  3 LOG_CRIM_h31          -1.663123        0.000590
  4 LOG_RAD_h31           -0.341921        0.001137
  5 LOG_ZN_h31             9.821306       -0.000146
  6 PWR_B_h31             -0.534231       -0.000717
  7 PWR_DIS_h31            3.251523       -0.000486
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
  8 PWR_INDUS_h31          9.177530        0.001529
  9 PWR_TAX_h31            0.517942        0.001118
 10 RM_h31                -5.356924       -0.000556
 11 SQRT_LSTAT_h31       -13.097508        0.001223
 12 SQRT_NOX_h31          19.060681        0.001513
 13 BIAS_h31              15.402516       -0.001348
 14 h31_MEDV              -0.035948        0.072416
 15 BIAS_MEDV              3.657651        0.009119

Value of Objective Function = 0.3557694648
```

20. The final weights of the partially frozen three-hidden-unit SNC network are listed at line **669**.

```
                Optimization Results
                Parameter Estimates
                                              Gradient
                                             Objective
  N Parameter              Estimate          Function

  1 EXP_AGE_h31            0.583995          0.000166
  2 EXP_PTRATIO_h31       -8.491350          0.000131
  3 LOG_CRIM_h31          -1.682741          0.000244
  4 LOG_RAD_h31           -0.352559          0.000192
  5 LOG_ZN_h31             9.365983      -0.000078309
  6 PWR_B_h31             -0.382254         -0.000314
  7 PWR_DIS_h31            2.864148         -0.000184
  8 PWR_INDUS_h31          8.792933          0.000292
  9 PWR_TAX_h31            1.121416       0.000022263
 10 RM_h31                 -5.511893       0.000057792
 11 SQRT_LSTAT_h31       -13.860402          0.000236
 12 SQRT_NOX_h31          18.631613       0.000068016
 13 BIAS_h31              15.866865       0.000013511
 14 h31_MEDV              -0.040064          0.014632
 15 BIAS_MEDV              3.660549         -0.013261

Value of Objective Function = 0.3542399056
```

21. Go to line **701** to see the entire set of starting weights after thawing. A partial listing is given below.

```
                 Optimization Start
                Parameter Estimates
                                              Gradient
                                             Objective
  N Parameter              Estimate          Function

  1 EXP_AGE_h11            0.048232          0.068118
  2 EXP_PTRATIO_h11       -0.079470          0.548879
  3 LOG_CRIM_h11          -0.061532         -0.115597
  4 LOG_RAD_h11            0.121688         -0.387108
      ...
 40 h11_MEDV               2.074265         -0.028901
 41 h21_MEDV              -0.324932         -0.004714
 42 h31_MEDV              -0.040064          0.014632
 43 BIAS_MEDV              3.660549         -0.013261

Value of Objective Function = 0.3542399056
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

22. The post training weights for the fully integrated three-hidden-unit SNC network appear at line **862**.

```
                Optimization Results
                Parameter Estimates
                                              Gradient
                                             Objective
  N Parameter              Estimate          Function

  1 EXP_AGE_h11            0.041772         -0.001541
  2 EXP_PTRATIO_h11       -0.077353         -0.001242
  3 LOG_CRIM_h11          -0.051462         -0.000857
  4 LOG_RAD_h11            0.118994         -0.001647
  5 LOG_ZN_h11            -0.025880          0.002077
  6 PWR_B_h11             -0.010565         -0.000651
  7 PWR_DIS_h11           -0.122458         -0.001470
  8 PWR_INDUS_h11          0.014691         -0.000494
  9 PWR_TAX_h11           -0.052696         -0.001079
 10 RM_h11                -0.015681         -0.000454
 11 SQRT_LSTAT_h11        -0.140945         -0.002379
 12 SQRT_NOX_h11          -0.167340          0.004557
 13 BIAS_h11              -0.235599         -0.003424
 14 EXP_AGE_h21            0.680265     -0.000051340
 15 EXP_PTRATIO_h21       -0.383308         -0.000261
 16 LOG_CRIM_h21           1.167814      0.000025670
 17 LOG_RAD_h21            0.502637         -0.000269
 18 LOG_ZN_h21            -0.306809          0.000177
 19 PWR_B_h21             -0.359019         -0.000179
 20 PWR_DIS_h21           -0.907297         -0.000160
 21 PWR_INDUS_h21          0.052042      0.000032115
 22 PWR_TAX_h21           -0.253967     -0.000000812
 23 RM_h21                -0.849071         -0.000133
 24 SQRT_LSTAT_h21        -0.863199         -0.000158
 25 SQRT_NOX_h21          -1.707940          0.000649
 26 BIAS_h21               0.616255          0.000186
 27 EXP_AGE_h31            5.440263     -0.000052614
 28 EXP_PTRATIO_h31      -54.271913     -0.000060152
 29 LOG_CRIM_h31         -10.193822     -0.000022447
 30 LOG_RAD_h31           -3.573653     -0.000061285
 31 LOG_ZN_h31            57.522787      0.000086785
 32 PWR_B_h31             -0.750248      0.000034449
 33 PWR_DIS_h31           17.821939      0.000092248
 34 PWR_INDUS_h31         51.650139     -0.000064379
 35 PWR_TAX_h31           14.605506     -0.000012381
 36 RM_h31               -35.876206      0.000053171
 37 SQRT_LSTAT_h31       -90.888223     -0.000022965
 38 SQRT_NOX_h31         120.231436     -0.000078082
 39 BIAS_h31              99.983171      0.000019779
 40 h11_MEDV               2.382169         -0.000605
 41 h21_MEDV              -0.327452          0.000487
 42 h31_MEDV              -0.073885          0.000405
 43 BIAS_MEDV              3.703265      0.000085702

Value of Objective Function = 0.3369762395
```

Of course, this construction process could be continued further, but perhaps it is best to assess whether further iterations are warranted by examining the goodness-of-fit information.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

23. Scroll to line **918** to see the SBC statistics for the three networks constructed so far.

```
Sequential Network Construction
One Hidden Unit Network

  Train:
Schwarz's
 Bayesian
Criterion

 380.271



Sequential Network Construction
Two Hidden Unit Network

  Train:
Schwarz's
 Bayesian
Criterion

 368.715



Sequential Network Construction
Three Hidden Unit Network

  Train:
Schwarz's
 Bayesian
Criterion

 438.251
```

Notice that, although the Schwarz Bayesian Criterion (SBC) value initially declined when the second hidden unit was added, it rose sharply when the third hidden unit was added. This suggests that the two-hidden-unit network is optimal. Of course, this rise in SBC value could be an aberration. Perhaps adding another hidden unit would bring the SBC down to its lowest value yet.

In the next exercise you test the contention that adding a fourth hidden unit to the SNC network would bring the Schwarz Bayesian Criterion value down.

24. Close the Results window and the SAS Code node.

25. Close the “snc” diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.03 Poll

The sequential network construction methodology creates a neural network consisting of multiple hidden layers.

- ❍ True
- ❍ False

28

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Exercises

2. Sequential Network Construction

In the previous demonstration, it was suggested that adding a fourth hidden unit might reverse the upward trend in the Schwarz Bayesian Criterion value. Here you explore this question. If the SBC value continues to rise when the fourth hidden is added, so too does your confidence that the two-hidden-unit MLP is optimal. Otherwise, the merits of a more complex network should be assessed.

a. Right-click **Diagrams** in the project panel, and select **Create Diagram** from the pop-up menu.

b. Enter **exercise 4.2** in the **Diagram Name** field, and click **OK**.

c. Open the **snc** demonstration (c4s3d1), and copy and paste the entire flow into the exercise 4.2 diagram. Your copied flow diagram should look similar to the one shown below.

d. Close the **snc** diagram.

e. Activate the SAS Code node properties panel, and click the ellipsis (**…**) next to **Code Editor**.

f. Using the previous hidden unit additions as a guide, add a fourth hidden unit to the PROC NEURAL sequential network construction code.

g. Also add another PRINT procedure to the bottom of the program in order to display the Schwarz Bayesian Criterion (SBC) value that was generated by the addition of the fourth hidden unit.

h. Answer the following questions:

- Did the Schwarz Bayesian value fall after the fourth hidden unit was added?
- What does this suggest with respect to the optimal number of hidden units required to adequately model the **HOUSING** data?

i. Close the SAS Code node's results.

j. Close the "exercise 4.2" diagram.

End of Exercises

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.4 Generalized Additive Neural Networks (Self-Study)

Objectives

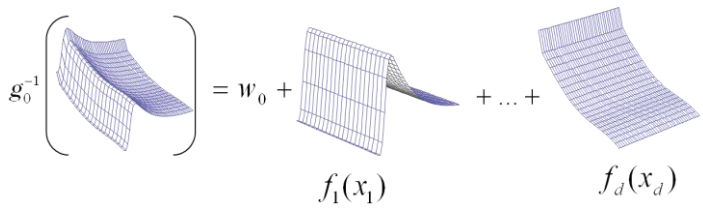
- Define a generalized additive neural network (GANN).
- Illustrate the use of the GANN paradigm, and compare its performance against other methods.

33

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Generalized Additive Models

$$g_0^{-1}(E(y)) = w_0 + \sum_{i=1}^{d} f_i(x_i)$$



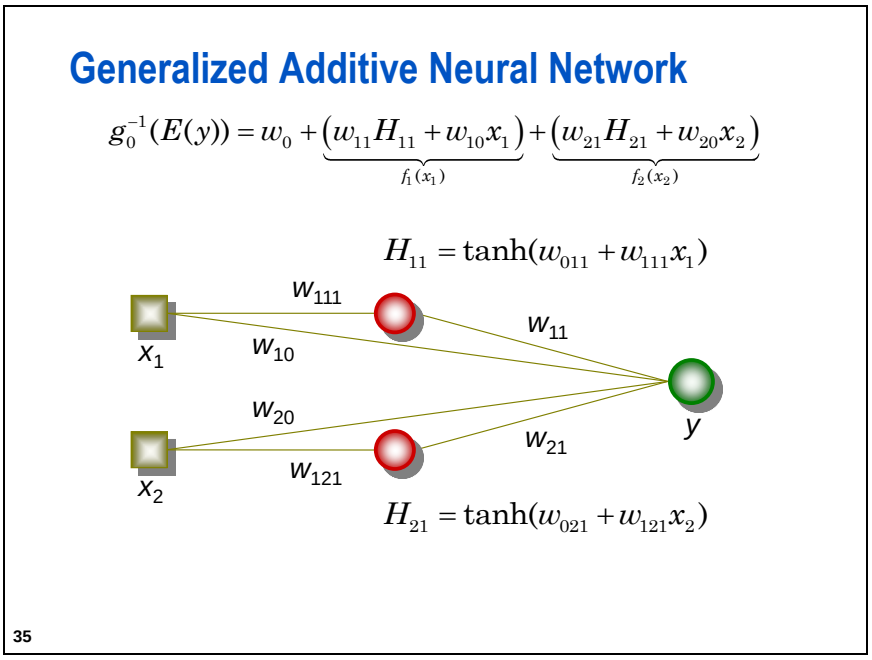
34

In a *generalized additive model* (Hastie and Tibshirani 1986, 1990), the target is expressed as the sum of individual nonlinear univariate functions, $f_i(x_i)$. Each of the univariate functions can be interpreted as the effect of the corresponding input, holding the other inputs constant.

$$g_0^{-1}(E(y)) = w_0 + \sum_{i=1}^{d} f_i(x_i)$$

A generalized additive model is actually a restrictive class of models, as it does not allow for interactions between the input variables (Bishop 1995). In other words, generalized additive models are ***not*** universal approximators. Because the individual univariate functions can be easily plotted, however, they facilitate interpretation. Moreover, given that each of the individual additive terms is estimated using a univariate smoother, the “curse of dimensionality” is avoided.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

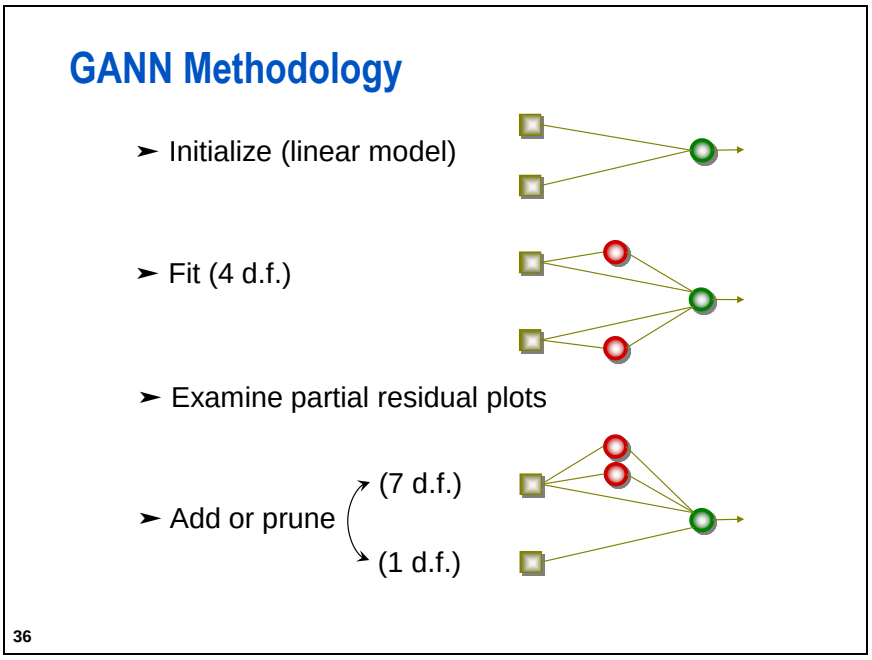


The *generalized additive neural network* (GANN) is a compromise between a generalized linear model and a fully connected neural network. As such, it combines the advantages of both.

Neural networks form the individual linear, or nonlinear, univariate functions of the associated input. Any number of hidden units, indeed any architecture, can be used to implement the input unit transformations. Hidden units can be added or removed to, respectively, either increase or decrease the nonlinearity of the input transformation. Including a skip layer makes the generalized linear model a special case.

Because there is no interaction between inputs, the magnitude of the weight connecting an input's mini-network to the output unit can now validly be interpreted as the input's importance. As was previously noted, this increase in interpretability comes at a price; ***a GANN is not a universal approximator***. It would not be expected to perform well, for example, in the presence of strong input interactions.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Methodology

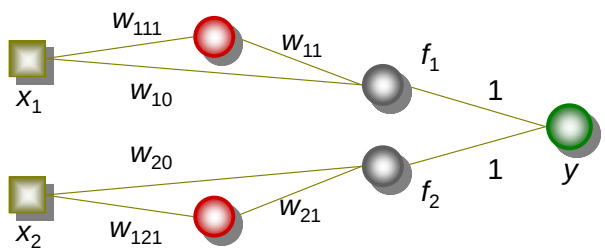
1. Create a linear perceptron for each input. Fit these linear models in order to get initial estimates for the skip layer weight and the output bias.
2. Add a hidden unit to each input, creating a 4-degree-of-freedom skip-layer network.
3. Train the network.
4. Examine each of the univariate functions overlaid on their partial residuals (discussed momentarily).
5. Prune hidden units from each input's network with apparently linear effects, and add a hidden unit when the nonlinear trend seems to under fit the data. If any change is made, go to step 3.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Consolidation Layer

$$g_0^{-1}(E(y)) = w_0 + \underbrace{\left(w_{11}H_{11} + w_{10}x_1\right)}_{f_1(x_1)} + \underbrace{\left(w_{21}H_{21} + w_{20}x_2\right)}_{f_2(x_2)}$$

$$H_{11} = \tanh(w_{011} + w_{111}x_1)$$



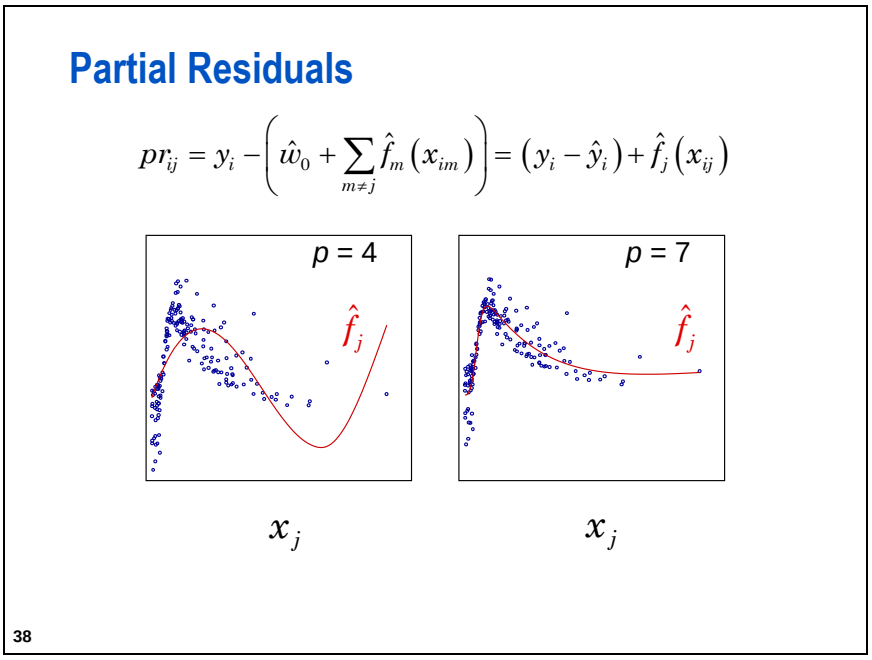
$$H_{21} = \tanh(w_{021} + w_{121}x_2)$$

37

To facilitate the generation of the partial residual plots used to guide the growing or pruning of the networks associated with each input, a useful modification of the standard GANN architecture is the addition of a linear layer to consolidate the output from the hidden layer for each input.

Because the identity activation function is used, and the consolidation layer-to-output connections are all frozen at one, this layer does not transform the input that it receives in any way. The purpose of the consolidation layer is strictly accounting (i.e., to record the activation of each input's mini-network). The output of the consolidation layer is simply the activation of the individual univariate functions.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Partial residual plots (Ezekiel 1924) show the effect of the input variables, adjusted for the effects of the other inputs. They are a more effective visual diagnostic than simple plots of the target versus an input. Partial residuals are calculated as the difference between the target variable and the part of the additive model that does not include the input under investigation. If the identity link function is used, the partial residual for input x_{ij} is calculated as follows:

$$pr_{ij} = y_i - \left(\hat{w}_0 + \sum_{m \neq j} \hat{f}_m(x_{im})\right) = (y_i - \hat{y}_i) + \hat{f}_j(x_{ij})$$

More generally, if g_0^{-1} is nonlinear, you can either link-transform the target,

$$pr_{ij} = g_0^{-1}(y_i) - \left(\hat{w}_0 + \sum_{m \neq j} \hat{f}_m(x_{im})\right) = g_0^{-1}(y_i) - g_0^{-1}(\hat{y}_i) + \hat{f}_j(x_{ij})$$

or you can use first-order approximation (Cai and Tsai 1999).

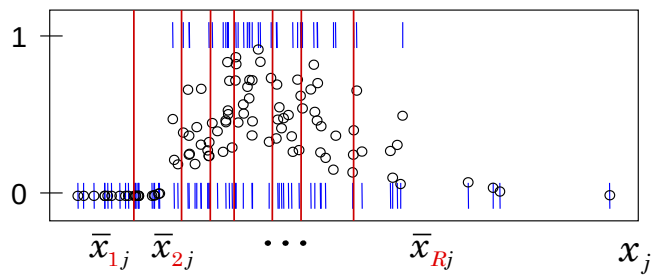
$$pr_{ij} = \frac{\partial g_0^{-1}(\hat{y}_i)}{\partial y_i}(y_i - \hat{y}_i) + \hat{f}_j(x_{ij})$$

Berk and Booth (1995) show that the partial residuals based on a general additive fit are more reliable than those based on a linear fit. However, it must be remembered that model selection based on partial residual plots is subjective. Formal measures of fit should be incorporated into the selection process.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Empirical Partial Residuals

$$pr_{rj} = \left[\ln\left(\frac{m_{1rj}+1}{m_{rj}-m_{1rj}+1} \right) - \ln\left(\frac{\bar{\hat{y}}_{rj}}{1-\bar{\hat{y}}_{rj}} \right) \right] + \bar{\hat{f}}_{rj}(x_j)$$



39

When the target variable is discrete, partial residuals can only be approximated. For example, when the target is binary, a logit-link (logistic activation) function is appropriate. However, the logit-link function is infinite at zero and 1, so transformation is infeasible. And a first-order approximation produces wild outliers when the predicted probabilities are close to zero or 1 (Landwehr 1984).

Empirical partial residuals are an alternative. The empirical partial residual for the jth input is calculated by grouping the input variable into *R* equal sized bins and then transforming the sample proportions. More specifically, the empirical partial residual of the rth bin is given by the following:

$$pr_{rj} = \left[\ln\left(\frac{m_{1rj}+1}{m_{rj}-m_{1rj}+1} \right) - \ln\left(\frac{\bar{\hat{y}}_{rj}}{(1-\bar{\hat{y}}_{rj})} \right) \right] + \bar{\hat{f}}_{rj}(x_j)$$

where

m_{1rj} = the total number of events in the bin

m_{rj} = the total number of cases in the bin

$\bar{\hat{y}}_{rj}$ = the mean predicted probability of the bin

$\bar{\hat{f}}_{rj}(x_j)$ = the mean component of the bin.

The generation of empirical partial residual plots to guide network pruning/growth decisions when the target is discrete is illustrated in Appendix B.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Generalized Additive Neural Network

c4s4d1

This demonstration illustrates the generalized additive neural network (GANN) architecture. The Boston **HOUSING** data (Harrison and Rubinfeld 1978) is again used.

Starting with a 4-degree-of-freedom model is common practice in GAM estimation (Potts 1999), so the initial GANN structure will have a single hidden unit and a skip layer associated with each input. It also has a linear consolidation layer between the hidden and output layers. This layer facilitates generation of the partial residual plots that will be used to guide the pruning and growing of each input's mini-network.

1. Right-click **Diagrams** in the project panel, and select **Create Diagram** from the pop-up menu.
2. Enter the name **gann** in the **Diagram Name** field and click **OK**.
3. Because the **HOUSING** data has been defined, drag and drop the data source onto the diagram.
4. As before, because many of the inputs are skewed, add a **Transform Variables** node (Modify tab) to the diagram, and connect it to the **HOUSING** data source.
5. Change the Default Method property for **Interval Inputs** to **Maximum Normal** (maximize normality).
6. Drag and drop a **SAS Code** node (Utilities tab) onto the diagram, and connect it to the **Transform Variables** node. Your diagram should now look like the one shown below.

7. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.
8. In the Training Code editor, right-click in the white space and select **Open**.
9. Select the program **c4s4d1a.sas** (in d:\workshop\winsas\dmnn41). The program is described below.

   First, the global system and graphics options are set.

```
options nodate nonumber;
goptions reset=all device=actximg;
```

   Then the macro function GANN is defined. It has a single argument, **dsn**, the data set to be analyzed.

```
%macro gann(dsn=);
```

   First, the interval input order is listed in the log file. This is used to identify the appropriate mini-network to modify (linearize, augment, or prune), as a result of the partial residual plot feedback.

```
    %put %EM_INTERVAL_INPUT;
```

   Only interval inputs need be considered, because discrete (nominal and ordinal) inputs cannot be assessed using the partial residual plot methodology.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Then the number of interval inputs is determined.

```
data _null_;
   call symputx('ninputs',length("%cmpres(%EM_INTERVAL_INPUT)")-
        length(compress("%cmpres(%EM_INTERVAL_INPUT)"))+1);
run;
```

Next the input data set is converted into a data mining database and cataloged.

```
proc dmdb data=&dsn dmdbcat=cat;
   var %EM_INTERVAL_INPUT %EM_TARGET ;
run;
```

Then the NEURAL procedure is invoked, passing it the newly cataloged input data set. As before, the RANSCALE= option sets the variance of the normal distribution used to generate the initial random values. Setting it to 0.1 means that training will slowly move away from a linear model.

```
title1 h=1.5 f=swissb 'Generalized Additive Neural Network';
proc neural data=&dsn ranscale=0.1 dmdbcat=cat;
```

The target layer is then defined. Because the original Harrison and Rubinfeld (1978) study assumed that the target was Poisson distributed, exponential activation and Poisson deviance are specified. The output (target) layer is named **out**.

```
   target %EM_TARGET/level=int act=exp error=poisson id=out;
```

A %DO loop is used to set up the input, skip, and consolidation layers. First, a %SCAN function is used to "peel off" the ith input from the interval input list. The INPUT statement then creates an input layer consisting of the input. The input layer is named $\mathbf{x_i}$, where **i** ranges from 1 to the ninputs.

Next the consolidation layer is created. Setting the activation function to identity (IDENTITY) and the combination function to linear (LIN), and removing the hidden unit's bias (NOBIAS) ensures that values that enter the consolidation layer will pass through the layer unchanged. The consolidation layer associated with the ith input is named $\mathbf{f_i}$.

Finally, the input layer is connected to the consolidation layer, creating the input's skip-layer, and the consolidation layer is connected to the output layer.

```
   %do i=1 %to &ninputs;
      input %scan(%EM_INTERVAL_INPUT,&i)/level=int id=x&i;
      hidden 1/act=identity comb=lin nobias id=f&i;
      connect x&i f&i out;
   %end;
```

The specified architecture is then initialized using the default network initialization values. This must be done before freezing of specific connections (which follows) can be implemented.

```
   initial;
```

As previously outlined, the consolidation units must be frozen at the value 1 to ensure that they, in no way, alter the incoming activation from the associated input's mini-network.

```
   %do i=1 %to &ninputs;
      freeze f&i->out/value=1;
   %end;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The NEURAL procedure is then asked to train the current architecture. Because each input's network currently consists only of direct connections, this initializes each input to reasonable starting values.

```
   train;
```

Now construction of the mini-networks can begin. Although each mini-network is exactly the same (i.e., a single hidden unit and a direct connection), because they will be individually pruned/grown after examination of the partial residual plots, each is separately defined.

```
   hidden 1 / id=h1;
   connect x1 h1 f1;

   hidden 1 / id=h2;
   connect x2 h2 f2;

   hidden 1 / id=h3;
   connect x3 h3 f3;

   hidden 1 / id=h4;
   connect x4 h4 f4;

   hidden 1 / id=h5;
   connect x5 h5 f5;

   hidden 1/ id=h6;
   connect x6 h6 f6;

   hidden 1 / id=h7;
   connect x7 h7 f7;

   hidden 1 / id=h8;
   connect x8 h8 f8;

   hidden 1/ id=h9;
   connect x9 h9 f9;

   hidden 1 / id=h10;
   connect x10 h10 f10;

   hidden 1 / id=h11;
   connect x11 h11 f11;

   hidden 1 / id=h12;
   connect x12 h12 f12;

   prelim 5 maxiter=10;
   train outfit=fit;
   score data=&dsn out=&EM_EXPORT_TRAIN;
run;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The mini-networks undergo five preliminary training runs, each lasting 10 iterations, before training begins. Various fit statistics are computed and written out (**outfit**=) to the data set **fit**. Finally, the input data set is scored, and the predicted values are output to **&EM_EXPORT_TRAIN**.

Unfortunately, because of the addition of the consolidation layer units, any metrics that consider the number of parameters in the model (such as the SBC and AIC) are computed incorrectly. The frozen parameters (one for each input) are counted in the AIC and SBC penalty terms, making their values too high. Consequently, these statistics need to be corrected.

```
title2 'Corrected Fit Statistics';
proc sql;
   select _dfm_-&ninputs as Parameters,
          _err_+(calculated Parameters)*log(_dft_) as SBC
   from fit
   where _name_='OVERALL';
quit;
```

To correct the AIC value use **_err_ + (calculated Parameters)*2 as AIC**.

Next the partial residuals to be plotted are calculated. Given that the log-link (exponential activation) function was used, the partial residuals are calculated on the log scale.

```
data &EM_EXPORT_TRAIN;
   set &EM_EXPORT_TRAIN;
   lres=log(%EM_TARGET)-log(p_%EM_TARGET);
   %do i=1 %to &ninputs;
      parres&i=lres+f&i.1;
   %end;
run;
```

These partial residuals are then plotted, completing the GANN macro function definition.

```
title2 'Partial Residual Plot';
proc gplot data=&EM_EXPORT_TRAIN ;
   symbol1 c=b v=circle i=none;
   symbol2 c=bl v=none i=splines;
   %do i=1 %to &ninputs;
       plot parres&i*%scan(%EM_INTERVAL_INPUT,&i)=1
          f&i.1*%scan(%EM_INTERVAL_INPUT,&i)=2 / overlay frame;
   %end;
run;
quit;
```

The GANN macro function is then invoked, passing it the data set to be modeled (in this case the input data set pointed to by the **%EM_INPUT_DATA** macro variable).

```
%gann(dsn=&EM_IMPORT_DATA)
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

10. Run the diagram by selecting the running man icon. Confirm the run. When training is complete, click **Results**. The initial output in the listing (line **25**) gives the initial weight values for the GANN.

```
The NEURAL Procedure

                 Optimization Start
                Parameter Estimates
                                                Gradient
                                               Objective
  N Parameter              Estimate             Function

  1 EXP_AGE_f11           -0.004298            11.415395
  2 EXP_PTRATIO_f21       -0.009999             9.474523
  3 LOG_CRIM_f31          -0.024349             8.879359
  4 LOG_RAD_f41           -0.022226             9.785746
  5 LOG_ZN_f51             0.007353           -10.059796
  6 PWR_B_f61              0.049937            -5.667616
  7 PWR_DIS_f71           -0.152119           -11.677829
  8 PWR_INDUS_f81          0.079180            14.482055
  9 PWR_TAX_f91            0.057221            11.979781
 10 RM_f101                0.017571           -11.162997
 11 SQRT_LSTAT_f111       -0.144361            13.309119
 12 SQRT_NOX_f121          0.044887            13.287543
 13 BIAS_MEDV              3.114972             0.954327

Value of Objective Function = 3.6708713511
```

Only the input to consolidation layer hidden unit (**f11** to **f121**) connections are specified. In other words, the GANN is currently only a generalized linear model.

11. Right-click anywhere in the Output window. To see the final parameter values for the generalized linear model, select **Go to line** and enter **116**.

```
The NEURAL Procedure

                Optimization Results
                Parameter Estimates
                                                Gradient
                                               Objective
  N Parameter              Estimate             Function

  1 EXP_AGE_f11           -0.001543          0.000001000
  2 EXP_PTRATIO_f21       -0.069062          0.000000794
  3 LOG_CRIM_f31          -0.148530          0.000002535
  4 LOG_RAD_f41            0.112721          0.000001242
  5 LOG_ZN_f51            -0.010178         -0.000000373
  6 PWR_B_f61              0.020677         -0.000001516
  7 PWR_DIS_f71           -0.177190         -0.000001416
  8 PWR_INDUS_f81         -0.025551          0.000000806
  9 PWR_TAX_f91           -0.059754          0.000001043
 10 RM_f101                0.074291         -0.000000854
 11 SQRT_LSTAT_f111       -0.220143         -0.000000170
 12 SQRT_NOX_f121         -0.107391          0.000000467
 13 BIAS_MEDV              3.048774         -6.473042E-8

Value of Objective Function = 0.5828866204
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

12. Scroll to line **145** to see the preliminary training results for the 4-degree-of-freedom mini-networks.

```
The NEURAL Procedure

Preliminary       Starting         Objective         Number
 Training          Random           Function           of        Terminating
    Run             Seed             Value         Iterations     Criteria

        1            42129862    0.426964302677           10
        2           679865644    0.469054696714           10
        3            43195411    0.452003383216           10
        4          1356277451    0.537075429048           10
        5          1116558348    0.437713623448           10
```

Because the first preliminary training run generated the smallest objective function value, the weights at the end of the first run will be used as the starting weights for training.

13. A partial listing of these parameters, which start at line **161** in the output, is reproduced below.

```
The NEURAL Procedure

                 Optimization Start
                Parameter Estimates
                                          Gradient
                                         Objective
  N Parameter              Estimate       Function

  1 EXP_AGE_h11           -0.601062       0.037640
  2 BIAS_h11              -0.398101      -0.075120
  3 EXP_AGE_f11            0.163836       0.089175
  4 h11_f11                0.369444       0.038896
  5 EXP_PTRATIO_h21        0.072858       0.000297
  6 BIAS_h21               0.014810      -0.000910
  7 EXP_PTRATIO_f21       -0.064611       0.089255
  8 h21_f21                0.003409       0.002473
  9 LOG_CRIM_h31           0.898040      -0.007225
 10 BIAS_h31              -0.854308       0.023336
 11 LOG_CRIM_f31          -0.051075       0.103309
 12 h31_f31               -0.259613       0.218921
 13 LOG_RAD_h41           -0.148920       0.000140
 14 BIAS_h41               0.074653      -0.000386
 15 LOG_RAD_f41            0.118945       0.097077
 ...
 40 h101_f101              0.403291       0.109161
 41 SQRT_LSTAT_h111        0.800558      -0.005399
 42 BIAS_h111             -1.086639       0.015062
 43 SQRT_LSTAT_f111       -0.107510       0.162222
 44 h111_f111             -0.223219       0.228280
 45 SQRT_NOX_h121          0.819980       0.039505
 46 BIAS_h121              1.018542      -0.050600
 47 SQRT_NOX_f121         -0.201207       0.092032
 48 h121_f121              0.318202      -0.129376
 49 BIAS_MEDV              2.872859      -0.269495

Value of Objective Function = 0.4269643027
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

14. The final parameter values for the 4-degree-of-freedom GANN start at line **343**. A partial listing of these weights is reproduced below.

```
The NEURAL Procedure

                Optimization Results
                Parameter Estimates
                                                Gradient
                                               Objective
  N Parameter              Estimate            Function

  1 EXP_AGE_h11           -0.229000            0.001145
  2 BIAS_h11              -0.164440           -0.000380
  3 EXP_AGE_f11            0.975047            0.000264
  4 h11_f11                4.495730        -0.000059123
  5 EXP_PTRATIO_h21       19.284957           -0.000124
  6 BIAS_h21               2.144745            0.000193
  7 EXP_PTRATIO_f21       -0.094131         0.000038999
  8 h21_f21                0.034206           -0.000400
  9 LOG_CRIM_h31           8.667920        2.4895789E-8
 10 BIAS_h31              -4.023939        -0.000000582
 11 LOG_CRIM_f31          -0.101380         0.000082212
 12 h31_f31               -0.105566            0.000109
 13 LOG_RAD_h41            1.578246         0.000001073
 14 BIAS_h41              -7.732346         0.000000647
 15 LOG_RAD_f41            0.128809            0.000135

   ...                       ...                 ...

  40 h101_f101              0.364133         0.000049889
 41 SQRT_LSTAT_h111        3.404183        -0.000000461
 42 BIAS_h111             -3.658092        -0.000000119
 43 SQRT_LSTAT_f111       -0.136148         0.000028640
 44 h111_f111             -0.111163         0.000057704
 45 SQRT_NOX_h121          1.299908         0.000043734
 46 BIAS_h121              1.626018        -0.000031075
 47 SQRT_NOX_f121         -0.187179            0.000269
 48 h121_f121              0.229467         0.000068011
 49 BIAS_MEDV              3.447087        -0.000083231

Value of Objective Function = 0.4046623109
```

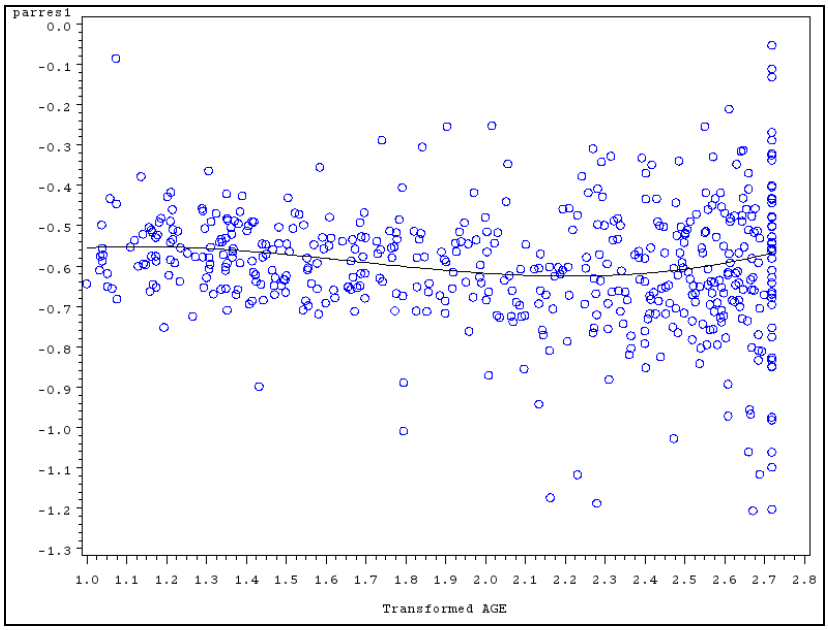
15. Finally, scroll to line **407** to see the corrected fit statistics.

```
Corrected Fit Statistics

Parameters       SBC
--------------------
        49  509.8594
```

After factoring out the connections to the consolidation layer units, there are 49 parameters (weights) in the 4-degree-of-freedom GANN. The corrected Schwarz's Bayesian Criterion (SBC) is 509.8594.

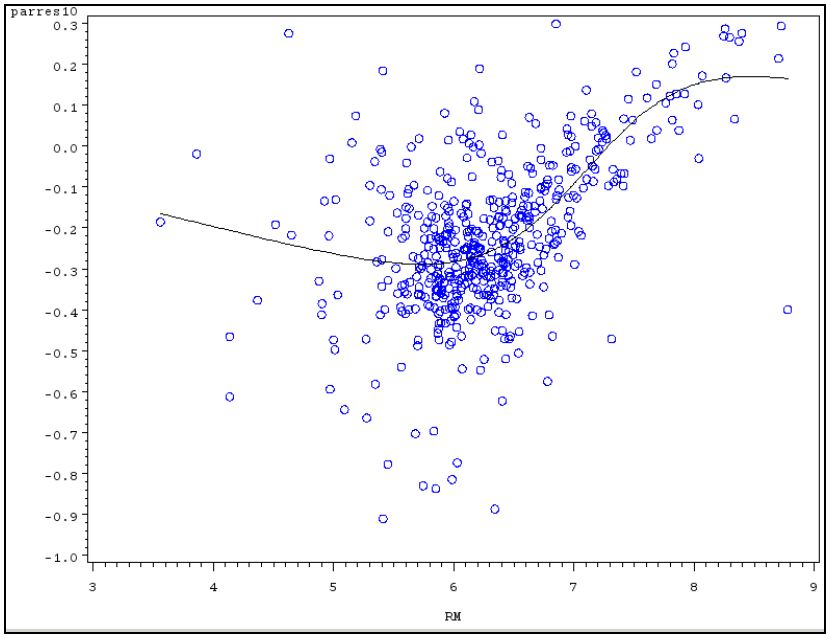
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

16. At the top of the window, select **View** ⇨ **SAS Results** ⇨ **Train Graphs**. Maximize the plot window. The first graph displays the partial residual plot for the transformed **AGE** variable.



The transformed **AGE** variable appears to be of little predictive value. The fit is approximately equal to the null model (i.e., a zero slope). This suggests that **AGE** can be deleted from the input list.

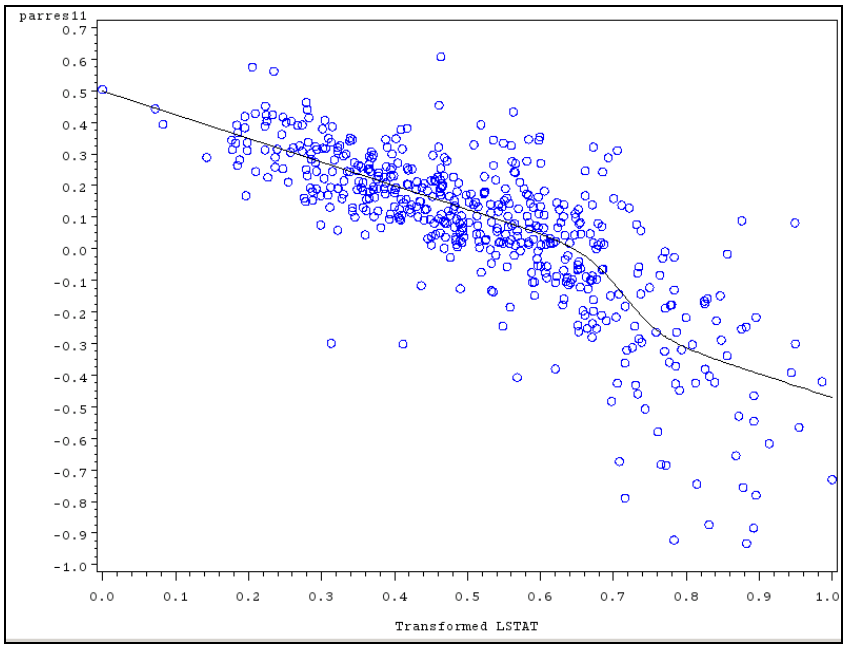
17. Click the single arrow (second button from the right) to see the partial residual plot for **RM**.



The 4-degree-of-freedom mini-network seems to generate an adequate fit to the data. No change is implied for the mini-network associated with the **RM** variable.

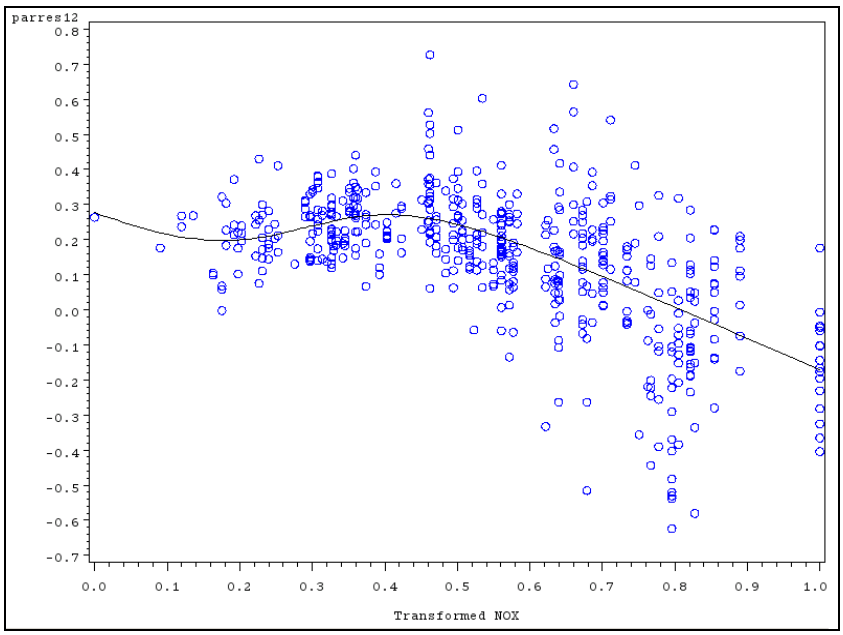
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

18. Again, click the single arrow (second button from the right) to see the next plot.



The transformed **LSTAT** variable appears to be rather linear. That is, there appears to be no need for the flexibility generated by the hidden unit in its network. Linearization (removal of the hidden unit) would seem to be implied.

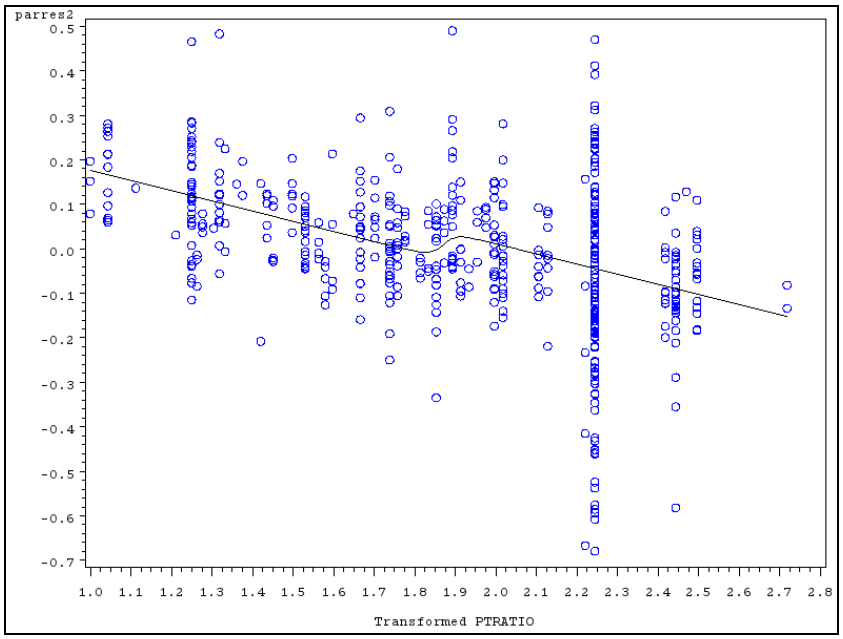
19. Click the single arrow (second button from the right) to see the plot of the transformed **NOX** variable.



In this case, the fit line appears to be able to adequately bend to fit the **NOX** partial residual pattern. The current 4-degree-of-freedom model seems adequate.

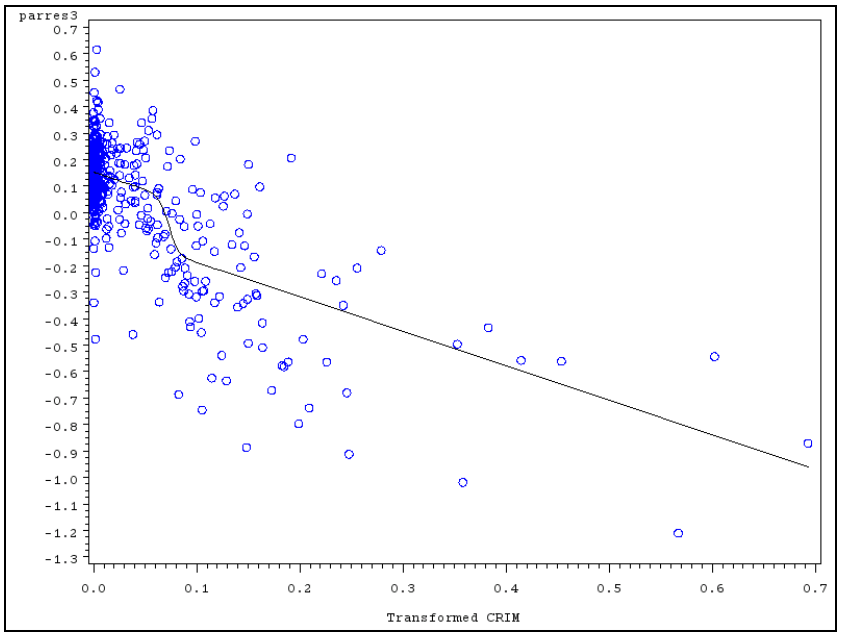
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

20. Click the second button from the right to see the plot of the transformed **PTRATIO** variable.



The sudden zigzag in the middle of the plot suggests that the mini-network associated with this input variable is too flexible. Linearization of the **PTRATIO** mini-network would seem appropriate.

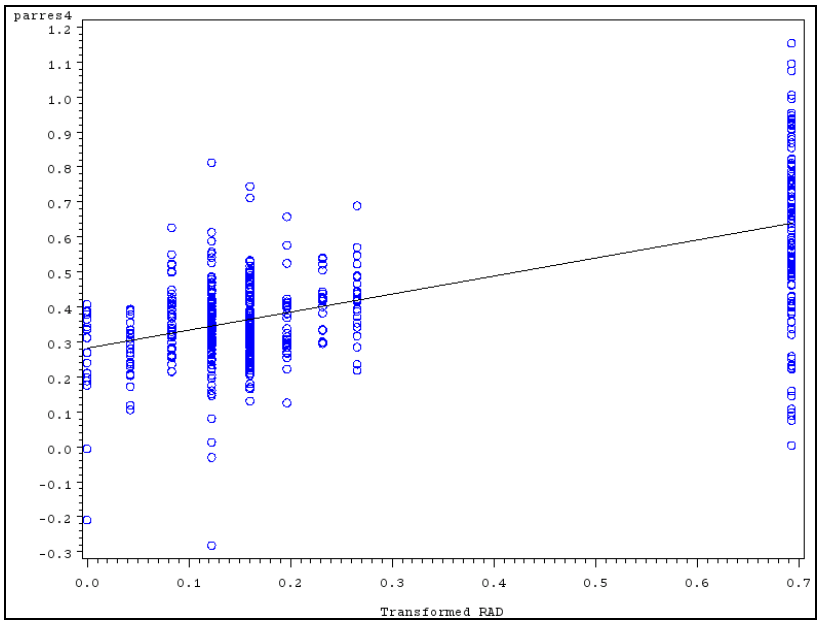
21. Click the second button from the right again to see the transformed **CRIM** variable's plot.



Because the distribution of the **CRIM** variable is still severely skewed, even after being transformed, the partial residual plot is more difficult to interpret. However, the bend at the lower values of the variable does not appear to add much to the fit. Therefore, linearization will be performed.

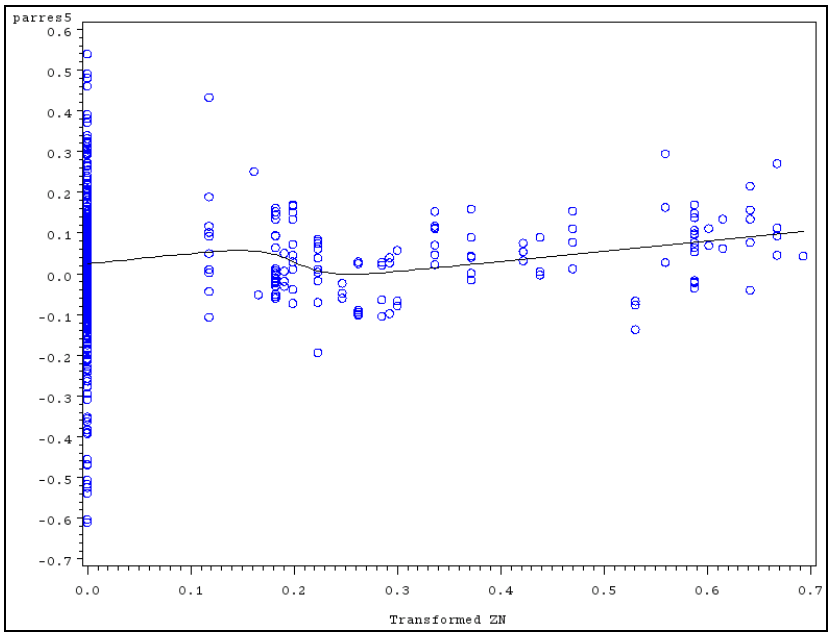
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

22. Click the second button from the right again to see the next partial residual plot.



Even with the hidden unit in its mini-network, the transformed **RAD** variable produced an extremely linear fit to the data. The hidden unit would not appear to be adding much. Linearization is implied.

23. Clicking the second button from the right once more displays the plot of the transformed **ZN** variable.

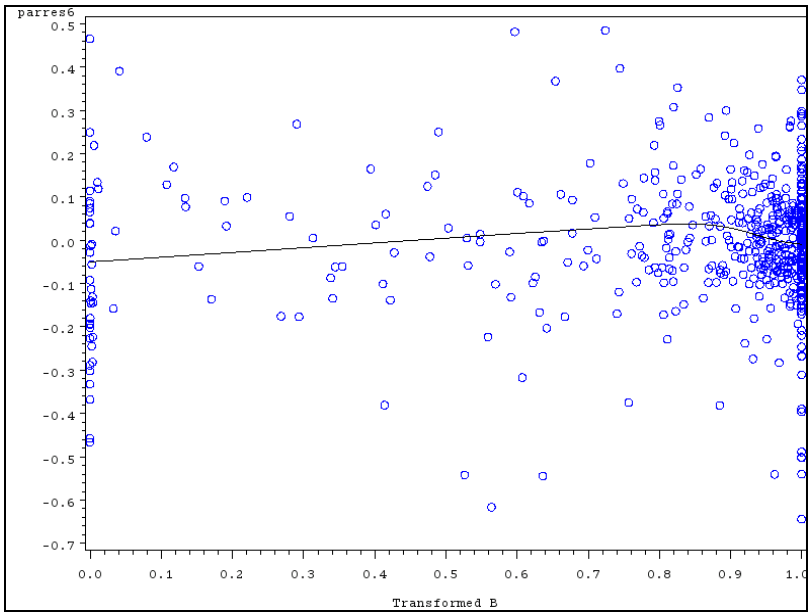


The transformed **ZN** variable, like the transformed **AGE** variable, generates a fit line with virtually no slope. Therefore, removal of the **ZN** variable from the input data set is suggested.

This is hardly surprising, because the **ZN** variable was consistently rated one of the best variables to eliminate in the previous (i.e., input selection) section.

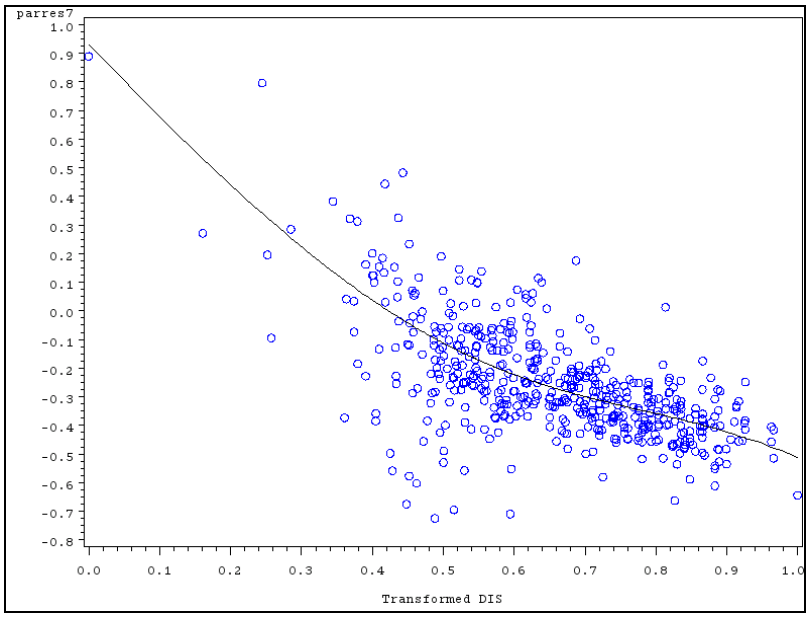
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

24. Click the right arrow button again to see the partial residual plot of the transformed **B** variable.



Like the transformed **AGE** and **ZN** variables, the fit to the transformed **B** variable approximates the null model. Therefore, elimination of **B** from the input data set is implied.

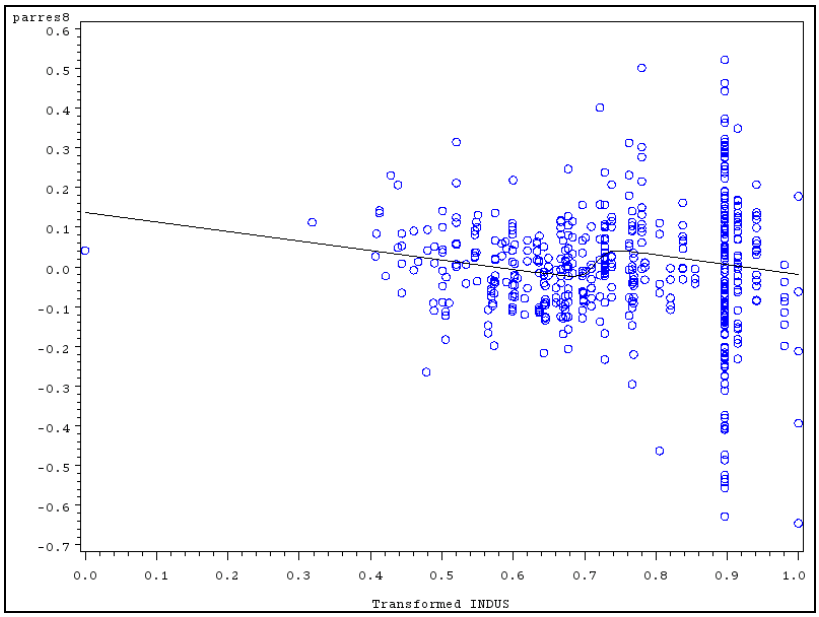
25. To see the plot of the transformed **DIS** variable, again click the second button from the right.



Although it could be argued that the partial residuals appear to be relatively linear, there does seem to be a slight curve, so this variable will be left as a 4-degree-of-freedom mini-network.

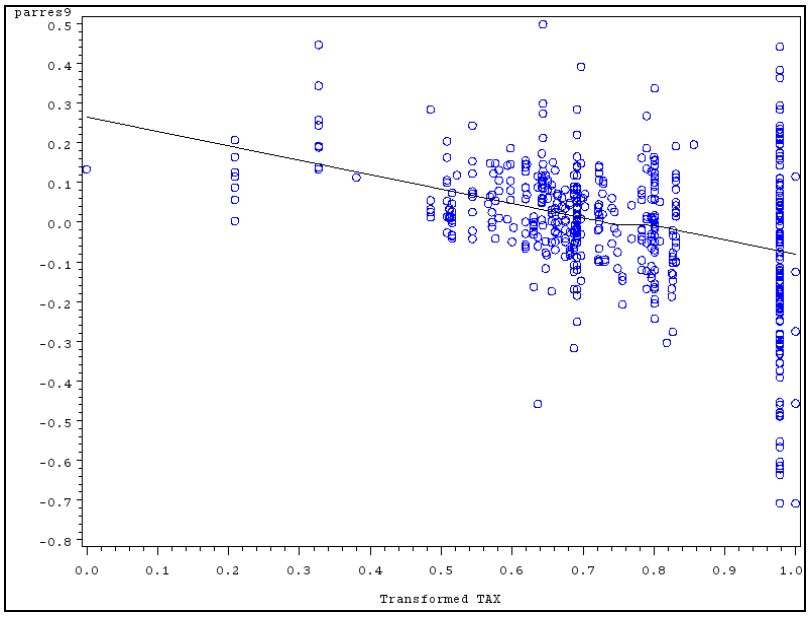
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

26. Click the right arrow again to see the partial residual plot of the transformed **INDUS** variable.



Apart from the clearly unnecessary bend in the fit, which would imply linearization at least, again the slope of the fit line appears to be nearly zero. Ockham's razor would seem to suggest that removal of the **INDUS** variable from the input data set would be the most parsimonious action.

27. Click the second button from the right one last time to reveal the final partial residual plot.



Given that there is both a linear relationship and relatively little amount of slope in the transformed **TAX** variable's partial residuals, either linearization or elimination could be performed. Both options were assessed, and removal of the **TAX** variable produced the best fit.

28. Close the SAS Code node's train graphs and Results windows.

29. Drag and drop a new **SAS Code** node onto the diagram. Connect it to the **Transform Variables** node.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

30. Right-click the new **SAS Code** node and select **Rename**. Name it **SAS Code (pruned)**. Your diagram should now look similar to the flow shown below.

31. To eliminate the transformed **AGE**, **ZN**, **B**, **INDUS** and **TAX** variables entirely, set the Use status of **EXP_AGE**, **LOG_ZN**, **PWR_B**, **PWR_INDUS**, and **PWR_TAX** to **No** (under **Variables**).

Name	Use	Report	Role	Level
CHAS	Default	No	Input	Binary
EXP_AGE	No	No	Input	Interval
EXP_PTRATIO	Default	No	Input	Interval
LOG_CRIM	Default	No	Input	Interval
LOG_RAD	Default	No	Input	Interval
LOG_ZN	No	No	Input	Interval
MEDV	Default	No	Target	Interval
PWR_B	No	No	Input	Interval
PWR_DIS	Default	No	Input	Interval
PWR_INDUS	No	No	Input	Interval
PWR_TAX	No	No	Input	Interval
RM	Default	No	Input	Interval
SQRT_LSTAT	Default	No	Input	Interval
SQRT_NOX	Default	No	Input	Interval

32. Click **OK**.

33. Click the ellipsis (**...**) next to **Code Editor** in the SAS Code (pruning) node's properties panel.

34. In the Training Code editor, right-click in the white space and select **Open**.

35. Select the program **c4s4d1b.sas** (in d:\workshop\winsas\dmnn41).

    To linearize the inputs that did not appear to need the flexibility provided by the initial 4-degree-of-freedom network, only the addition of the hidden unit need to be eliminated. To implement this change, the code corresponding to the addition of the hidden unit can simply be commented out.

```
*  hidden 1 / id=h1;  /* EXP_PTRATIO */
*  connect x1 h1 f1;

*  hidden 1 / id=h2;  /* LOG_CRIM */
*  connect x2 h2 f2;

*  hidden 1 / id=h3;  /* LOG_RAD */
*  connect x3 h3 f3;

   hidden 1 / id=h4;  /* PWR_DIS
   connect x4 h4 f4;

   hidden 1 / id=h5;  /* RM */
   connect x5 h5 f5;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
*  hidden 1/ id=h6;   /* SQRT_LSTAT */
*  connect x6 h6 f6;

   hidden 1 / id=h7;  /* SQRT_NOX */
   connect x7 h7 f7;
```

36. Run the diagram from within the SAS Code node by selecting the running man icon. Confirm the run. When training is complete, click **Results**.

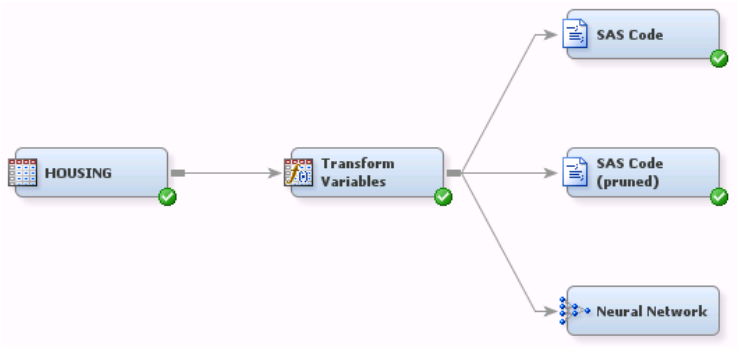
37. To see the fit achieved by the pruned GANN, scroll to line **285**.

```
Corrected Fit Statistics

Parameters       SBC
--------------------
        17  339.2897
```

The SBC value has also gone down substantially after pruning (from 509.8594 to 339.2897). In large part, this charge is due to the significant decrease in the total number of parameters in the model (from 49 to 17).

38. To compare the GANN's performance to the fit achieved by default neural network, drag and drop a **Neural Network** node (Model tab) onto the diagram.

39. Connect it to the **Transform Variables** node.



40. Activate the Neural Network node's properties panel, and click the ellipsis (**…**) next to **Network**.

41. Change the Target Layer Activation Function property to **Exponential**.

42. Change the Target Layer Error Function property to **Poisson**.

43. Click **OK** to save your changes.

44. Right-click the **Neural Network** node, and select **Run** from the pop-up menu.

45. Confirm the run, and click **Results** when training is complete.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

46. Maximize the Output window (bottom right corner), and examine the fit statistics at line **295**.

```
Fit Statistics

Target=MEDV

   Fit
Statistics    Statistics Label                     Train

 _DFT_        Total Degrees of Freedom            506.00
 _DFE_        Degrees of Freedom for Error        460.00
 _DFM_        Model Degrees of Freedom             46.00
 _NW_         Number of Estimated Weights          46.00
 _AIC_        Akaike's Information Criterion      240.17
 _SBC_        Schwarz's Bayesian Criterion        434.59
 _ASE_        Average Squared Error                 6.01
 _MAX_        Maximum Absolute Error                7.81
 _DIV_        Divisor for ASE                     506.00
 _NOBS_       Sum of Frequencies                  506.00
 _RASE_       Root Average Squared Error            2.45
 _SSE_        Sum of Squared Errors              3040.35
 _SUMW_       Sum of Case Weights Times Freq      506.00
 _FPE_        Final Prediction Error                7.21
 _MSE_        Mean Squared Error                    6.61
 _RFPE_       Root Final Prediction Error           2.69
 _RMSE_       Root Mean Squared Error               2.57
 _AVERR_      Average Error Function                0.29
 _ERR_        Error Function                      148.17
 _MISC_       Misclassification Rate                 .
 _WRONG_      Number of Wrong Classifications        .
```

Not only did the independence of the GANN's mini-networks better facilitate input selection and interpretation, the pruned GANN actually outperformed the three-hidden-unit MLP neural network with respect to the SBC value generated (i.e. 339.2897 versus 434.59).

47. Close the Neural Network node's Results window.

48. Close the "gann" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.04 Poll

A generalized additive neural network is a nonlinear model.

- ❍ True
- ❍ False

41

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.5 Stochastic Gradient Descent

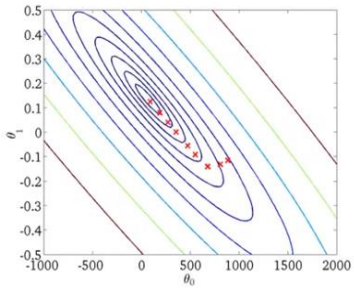
Objectives

- Outline the three variants of gradient descent:
  - batch gradient descent
  - stochastic gradient descent
  - mini-batch gradient descent
- Illustrate mini-batch gradient descent

46

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Batch Gradient Descent




```
for i = 1 to #iterations {
   for j = 1 to #observations {
         calculate the gradient
   }
   update the weights
}
```

- Uses all of the training observations to calculate the exact gradient on each descent step
- Results in a smooth progression to the error minima

47

As was noted previously (Chapter 3), the weights associated with a given error minimum are located using the following procedure:

1. Initialize the weight vector to small random values, $\mathbf{w}^{(0)}$.
2. Use an optimization method to determine the update vector, $\boldsymbol{\delta}^{(t)}$.
3. Add the update vector to the weight values from the previous iteration to generate new estimates:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \boldsymbol{\delta}^{(t)}$$

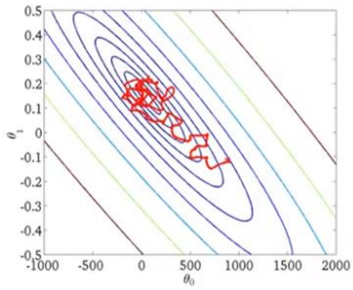
4. If none of the specified convergence criteria has been achieved, then go to step 2.

When the update vector uses the vector of partial derivatives of the error function with respect to the weights (i.e. the gradient) to determine the direction that minimizes the error function, the learning method is known as *gradient descent*. In the (default) batch variant of the gradient descent algorithm, generation of the weight update vector is determined by using all of the examples in the training set. That is, the exact gradient is calculated, ensuring a relatively smooth progression to the error minima.

For a linear neuron, with squared error, the error surface is a quadratic bowl. The vertical cross-sections are parabolas, and the horizontal cross-sections are ellipses. For multilayer networks, the error surface is much more complicated. But, provided that the weights are not too big, locally the error surface can be well approximated by a piece of a quadratic bowl (Hinton 2013).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Stochastic Gradient Descent




```
for i = 1 to #iterations {
   for j = 1 to #observations {
         calculate the gradient
         update the weights
   }
}
```

- Uses a single training observation to calculate an approximate gradient for each descent step
- Results in a chaotic progression to the error minima

48

However, when the training data set is large, computing the exact gradient is computationally expensive. The entire training data set must be assessed on each step down the gradient. Moreover, if the data is redundant, the error gradient on the second half of the data will be almost identical to the gradient on the first half. In this event it would be a waste of time to compute the gradient on the whole data set. You would be better off computing the gradient on a subset of the weights, updating the weights, and then repeat on a new subset. In this case, each weight update is based on an approximation to the true gradient. But as long as it points in approximately the same direction as the exact gradient, the approximate gradient is a useful alternative to computing the exact gradient (Hinton 2007).

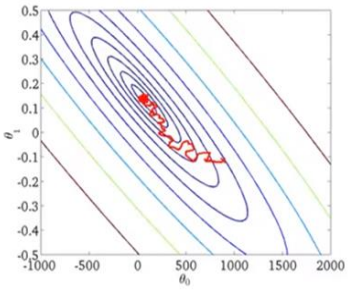
Taken to extremes, calculation of the approximate gradient can be based on a single training case. The weights are then updated and the gradient is calculated on the next case. This is known as *stochastic gradient descent* (also known as *online learning*). Stochastic gradient descent is very effective, particularly when combined with a momentum term, $\boldsymbol{\delta}^{(t-1)}$.

$$\boldsymbol{\delta}^{(t)} = -\eta \nabla \mathbf{g}^{(t)} + \alpha \boldsymbol{\delta}^{(t-1)}$$

Because stochastic gradient descent does not need to consider the entire training data set when calculating each descent step's gradient, it is usually faster than batch gradient descent. However, because each iteration is trying to better fit a single observation, some of the gradients might actually point away from the minima. This means that, although stochastic gradient descent will generally move the parameters in the direction of an error minima, it might not do so on each iteration. The result is a more circuitous path. In fact, stochastic gradient descent does not actually converge in the same sense as batch gradient descent does; rather it wanders around continuously in some region that is close to the minima (Ng 2013).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Mini-Batch Gradient Descent




```
divide the training data into batches
for i = 1 to #iterations {
   for j = 1 to #batches {
      calculate the gradient for batch j
      update the weights
   }
}
```

- Uses a subset of the training observations to calculate an approximate gradient on each descent step
- Results in a less erratic progression to the error minima than stochastic gradient descent

49

A compromise between batch gradient descent and single-case stochastic gradient descent is to divide the training data into small batches, compute the gradient using a single batch, make an update, and then move to the next batch of observations. This is known as *mini-batch gradient descent*.

Like single-case stochastic gradient descent, mini-batch gradient descent is typically faster than batch gradient descent. And because the weights are updated less often than when single-observation stochastic gradient descent is used, mini-batch gradient descent typically uses less computation updating the weights, i.e. each mini-batch computes the gradients for a number of cases in parallel. However, it is important that the mini-batches contain approximately balanced classes (Hinton 2013). One way to approximate balanced classes in each mini-batch is to randomly order the training data.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Mini-Batch Gradient Descent

Mini-batch gradient descent is often the method of choice when training neural networks on large data sets, particularly if the data set contains repetitive information. To illustrate mini-batch gradient descent learning, the variable annuity data set will be used. It is described below.

A target marketing campaign for a bank was undertaken to identify segments of customers who are likely to respond to a variable annuity (an insurance product) marketing campaign. A data set was created with 32,264 cases (banking customers) and 47 inputs. The 47 input variables represent other product usage in a three-month period and demographics. Two of the inputs are nominally scaled; the others are interval or binary. A binary target variable, **Ins**, indicates whether the customer bought the variable annuity product. The data set is called **develop**. The variables in the **develop** data set are listed below.

Variable	Description	Role	Level
ATM	Used ATM service (1=yes, 0=no)	Input	Binary
ATMAmt	ATM withdrawal amount	Input	Interval
AcctAge	Age of oldest account in years	Input	Interval
Age	Age of customer in years	Input	Interval
Branch	Branch of Bank (B1 – B19)	Rejected	Nominal
CC	Has credit card account (1=yes, 0=no)	Input	Binary
CCBal	Credit card balance	Input	Interval
CCPurc	Number of credit card purchases	Input	Interval
CD	Has certificate of deposit (1=yes, 0=no)	Input	Binary
CDBal	Certificate of deposit balance	Input	Interval
CRScore	Credit score	Input	Interval
CashBk	Number of times customer received cash back	Input	Interval
Checks	Number of checks	Input	Interval
DDA	Checking account (1=yes, 0=no)	Input	Binary
DDABal	Checking account balance	Input	Interval
Dep	Number of checking deposits	Input	Interval
DepAmt	Amount deposited	Input	Interval
DirDep	Direct deposit (1=yes, 0=no)	Input	Binary
HMOwn	Owns home (1=yes, 0=no)	Input	Binary

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

HMVal	Home value in thousands of dollars	Input	Interval
ILS	Has installment loan (1=yes, 0=no)	Input	Binary
ILSBal	Installment loan balance	Input	Interval
IRA	Has retirement account (1=yes, 0=no)	Input	Binary
IRABal	Retirement account balance	Input	Interval
InArea	Local address (1=yes, 0=no)	Input	Binary
Income	Income in thousands of dollars	Input	Interval
Ins	Purchase variable annuity account (1=yes, 0=no)	Target	Binary
Inv	Has investment account (1=yes, 0=no)	Input	Binary
InvBal	Investment account balance	Input	Interval
LOC	Has line of credit (1=yes, 0=no)	Input	Binary
LOCBal	Line of credit balance	Input	Interval
LORes	Length of residence in years	Input	Interval
MM	Has money market account (1=yes, 0=no)	Input	Binary
MMBal	Money market balance	Input	Interval
MMCred	Number of money market credits	Input	Interval
MTG	Has mortgage account (1=yes, 0=no)	Input	Binary
MTGBal	Mortgage balance	Input	Interval
Moved	Recent address change (1=yes, 0=no)	Input	Binary
NSF	Occurrence of insufficient funds (1=yes, 0=no)	Input	Binary
NSFAmt	Amount of insufficient funds	Input	Interval
POS	Number of point of sale transactions	Input	Interval
POSAmt	Amount in point of sale transactions	Input	Interval
Phone	Number of times customer used telephone banking	Input	Interval
Res	Area classification (R=rural, S=suburb, U=urban)	Rejected	Nominal
SDB	Has a safety deposit box (1=yes, 0=no)	Input	Binary
Sav	Saving account (1=yes, 0=no)	Input	Binary
SavBal	Saving balance	Input	Interval
Teller	Number of teller visits	Input	Interval

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

1. Right-click **Diagrams** in the project panel, and select **Create Diagram** from the pop-up menu.
2. Enter **mini-batch** in the **Diagram Name** field and click **OK**.
3. Right-click **Data Sources** in the project panel and select **Create Data Source**.
4. Click **Next** to use a SAS table as the metadata source.
5. Click **Browse** and select the **develop** data table from the **dmnn41** library.
6. Click **Next**. There are 48 variables and 32,264 observations in the data set.
7. Click **Next** again to start the metadata definition process.
8. Select the **Advanced** setting and then click **Next** to proceed.
9. Change the Role of the **Ins** variable to **Target**.
10. Three interval variables have been wrongly declared as nominal variables because they had fewer than 20 unique values. So change the level of **CashBk**, **CCPurc**, and **MMCred** back to **Interval**.
11. Set the role of the **Branch** and **Res** nominal variables to **Rejected**. Click **Next**.

    The **Branch** variable has 19 levels. It should be collapsed to be useful. And the **Res** variable is an irrelevant predictor of variable annuity purchase.

12. Because the **develop** data set is oversampled, you need to enter the population prior probabilities to get accurate estimates of the probability of responding. So select the **Yes** radio button on the Decision Processing screen, and click **Next**.
13. Click the **Prior Probabilities** tab. To enter values, select **Yes** under the question "Do you want to enter new prior probabilities?" The population response rate in the previous campaign was actually only 2%, which means that 98% of the prospects refused, so fill in the values as shown below.

Level	Count	Prior	Adjusted Prior
1	11175	0.3464	0.02
0	21089	0.6536	0.98

14. Click **Next** three times, and then click **Finish** to complete the metadata definition process.
15. Drag and drop the **develop** data source onto the diagram.
16. Add a **Data Partition** node (Sample tab) to the diagram, and connect it to the input data source.
17. Change the percentages to **70%** Training, **30%** Validation, and **0%** Test.
18. Drag and drop a **SAS Code** node (Utilities tab) onto the diagram, and connect the **Data Partition** node to the **SAS Code** node. Your diagram should now look like the one shown below.

19. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.
20. In the Training Code editor, right-click in the white space and select **Open**.
21. Select the program **c4s5d1.sas** (in d:\workshop\winsas\dmnn41). The program is described below.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

First, the global system options are set. The addition of the NONOTES option to the NODATE and NONUMBER options is an important one. Each time that the NEURAL procedure is called, it writes information into the log file. Because PROC NEURAL will be called many times, this soon fills up the available space. The NONOTES option minimizes the information written out to the log file.

```
options nodate nonumber nonotes;
```

For convenience, the target and the (interval) input variables are stored as macro variables.

```
%let target=ins;

/* generate interval input variable list */
proc contents data=&EM_IMPORT_DATA(drop=&target)
     out=temp(keep=name type) noprint;
run;
proc sql noprint;
   select name into: inputs separated by " "
   from temp
   where type=1;
quit;
```

Next the macro definition begins. The macro takes four arguments: the data set to be modeled (**dsn**), the number of hidden units in the hidden layer (**nhidden**), the number of batches to generate (**nbatches**), and the number of passes through the data set (**nepochs**).

```
%macro mini_batch(dsn=,nhidden=,nbatches=,nepochs=);
```

In order to prevent the generation of an excessive amount of output information, ODS is used to close all of the standard output channels (e.g. LISTING, HTML).

```
    ods _all_ close;
```

Next a uniform random variable (**random**) is created. It is used to randomize the observation order. Based on this random value, the RANK procedure partitions the data set into mini-batches.

```
   data temp;
      set &dsn;
      random=ranuni(12345);
   run;
   proc rank data=temp groups=&nbatches out=temp(drop=random);
      var random;
      ranks batch;
   run;
```

The RANKS statement in PROC RANK assigns the variable **batch**, which ranges from zero to **nbatches** – 1, to each of the randomly ordered mini-batches.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

A %DO loop iterates through the training data set, splitting the training data into mini-batches. The subsets, labeled **training0** to **training**<**nbatches**-1>, are converted into DMDBs by and cataloged.

```
   %do j=0 %to %eval(&nbatches-1);
      data training&j(drop=batch);
         set temp(where=(batch=&j));
      run;
      proc dmdb data=training&j dmdbcat=cat;
         class &target;
         var &inputs;
         target &target;
      run;
   %end;
```

Next the network architecture and initial weight values are set up. A multi-layer perceptron architecture is used, with **nhidden** hidden units in the hidden layer. The architecture is written out to the file **architecture**, and the initial weights (obtained by preliminary training) are written out to **weights**.

```
   proc neural data=training0 dmdbcat=cat;
      input &inputs;
      target &target;
      archi mlp hidden=&nhidden;
      prelim 5 maxiter=10 outest=weights(where=(_NAME_='_LOWEST_'));
      save network=architecture;
   run;
```

The heart of the mini-batch learning algorithm is the %DO loop that follows. For each pass thorough the data set (**nepochs**), the program loops over the set of mini-batch files, first reading in the current weight file (**weights**), and then training the network for a single iteration (i.e. **maxiter**=1). After the single iteration, the **weights** file is updated (**outest**=).

```
   %do i=1 %to &nepochs;
      %do j=0 %to %eval(&nbatches-1);
         proc neural data=training&j dmdbcat=cat
network=architecture;
            initial inest=weights;
            train tech=bprop learn=0.5 momentum=0.9 maxiter=1
estiter=0
                  outest=weights(where=(_NAME_='_LAST_'));
         run;
      %end;
   %end;
```

After the mini-batch training, the ODS output listing is again turned on so that the model's results can be reported.

```
   ods listing;
```

A %MEND statement finalizes the macro definition.

```
%mend mini_batch;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The **mini_batch** macro definition is then called, passing it the data set to be analyzed (**dsn**=), the number of hidden units (**nhidden**=) in the hidden layer, the number of mini-batches to create from the training data set (**nbatches**=), and the number of passes through the training data (**nepochs**=).

```
%mini_batch(dsn=&EM_IMPORT_DATA,nhidden=3,nbatches=20,nepochs=50)
```

To assess the proposed solution, PROC NEURAL is again called, passing it the final weights derived by the **mini_batch** macro. First the training data is scored, and selected fit information is output to the **fit** data set. The scored training data is output to **&EM_EXPORT_TRAIN**. Then the validation data is scored, and the scores output to **&EM_EXPORT_VALIDATE**.

```
   title1 'Mini-Batch';
   proc neural data=&EM_IMPORT_DATA dmdbcat=cat
         network=architecture;
      initial inest=weights;
      score data=&EM_IMPORT_DATA
            outfit=fit(where=(_NAME_='OVERALL'))
         out=&EM_EXPORT_TRAIN;
      score data=&EM_IMPORT_VALIDATE out=&EM_EXPORT_VALIDATE;
   run;
```

Next the training data set fit statistics are reported. Specifically, the average squared error (**_ase_**), Akaike information criteria (**_aic_**), the Schwarz-Bayesian information criterion (**_sbc_**), and the misclassification rate (**_misc_**) are output (using PROC PRINT).

```
   proc print data=fit;
      var _ase_ _aic_ _sbc_ _misc_;
   run;
```

Finally, two frequency tables are listed (PROC FREQ), each displaying the chi-square statistic reflecting the degree of association between the actual classification (**f_ins**) and the predicted classification (**i_ins**). The first table displays the training data association.

```
proc freq data=&EM_EXPORT_TRAIN;
   tables f_ins*i_ins/chisq nopercent nocol;
run;
```

The second table shows the performance against the holdout (validation) data.

```
proc freq data=&EM_EXPORT_VALIDATE;
   tables f_ins*i_ins/chisq nopercent nocol;
run;
```

This completes the SAS Code node program.

22. Click the running man to run the program. Confirm the run when asked, and open the results when training is complete.

23. Scroll down to line **27** and examine the training data fit statistics.

```
Obs     _ASE_       _AIC_       _SBC_      _MISC_

 1     0.17106    23536.38    24700.00    0.25589
```

The average squared error was only about 17%, and there was a 25% misclassification rate.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

24. Scroll down to line **35** and examine the training data frequency information.

```
The FREQ Procedure

Table of F_Ins by I_Ins

F_Ins(From: Ins)
          I_Ins(Into: Ins)

Frequency|
Row Pct  |0       |1       |  Total
---------+--------+--------+
0        |  12385 |   2376 |  14761
         |  83.90 |  16.10 |
---------+--------+--------+
1        |   3403 |   4420 |   7823
         |  43.50 |  56.50 |
---------+--------+--------+
Total       15788     6796    22584


Statistics for Table of F_Ins by I_Ins

Statistic                     DF       Value      Prob
------------------------------------------------------
Chi-Square                     1   3967.7973    <.0001
Likelihood Ratio Chi-Square    1   3887.2222    <.0001
Continuity Adj. Chi-Square     1   3965.8769    <.0001
Mantel-Haenszel Chi-Square     1   3967.6216    <.0001
Phi Coefficient                       0.4192
Contingency Coefficient               0.3866
Cramer's V                            0.4192
```

There was strong evidence ($p < 0.0001$) of an association between the actual and predicted target classes. Moreover, the Cramer's V value (0.4192) suggests that the association was fairly strong.

25. Scroll a little farther down (line **82**) to see the model's performance against the validation data.

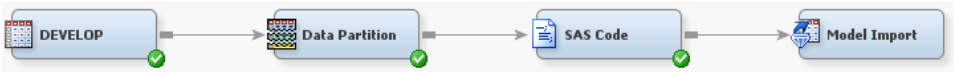
```
The FREQ Procedure

Table of F_Ins by I_Ins

F_Ins(From: Ins)
          I_Ins(Into: Ins)

Frequency|
Row Pct  |0       |1       |  Total
---------+--------+--------+
0        |   5276 |   1052 |   6328
         |  83.38 |  16.62 |
---------+--------+--------+
1        |   1489 |   1863 |   3352
         |  44.42 |  55.58 |
---------+--------+--------+
Total        6765     2915     9680
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

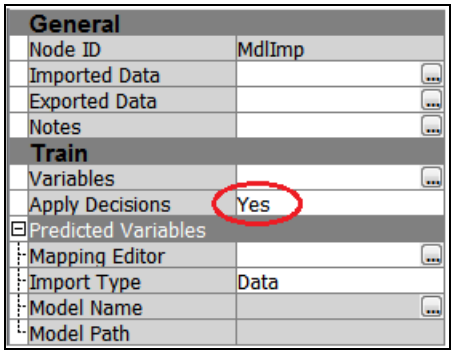
```
Statistics for Table of F_Ins by I_Ins

Statistic                     DF       Value      Prob
------------------------------------------------------
Chi-Square                     1   1579.9698    <.0001
Likelihood Ratio Chi-Square    1   1546.1605    <.0001
Continuity Adj. Chi-Square     1   1578.1194    <.0001
Mantel-Haenszel Chi-Square     1   1579.8066    <.0001
Phi Coefficient                       0.4040
Contingency Coefficient               0.3746
Cramer's V                            0.4040
```

Again there was strong evidence ($p < 0.0001$) of an association between the actual and predicted target classes, and the Cramer's V value (0.4040) suggests that the association was fairly strong.

26. Add a **Model Import** node (sixth from the right in the Model tab) to the flow and attach it to the output of the **SAS Code** node. The Model Import node allows the SAS Code node's mini-batch results to later be assessed using a Model Comparison node.

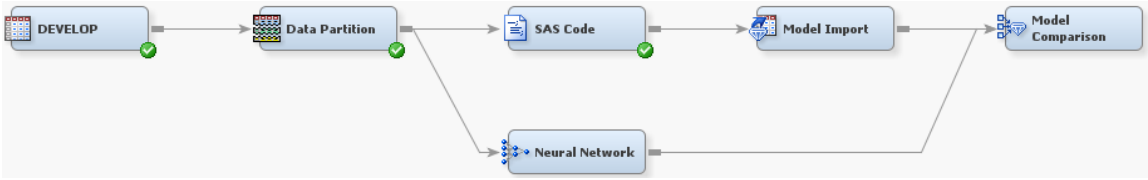


27. In the Model Import node's properties panel, change the Apply Decisions setting from **No** to **Yes**.



28. In order to compare the mini-batch algorithm to the traditional batch MLP, add a Neural Network node to the diagram and connect the Data Partition node to the neural network node.

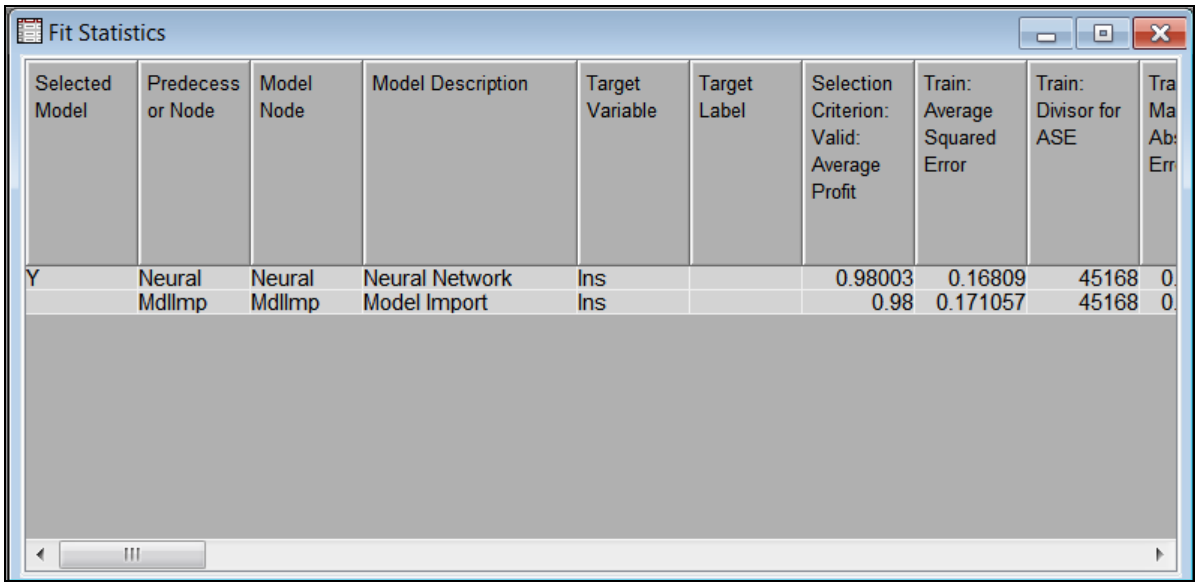
29. Finally, add a **Model Comparison** node (Assess tab) to the flow, connecting the output of both the **Model Import** and **Neural Network** nodes as input links to the **Model Comparison** node. Your diagram should now look similar to the one below.



30. Right-click the **Model Comparison** node and select **Run** from the pop-up menu. Confirm the run when asked, and open the results when training is complete

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

31. Maximize the **Fit Statistics** panel (upper right corner).



Fit Statistics

Selected Model	Predecessor Node	Model Node	Model Description	Target Variable	Target Label	Selection Criterion: Valid: Average Profit	Train: Average Squared Error	Train: Divisor for ASE	Tra Ma Ab Err
Y	Neural	Neural	Neural Network	Ins		0.98003	0.16809	45168	0.
	MdlImp	MdlImp	Model Import	Ins		0.98	0.171057	45168	0.

The (default) batch variant performed only slightly better on the validation data than did the mini-batch variant. In the era of big data, where it is often impractical to train a network on all of the available data, the mini-batch variant might actually be preferable.

32. Close the Results window.

33. Close the “mini-batch” diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.6 Chapter Summary

PROC NEURAL underlies the functionality of the Neural Network node. The NEURAL procedure can be used to construct custom network architectures including multiple hidden layer networks, and hybrid architectures like the useful generalized additive neural network (discussed in the next chapter).

To illustrate the power in coding the networks directly using PROC NEURAL, a variety of input selection techniques were considered. The most obvious input selection approach is to adopt a page from the linear regression, and identify the value of the input-to-hidden layer weights as a measure of input importance. Unfortunately, the interactive nature of a neural network is ignored in simple weight examination making this approach infeasible. A practical alternative is, sensitivity-based pruning. In sensitivity-based pruning, each input is sequentially held constant at a typical value (for example, at its median value) to see how the change in the input influences the predicted values. Small changes suggest the input can be pruned.

Principled network construction is also possible using PROC NEURAL. For example, in the sequential construction network (SNC) methodology, hidden units are added one at a time. Each added hidden unit attempts to account for some of the residual information missed by the less complex network. Hidden unit additions continue until an added hidden unit results in a drop in the specified fit measure. In that case, the last hidden unit that was added is removed and construction stops.

A generalized additive neural network (GANN) is another method of principled network construction. A GANN is a compromise between a generalized linear model and an MLP. It has a separate (independent) neural network associated with each input**.** These networks form individual linear or nonlinear univariate functions. Because these networks do not have connections to more than one input, the magnitude of the coefficients associated each mini-network can be interpreted as the corresponding input's importance.

Finally, the NEURAL procedure can be used to control the learning process. The default method is batch gradient descent. When the training data set is large, computing the exact gradient can be computationally expensive. A trade-off between the (default) gradient descent method and single observation stochastic gradient descent in which the observations are learned one at a time, is to divide the training data into small batches, compute the gradient using a single batch, make an update, and then move to the next batch of observations. This is known as mini-batch gradient descent. Mini-batch gradient descent is the method of choice when training neural networks on large data sets, particularly if the data set contains repetitive information. Mini-batch gradient descent is typically faster than batch gradient descent. And because the weights are updated less often than when single observation stochastic gradient descent is used, mini-batch gradient descent typically uses less computation updating the weights.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.7 Solutions

Solutions to Exercises

1. Sensitivity-Based Pruning (Bernoulli Distributed Target)

The complete PROC NEURAL program is shown below.

```
proc neural data=&EM_IMPORT_DATA ranscale=0.1 dmdbcat=cat;
   input %EM_INTERVAL_INPUT / level=int std=std;
   target %EM_TARGET / level=nom;
   archi mlp hidden=3;
   prelim 5 maxiter=10;
   train outfit=stats;
   code file="&codefile";
run;
```

Because you were asked to consider only interval inputs, the **%EM_INTERVAL_INPUT** macro variable proves useful. Or you could simply have listed the interval variables as shown below.

```
input ADBDDA ATMCT ATRES DDADEP DDATOT INCOME INVEST SAVBAL /
      level=int std=std;
```

Specification of the TARGET statement is straightforward. Again the **%EM_TARGET** macro variable proves useful. Or you could specify the target variable directly, as follows:

```
target ACQUIRE / level=nom;
```

It is important that the target level be specified as nominal (**nom**) because SAS Enterprise Miner does not recognize binary as a level designation.

Finally, the ARCHI statement requires only that you specify the neural network architecture type (**mlp**) and how many hidden units are required (**hidden=3**).

To see the variable sensitivity at the first quartile (25th percentile), scroll to line **216**.

```
Quantile = Q1

variable    sensitivity

 ATRES         11.632
 DDATOT        12.233
 ATMCT         54.475
 INCOME        54.559
 INVEST        71.279
 DDADEP       159.390
 SAVBAL       308.987
 ADBDDA       355.764
```

ATRES was determined to be the least sensitive and, therefore, the most expendable input. The **DDATOT** variable is a close second.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The median (50th percentile) sensitivity listing appears at line **232**.

```
Quantile = MEDIAN

variable    sensitivity

 ATRES          9.629
 DDATOT        11.289
 INCOME        31.388
 ATMCT         38.555
 INVEST        71.279
 DDADEP       113.356
 ADBDDA       252.899
 SAVBAL       308.987
```

Again, **ATRES** was found to be the least sensitive (most dispensable) variable. And again **DDATOT** was a close second choice.

The listing of the sensitivity networks at the third quartile (75th percentile) starts at line **248**.

```
Quantile = Q3

variable    sensitivity

 DDATOT        10.654
 ATRES         15.396
 ATMCT         39.654
 INCOME        41.292
 INVEST        71.279
 SAVBAL       268.346
 DDADEP       375.220
 ADBDDA       934.465
```

For the first time, **DDATOT** was the most expendable input. **ATRES** is now the second choice.

2. Sequential Network Construction

The complete PROC NEURAL sequential network construction code, including the addition of the fourth hidden unit, is shown below.

```
proc neural data=&EM_IMPORT_DATA dmdbcat=cat;
   input %EM_INTERVAL_INPUT / level=int id=in;
   target %EM_INTERVAL_TARGET / level=int act=exp error=poisson
          id=out;
   title2 'One Hidden Unit Network';
   hidden 1 /id=h1;
   connect in h1 out;
   prelim 5 maxiter=10;
   train outfit=fit1(where=(_name_='OVERALL'));
   title2 'Two Hidden Unit Network';
   freeze in->h1 h1->out bias->h1;
   hidden 1 /id=h2;
   connect in h2 out;
   prelim 5 maxiter=10;
   train;
   thaw in->h1 h1->out bias->h1;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
   train outfit=fit2(where=(_name_='OVERALL'))
   title2 'Three Hidden Unit Network';
   freeze in->h1 h1->out bias->h1 in->h2 h2->out bias->h2;
   hidden 1 /id=h3;
   connect in h3 out;
   prelim 5 maxiter=10;
   train;
   thaw in->h1 h1->out bias->h1 in->h2 h2->out bias->h2;
   train outfit=fit3(where=(_name_='OVERALL'));

   /* --------- new code shown below ------------- */

   title2 'Four Hidden Unit Network';
   freeze in->h1 h1->out bias->h1 in->h2 h2->out bias->h2 in->h3
      h3->out bias->h3;
   hidden 1 /id=h4;
   connect in h4 out;
   prelim 5 maxiter=10;
   train;
   thaw in->h1 h1->out bias->h1 in->h2 h2->out bias->h2 in->h3
      h3->out bias->h3;
   train outfit=fit4(where=(_name_='OVERALL'));
run;
```

The code to print the resulting fit is straightforward. (See below.)

```
title2 'Four Hidden Unit Network';
proc print data=fit4 noobs label;
   var _sbc_;
run;
```

After the modified program is run, the SBC continues to rise. (See line **1358**.)

```
Four Hidden Unit Network

  Train:
Schwarz's
 Bayesian
Criterion

 510.336
```

In other words, adding a fourth hidden unit made the MLP's final fit to the Boston **HOUSING** data is even worse. This further strengthens the conclusion that the two-hidden-unit network is optimal.

End of Solutions

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Solutions to Student Activities (Polls and Quizzes)

4.01 Poll – Correct Answer

PROC DMDB must be called to catalog the input to PROC NEURAL every time that the input data set is used.

- ❍ True
- ❍ False

10

Each data set to be used by PROC NEURAL needs to be cataloged only once. The catalog name generated by PROC DMDB must be provided when the data set is subsequently used.

4.02 Poll – Correct Answer

Pruning the input with the smallest input-to-hidden weight can negatively impact the neural network's performance.

- ❍ True
- ❍ False

21

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

4.03 Poll – Correct Answer

The sequential network construction methodology creates a neural network consisting of multiple hidden layers.

- ❍ True
- ❍ False

29

Although each hidden unit added is called a *layer*, a layer consisting of a single hidden unit is just a single hidden unit. It is the connectivity pattern between the hidden units that matters, and in an SNC, the connectivity pattern results in a single hidden layer network.

4.04 Poll – Correct Answer

A generalized additive neural network is a nonlinear model.

- ❍ True
- ❍ False

42

Although the output of the GANN can be very nonlinear, the nonlinearity comes from the linear combination of the nonlinear terms. The model itself is linear with respect to its inputs.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Chapter 5 Augmented Networks

5.1 Time Delay Neural Networks 5-3

Demonstration: Implementing a Time Delay Neural Network 5-13

5.2 Surrogate Models 5-22

Demonstration: Interpreting a Neural Network with a Continuous Target 5-24

Demonstration: Interpreting a Neural Network with a Categorical Target (Self-Study) 5-27

5.3 Chapter Summary 5-31

5.4 Solutions 5-32

Solutions to Student Activities (Polls and Quizzes) 5-32

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

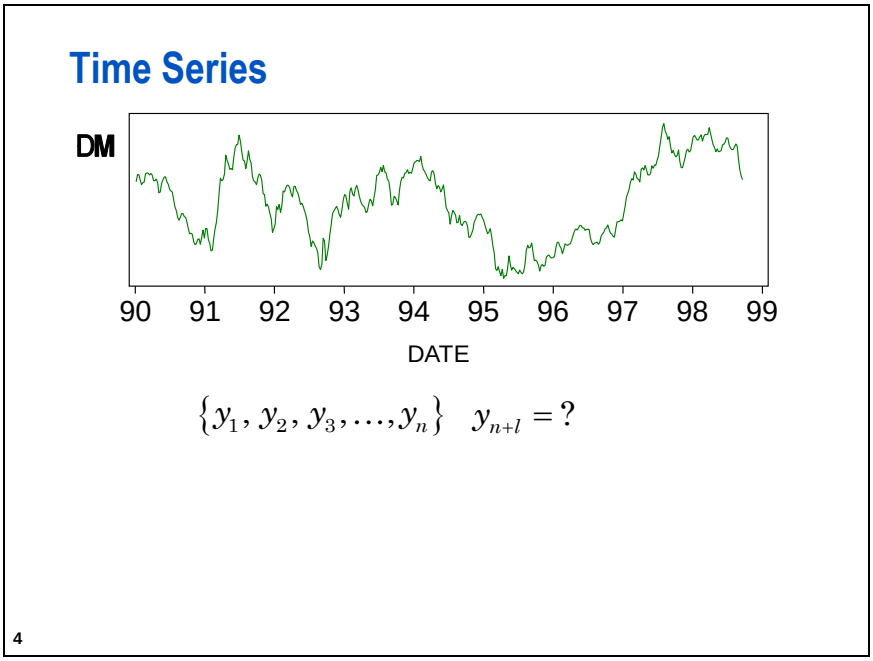
5.1 Time Delay Neural Networks

Objectives

- Outline the use of linear and vector autoregression to predict the continuation of a time series.
- Briefly describe time delay neural networks (TDNN).

3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Time series analysis involves the description and prediction of outcomes that are measured at sequential points in time. It has three related goals: forecasting (prediction), modeling, and characterization. Many time series can be effectively captured by an *autoregressive moving average* (ARMA) model:

$$\hat{y}_t = \beta_0 + \sum_{i=1}^{p} \beta_i y_{t-i} + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i} + \varepsilon_t$$

An ARMA model consists of two components:

1. The *autoregressive* (AR) component predicts that the current value of the series is equal to a constant offset, β_0 (often zero) plus a combination of the weighted average of a number of previous values. An autoregressive model of order *p* represents the expected value of the current value in the series, *y*, as a linear combination of *p* lagged values.

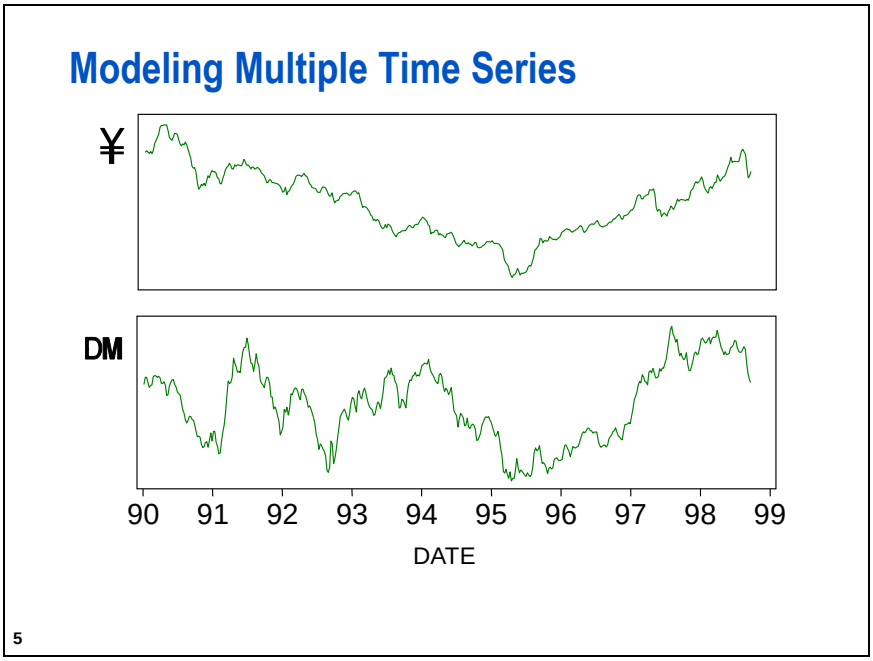
$$\hat{y}_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \ldots + \beta_k y_{t-p} = \beta_0 + \sum_{i=1}^{p} \beta_i y_{t-i}$$

2. The *moving average* (MA) component contends that the current value of a time series is given by a weighted (α_i) sum of a finite number of previous errors (called *shocks*), ε_{t-i}, plus the current error, ε_t. The moving average component is, therefore, given by the equation:

$$\hat{y}_t = \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}$$

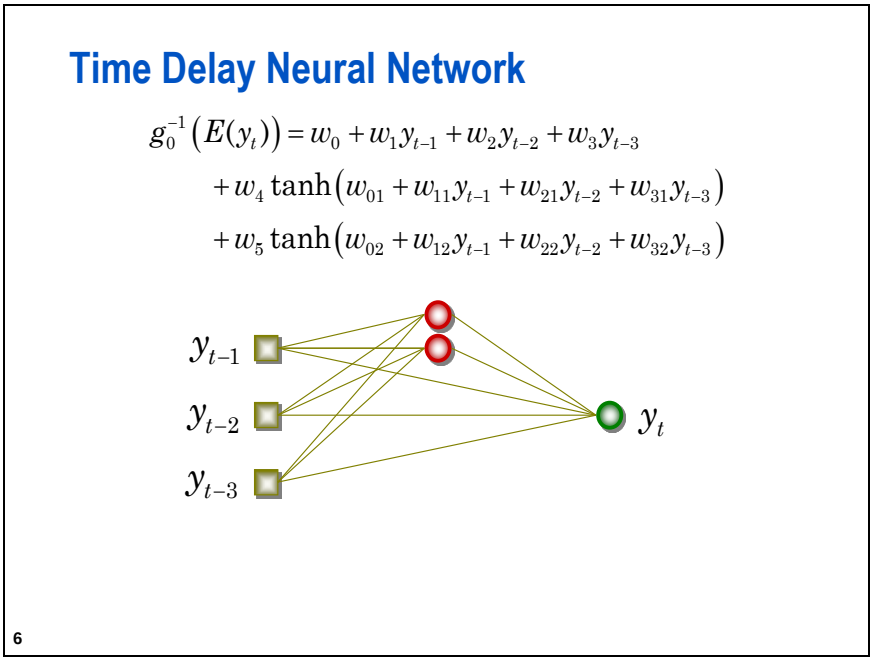
An autoregressive ***integrated*** moving average (ARIMA) model is a generalization of the ARMA model. The addition of the "integrated" component in the ARIMA model refers to the process of differencing (subtracting) time point values in order to achieve stationarity.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Many practical problems involve related time series. For example, to predict the exchange rate of the US dollar, it might be helpful to consider the exchange rates of other currencies such as the yen and euro. Multivariate time series analyses use the cross-correlations between two or more time series. In a *vector autoregression* model (Holden 1995), the value of each series at time *t* is a linear combination of its own past values as well as past values of related time series.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



By treating successive values (y_t) of a time series as targets, and lagged values (y_{t-1}, y_{t-2}, ..., y_{t-k}) as inputs, the autoregression component of ARMA can be cast as a supervised prediction problem. A linear model of order k represents the expected value of the series at time t as a linear combination of k lagged values:

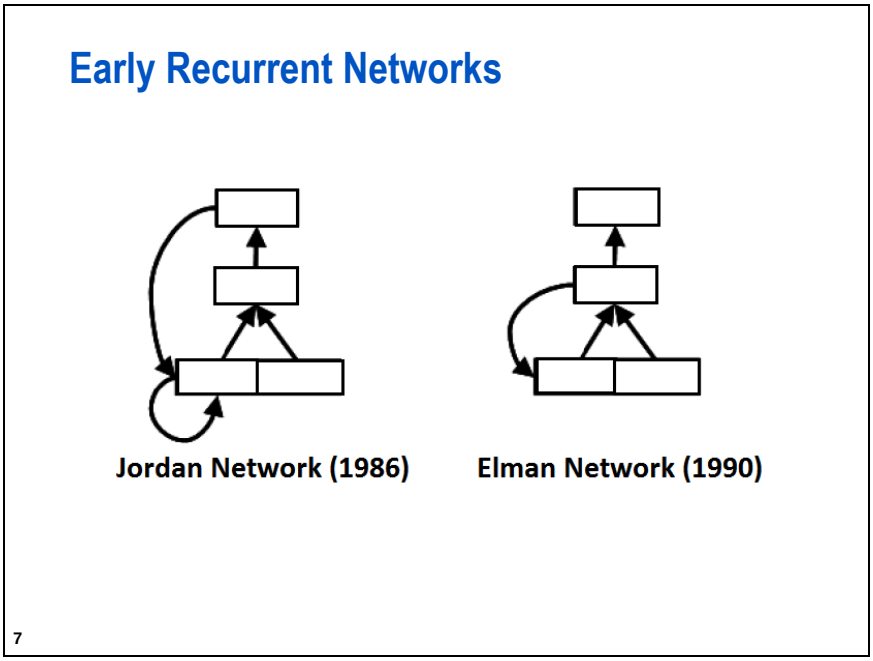
$$E(y_t) = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \ldots + \beta_k y_{t-k} + \varepsilon_t$$

Feeding time-lagged inputs into a multilayer neural network produces a ***nonlinear*** autoregression model. These models are known as *time delay neural networks* (TDNNs).

Often direct connections are also added. As was noted previously, the addition of a skip-layer seldom improves a neural network's performance. However, Leisch et al. (1999) showed that a multilayer neural network without a skip layer is inherently stationary. This means that it will likely perform poorly when it is applied to nonstationary data; and time series data is often nonstationary (e.g., trends or seasonality).

In mathematics and statistics, a *stationary process* is a stochastic process whose joint probability distribution does not change when shifted in time. Consequently, parameters such as the mean and variance, if they are present, also do not change over time and do not follow any trends. Stationarity is used as a tool in time series analysis, where the raw data is often transformed to become stationary; for example, economic data is often seasonal and/or dependent on a nonstationary price level. An important type of nonstationary process that does not include a trend-like behavior is the cyclostationary process (Wikipedia).

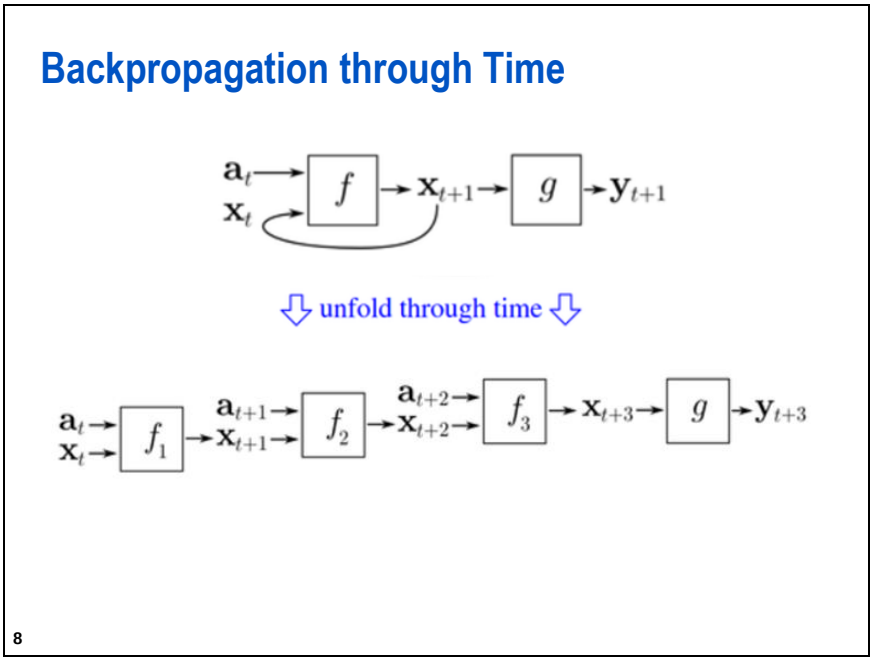
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Although neural network time series models will, in theory, achieve performance levels as good as those exhibited by classical time series approaches (Weigend and Gershenfeld 1994), because a TDNN models only the autoregression component of the ARMA paradigm, it often does not achieve the accuracy of the models that also consider the moving average component of ARMA.

The way to implement the moving average component is to use a *recurrent network*, in which a network's output is fed back as an input on the next iteration. This gives the neural network a "memory." There are many variants of the recurrent neural network paradigm. One of the earliest was the architecture proposed by Jordan (1986). Jordan's recurrent model is a three-layer network that added a set of *context units* to the input layer. These context units are fed from the output layer. The context units in a Jordan network also have a recurrent connection to themselves. At each time step, the input is propagated in a standard feed-forward fashion, and then a learning rule is applied. Another early recurrent network was Jeff Elman's (1990) model. In Elman's variant, which is also a three-layer network, the context units are fed from the hidden layer instead of the output layer. Thus, the context units maintain a copy of the previous values of the hidden units. In this way the network can maintain a state memory, allowing it to perform sequence-prediction tasks that are beyond the power of a standard multilayer perceptron (Wikipedia).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Paul Werbos' (1988) *backpropagation through time* (BPTT) algorithm is still widely used today. It is a gradient-based technique for training certain types of recurrent neural networks (e.g. Elman networks). The BPTT algorithm begins by "unfolding" the recurrent network. This amounts to reproducing the hidden layer (f) weight matrix k times, where k is the number of time steps to be retained. Each of the hidden layers receives two input vectors: the original vector to be learned at that time step in the sequence (a_k) and the output vector (i.e. context) from the previous hidden layer (x_k). For the first hidden layer (f_1), the context vector is initialized to zero.

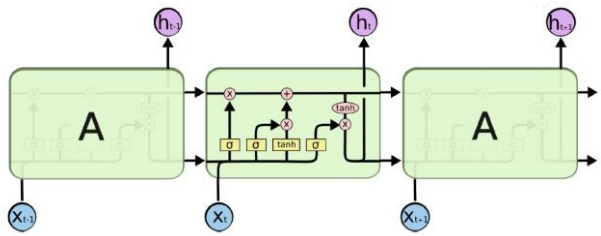
The input and context vectors are forward propagated through the network, and the error calculated. The error is then back-propagated through the whole unfolded network. The updated weight values in each instance of f are then averaged so that the weights in each hidden layer are again identical. Finally, the context for each time step is computed, i.e. $x_{k+1} = fx_k$.

A drawback of BPTT is that it generally has difficulties capturing long-lived memory effects, in large part because back-propagated error gradient information tends to "dilute" exponentially over time. In fact, a frequently stated opinion is that memory spans longer than 10 to 20 time steps are difficult to achieve with BPTT (Jaeger, 2002). The computational complexity of one epoch of the algorithm is $O(T N^2)$, where T is the number of time lags to be learned, and N is the number of internal units (Jaeger, 2001). Moreover, BPTT has difficulty with local optima.

With recurrent neural networks, in general, local optima is a more significant problem than it is with feed-forward neural networks (Cuellar et. al., 2006).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Long Short-Term Memory

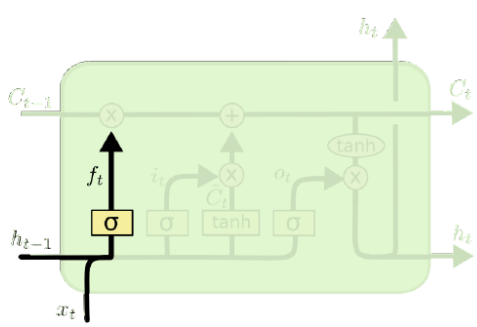


9 Source: Olah, C. (2015) "Understanding LSTM Networks"

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber. Explicitly designed to learn long-lived dependencies, LSTMs work well on a large variety of problems, and are now widely used (Olah 2015).

Similar to Werbos' BPTT paradigm (described previously), LSTM networks use unfolding to represent the recurrent relationship. But instead of transitioning the input through a single weight matrix at each time step, LSTM has four components governing the time step's output.

The first component is a "forget gate," responsible for deciding what information to throw away. It returns a value between zero and one, with zero representing "completely forget the input" and one representing "pass the input through undiminished."

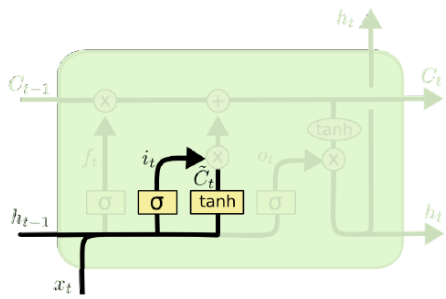


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

The illustrations in this section were originally presented in "Understanding LSTM Networks" found at http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

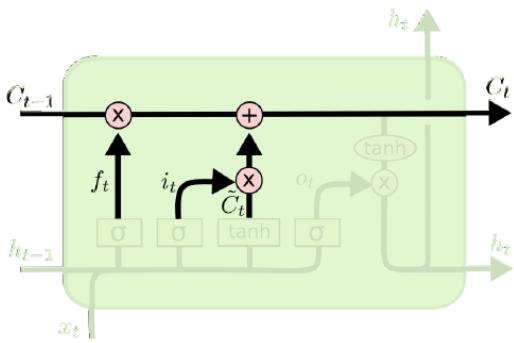
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The second component has two parts, a logistic sigmoid layer that decides which values will be updated, and a hyperbolic tangent (tanh) layer responsible for creating a new value ($\tilde{C}_t$) that could be added.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

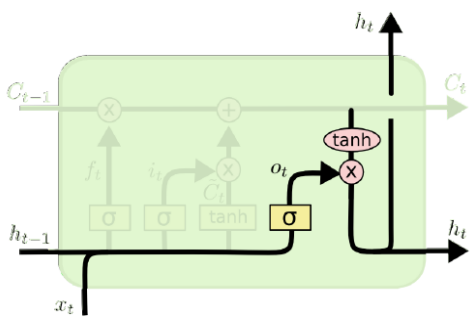
The third component actually performs the update, as directed by the first two components.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The ⊗ symbol above represents multiplication, and the ⊕ symbol represents addition.

And the final component, generates the time step's output (h_t), based on the input value (x_t) and the context (C_t) derived by component three.

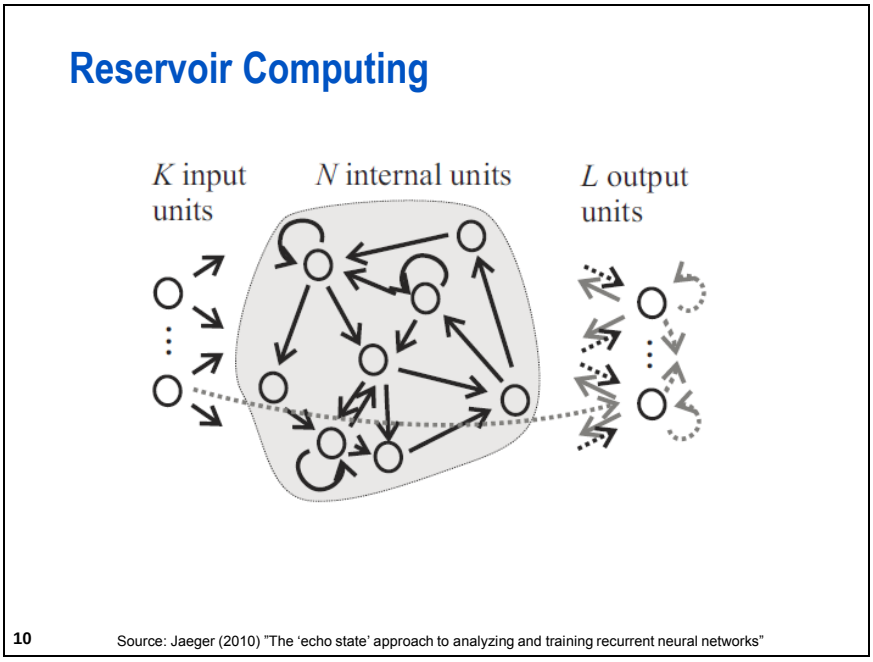


$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

To minimize LSTM's total error on a set of training sequences, BPTT can be used to change each weight in proportion to its derivative with respect to the error.

Although the LSTM overcomes the vanishing gradient problem by trapping the error in the memory portion of the block (Wikipedia), because of their inherent complexity, LSTM networks were often slow to train. With the increase in computational power in recent years, this problem has largely disappeared.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Traditional recurrent models are often severely limited by problems of slow convergence and local optima (Pearlmutter 1995; Suykens et al. 2008). In contrast, *reservoir computing* networks typically train quickly. Their speed arises from the fact that ***only the output layer learns***. This reduces the challenges associated with training a recurrent multilayer perceptron, to the much simpler task of training a linear perceptron. Moreover, if least squares gradient descent is used, there will be no local minima.

Reservoir computing networks have strong similarities to kernel machines (e.g. support vector machine). Both models map the input data into a higher-dimension *feature space*, allowing complex classification or regression decisions to be performed by a linear model. There are two main variants of the reservoir computing framework:

- Jaeger's (2001) *echo state network*,
- Maass' (2002) biologically inspired *liquid state machine*.

Although there are some fundamental differences between the liquid state and echo state paradigms (for example, echo state uses analog sigmoidal neurons whereas liquid state uses spiking neurons), the paradigms are quite similar. Therefore, only the echo state network variant is described below.

A potential third member of the reservoir computing paradigm is the *backpropagation decorrelation* learning rule, proposed by Schiller and Steil (2005).

Echo state networks (ESN) maintain the LSTM's ability to learn long-lived temporal dependencies by feeding the output of its randomly initialized and unchanging reservoir of hidden units, back though the reservoir. The reservoir thus serves as a memory, providing temporal context. More specifically, given an input, $\mathbf{u}(n)$, and a temporally associated target $\mathbf{y}(n)$, the reservoir activation is given by the equation:

$$\mathbf{x}(n) = \tanh(\mathbf{W}^{\mathbf{in}}[1;\mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1))$$

Here n goes from one to the number of data points in the training data set, $\mathbf{W}^{\mathbf{in}}$ is the randomly initialized matrix of input-to-hidden weights, and $\mathbf{W}$ is the reservoir's (sparse) matrix of internal connections.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Although **W** is also initialized to random values, in order to ensure convergence, it should have a *spectral radius* (maximum eigenvalue) less than or equal to one. This ensures that the reservoir will asymptotically wash-out any information from the initial conditions (Jaeger, 2007). In this event, the reservoir is said to have the *echo state property.* The following procedure seems to guarantee that a reservoir will have the echo state property (Jaeger, 2002):

1. Randomly generate weight matrix $\mathbf{W_0}$
2. Normalize W_0 to unit spectral radius by multiplying its elements by the reciprocal of the largest eigenvalue, i.e. $\mathbf{W_1} = 1/|\lambda_{\max}|\,\mathbf{W_0}$
3. Generate the reservoir matrix by multiplying $\mathbf{W_1}$ by a scalar (ε) less than one, i.e. $\mathbf{W} = \varepsilon\mathbf{W_1}$. This generates a reservoir matrix with a spectral radius of ε,

Often *leaky integrator* neurons are used in place of the aforementioned reservoir state equation:

$$\tilde{\mathbf{x}}(n) = \tanh(\mathbf{W^{in}}[1;\mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1))$$
$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + a\tilde{\mathbf{x}}(n)$$

Here $0 < \alpha \leq 1$ is the *leaking rate*. This turns the state equation into an exponential moving average, ensuring that the model will perform at least as well as the simple ESN (Lukosevicius et al., 2010).

Some variants of the ESN paradigm also include an output-to-reservoir feedback loop. This variant of the leaky integrator reservoir state is given by the following equation:

$$\tilde{\mathbf{x}}(n) = \tanh(\mathbf{W^{in}}[1;\mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1) + \mathbf{W^{fb}}\,y(n-1))$$

The output $\mathbf{y}(n\text{-}1)$ is set to zero on the first iteration. Thereafter, either looping the calculated output (see equation below) or feeding in the desired value (known as *teacher forcing*), provides the value of $\mathbf{y}(n\text{-}1)$. Like the other weight matrices, the feedback matrix $\mathbf{W^{fb}}$ is randomly initialized.

Although including an output-to-reservoir feedback loop means that the reservoir state equation no longer relies on fixed random input-driven dynamics to construct the output, Lukosevicius (2012) goes on to note that stability issues can arise. This led him to conclude that ***you should use feedback to the reservoir only if it is necessary for the task***.

After the state vectors have been calculated, taking the inner product of the (trainable) output weights ($\mathbf{W^{out}}$) and the merged the state and input vectors yields the final output ($\mathbf{y}(n)$). More formally,

$$\mathbf{y}(n) = g(\mathbf{W}^{out}[\mathbf{x}(n);\mathbf{u}\{n)]),$$

where $g(.)$ is the output activation function, typically either identity or logistic.

Training of the output layer weights can be performed by many algorithms. Probably the most universal and stable solution is obtained using ridge regression (Lukosevicius, 2012). If **X** is the matrix of merged reservoir and input vectors, **Y** is the target matrix, and β is the regularization coefficient, then the output weight matrix can be calculated using Tikhonov regularization. (See below.)

$$\mathbf{W}^{out} = \mathbf{YX}'(\mathbf{XX}' + \beta\mathbf{I})^{-1}$$

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Implementing a Time Delay Neural Network

c5s1d1

In this demonstration, you construct a time delay neural network and evaluate its performance on the Box and Jenkins (1976) **WINE** time series data, containing information about consumption levels of red and white wine in Australia over the period January 1980 to December 1994. The variables are listed below.

Variable	Description	Role	Level
date	Current time point	Time ID	Interval
red	Amount of red wine consumed	Target	Interval
white	Amount of white wine consumed	Target	Interval

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **tdnn** in the **Diagram Name** field, and click **OK** to save the change.
3. Right-click **Data Sources** in the project panel and select **Create Data Source**.
4. Click **Next** to create a SAS data table.
5. Select the **WINE** data table (in the d:\workshop\winsas\DMNN41 library). Click **OK**.
6. Click **Next** to display information about the **WINE** data. There are 180 observations and 3 variables.
7. Click **Next** again to start the metadata definition process.
8. The **Basic** setting suffices in this case, so click **Next** to proceed.
9. Change the role of the **DATE** variable to **Time ID**. Also change the role of both the **RED** and **WHITE** variables to **Target**.

Name	Role	Level	Report	Order	Drop	Lower Limit
DATE	Time ID	Interval	No		No	.
RED	Target	Interval	No		No	.
WHITE	Target	Interval	No		No	.

As was noted previously, unlike most other modeling techniques, neural networks can fit multiple targets ***in the same model***. By specifying two active target variables, predictions for both red and white wine consumption will simultaneously be made.

10. Click **Next** three times, and then click **Finish** to complete the data source definition.
11. Drag and drop the newly created **WINE** data source onto the diagram.
12. Add a **SAS Code** node (Utility tab) to the diagram, and connect it to the **WINE** data source.

13. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

14. In the (default) Training Code editor, right-click in the white space and select **Open**.

15. Select the program **c5s1d1a.sas** (in d:\workshop\winsas\dmnn41).

    Because the default neural network assumes that continuous target variables are normally distributed, the program first calls PROC UNIVARIATE to assess the normality of **RED** and **WHITE**.

```
proc univariate data=&EM_IMPORT_DATA normal;
   var %EM_TARGET;
run;
```

16. Close the SAS Code node, and save your changes when asked.

17. Right-click the **SAS Code** node and rename the node **Univariate**. Click **OK** to confirm the change.

18. Right-click the **Univariate** node and select **Run** from the pop-up menu. Click **OK** to confirm the run.

19. Open the results when the run is completed.

20. Scroll down to the tests for normality for the **RED** target variable (line **56**). All of the tests suggest that the H_0 of normality should be rejected at the alpha=0.05 level.

```
                    Tests for Normality

Test                   --Statistic---    -----p Value------

Shapiro-Wilk           W     0.973467    Pr < W      0.0016
Kolmogorov-Smirnov     D     0.089131    Pr > D     <0.0100
Cramer-von Mises       W-Sq  0.243287    Pr > W-Sq  <0.0050
Anderson-Darling       A-Sq  1.459864    Pr > A-Sq  <0.0050
```

21. Scroll farther down (to line **133**) to see the normality information on the **WHITE** target. Again, the *p*-values suggest that the null hypothesis of normality should be rejected at the alpha=0.05 level.

```
                    Tests for Normality

Test                   --Statistic---    -----p Value------

Shapiro-Wilk           W     0.956433    Pr < W     <0.0001
Kolmogorov-Smirnov     D     0.091811    Pr > D     <0.0100
Cramer-von Mises       W-Sq  0.251973    Pr > W-Sq  <0.0050
Anderson-Darling       A-Sq  1.640071    Pr > A-Sq  <0.0050
```

22. Close the SAS Code node's Results window.

23. Add another **SAS Code** node (Utility tab) to the diagram, and connect it to the **WINE** data source.

24. Activate the SAS Code (2) node's properties panel and click the ellipsis (**...**) next to **Code Editor**.

25. In the (default) Training Code editor, right-click in the white space and select **Open**.

26. Select the program **c5s1d1b.sas** (in d:\workshop\winsas\dmnn41). This program generates the lagged inputs and performs data splitting into training and validation data sets.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The macro program is defined below.

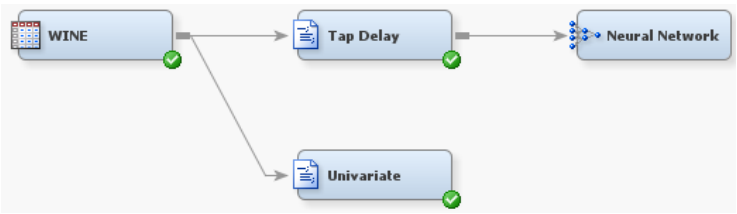
```
%macro tap_delay(ntaps=,cutoff=);
   data &EM_EXPORT_TRAIN &EM_EXPORT_VALIDATE;
      set &EM_IMPORT_DATA;
      %do i=1 %to &ntaps;
         if red ge 0 then red=log(red+1);
         if white ge 0 then white=log(white+1);
         r&i=lag&i(red);
         w&i=lag&i(white);
      %end;
      if _n_ > &ntaps then do;
         if date < &cutoff then output &EM_EXPORT_TRAIN;
         else output &EM_EXPORT_VALIDATE;
      end;
   run;
%mend tap_delay;

%tap_delay(ntaps=12,cutoff='1jan92'd)
```

Because the UNIVARIATE results showed that neither target variable was normally distributed, the macro first LOG-transforms the target variables.

In addition to increasing the normality of a skewed distribution, LOG-transforming the targets can also linearize decelerating time series and mitigate the influence of outliers.

Then **NTAPS** (in this demonstration, 12 months) lagged inputs are generated for each target variable. The macro also splits the data into training and validation data sets at the date value specified by the CUTOFF= argument. Here, if the date is between January 1981 and December 1991, the case is written to the training data table **&EM_EXPORT_TRAIN**. Otherwise, the case is written into the validation data set (**&EM_EXPORT_VALIDATE**).

27. Close the SAS Code node, and save your changes when asked.
28. Right-click the **SAS Code (2)** node and rename the node **Tap Delay**.
29. Click **OK** to confirm the change.
30. Right-click the **Tap Delay** node, and select **Run** from the pop-up menu.
31. Click **OK** to confirm the run, and then **OK** again when the run is completed.
32. Add a **Neural Network** node (Model tab) to the flow and connect it to the **Tap Delay** node. Your diagram should look similar to the one shown below.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

33. Activate the Neural Network node's properties panel, and click the ellipsis (…) next to **Variables**.

Name	Use	Report	Role	Level
RED	**Yes**	**No**	Target	Interval
WHITE	**Yes**	**No**	Target	Interval
r1	**Default**	**No**	Input	Interval
r10	**Default**	**No**	Input	Interval
r11	**Default**	**No**	Input	Interval
r12	**Default**	**No**	Input	Interval
r2	**Default**	**No**	Input	Interval
r3	**Default**	**No**	Input	Interval
r4	**Default**	**No**	Input	Interval
r5	**Default**	**No**	Input	Interval
r6	**Default**	**No**	Input	Interval
r7	**Default**	**No**	Input	Interval
r8	**Default**	**No**	Input	Interval
r9	**Default**	**No**	Input	Interval
w1	**Default**	**No**	Input	Interval
w10	**Default**	**No**	Input	Interval
w11	**Default**	**No**	Input	Interval
w12	**Default**	**No**	Input	Interval
w2	**Default**	**No**	Input	Interval
w3	**Default**	**No**	Input	Interval
w4	**Default**	**No**	Input	Interval
w5	**Default**	**No**	Input	Interval
w6	**Default**	**No**	Input	Interval
w7	**Default**	**No**	Input	Interval
w8	**Default**	**No**	Input	Interval
w9	**Default**	**No**	Input	Interval

Notice that 12 lagged red wine values (**r1** to **r12**), and 12 white wine lagged variables (**w1** to **w12**) have been created as inputs. Because all the variables will be used to predict the wine consumption levels, the network will perform vector autoregression.

34. Click **OK** to close the Variables window.

35. Next click the ellipsis (…) next to **Network**.

36. Change the **Direct Connection** option to **Yes** and reduce the **Number of Hidden Units** to **1**.

.. Property	Value
Architecture	Multilayer Perceptron
Direct Connection	Yes
Number of Hidden Units	1
Randomization Distribution	Normal
Randomization Center	0.0
Randomization Scale	0.1
Input Standardization	Standard Deviation
Hidden Layer Combination Function	Default
Hidden Layer Activation Function	Default
Hidden Bias	Yes
Target Layer Combination Function	Default
Target Layer Activation Function	Default

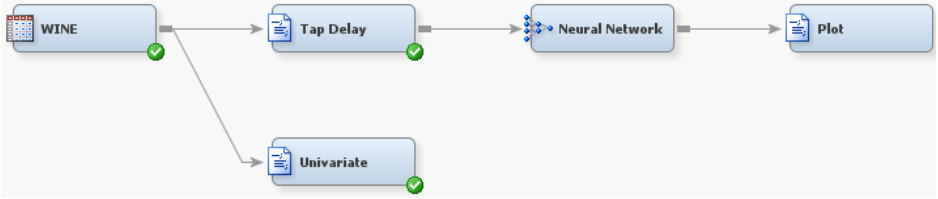
As was noted previously, although direct connections are seldom helpful, the addition of direct connections in a time series model can help manage non-stationarity.

37. Click **OK** to save your changes.

38. Add another **SAS Code** node (Utility tab) to the diagram. Connect it to the **Neural Network** node.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

39. Right-click the **SAS Code (3)** node and rename the node **Plot**. Click **OK** to confirm the change. Your flow should look similar to the one shown below.



40. Activate the Plot node's properties panel and click the ellipsis (**...**) next to **Code Editor**.

41. In the (default) Training Code editor, right-click in the white space and select **Open**. Read in program **c5s1d1c.sas** (in d:\workshop\winsas\dmnn41).

```
%macro plotseries(dsn=,target=,id=);
   data _null_;
      call symputx('plot',trim(scan("&dsn",2,'_')));
      call symputx('predicted',%str("p_&target"));
      call symputx('residual',%str("r_&target"));
   run;
   title1 h=1.5 f=swissb "&plot Data Set";
   proc gplot data=&dsn;
      symbol1 c=bl i=join v=none;
      symbol2 c=r i=join v=none l=3;
      legend1 label=none mode=share position=(top center inside);
      axis1 label=none;
      format date monyy6.;
      plot &target*date=1 &predicted*date=2/
          overlay frame legend=legend1 vaxis=axis1 description=&id;
   run;
   quit;
   data stats;
      set &dsn end=eof;
      sum+abs(&residual/&target);
      if eof then do;
         mape=(100/_n_)*sum;
         output;
      end;
   run;
   proc print data=stats noobs;
      format mape 18.4;
      var mape;
   run;
%mend plotseries;

%plotseries(dsn=&EM_IMPORT_DATA,target=RED,id="plot1")
%plotseries(dsn=&EM_IMPORT_VALIDATE,target=RED,id="plot2")
%plotseries(dsn=&EM_IMPORT_DATA,target=WHITE,id="plot3")
%plotseries(dsn=&EM_IMPORT_VALIDATE,target=WHITE,id="plot4")
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The program first creates three macro variables: **PLOT, PREDICTED**, and **RESIDUAL**. They hold the name of the data set used to generate the plots (i.e. TRAIN or VALIDATE), the predicted values, and the residual between the actual and predicted consumption values.

Next PROC GPLOT generates the training or validation plot of the wine consumption time series. To facilitate comparison, the predicted consumption is overlaid on top of the actual consumption data.

Finally, the mean absolute percentage error (MAPE) value of the series is calculated and displayed. The MAPE is a commonly used measure of accuracy in a time series. It is defined as follows:

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{(a_t - f_t)}{a_t}\right|$$

where a_t is the actual value and f_t is the forecast (predicted) value at time t.

42. Click the running man icon at the top of the page in the Plot node to run the program. Confirm the run when asked, and open the results when they become available.

43. Scroll down to line **26** to see the MAPE values for the red wine training and validation data (respectively). The values are reproduced below.

```
TRAIN Data Set

              mape

            0.0517


VALIDATE Data Set

              mape

            0.0949
```

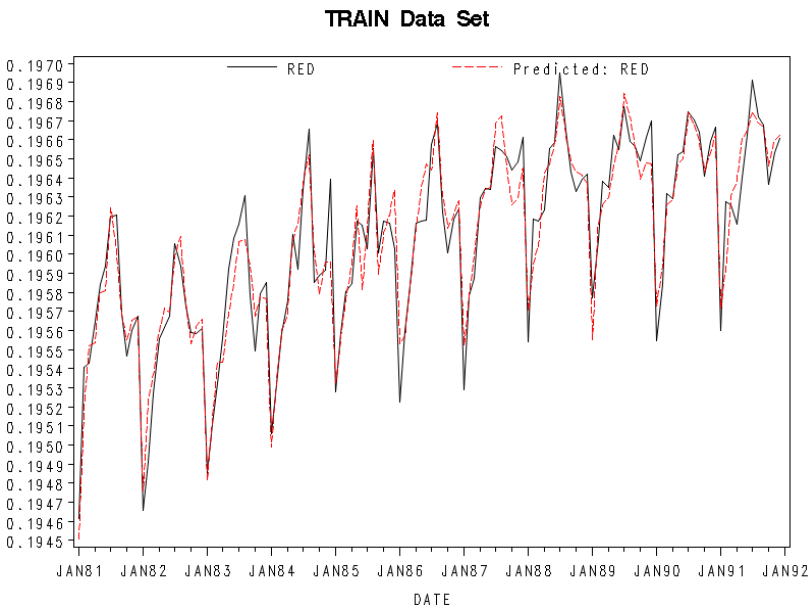
Both training and validation performance was excellent, generating MAPE values less than 1%. This is a remarkably small MAPE value.

44. The training and validation performance on the white wine series appear immediately below the red wine values. Again the model generated MAPE values less than 1% on both data sets.

```
TRAIN Data Set

              mape

            0.0423



VALIDATE Data Set

              mape

            0.0724
```

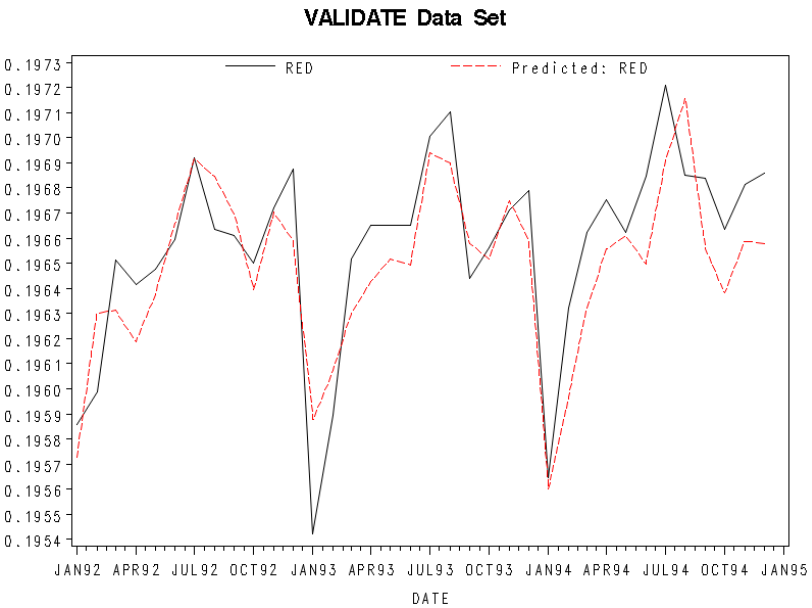
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

45. To see the plots, select **View** ⇨ **SAS Results** ⇨ **Train Graphs** at the top of the window. The first plot shows the predicted and actual red wine consumption over the training period.



As the small MAPE value suggested, the predicted (dashed) and actual (solid) consumption values match quite closely. Of more interest, however, is how the model performed on the holdout data.

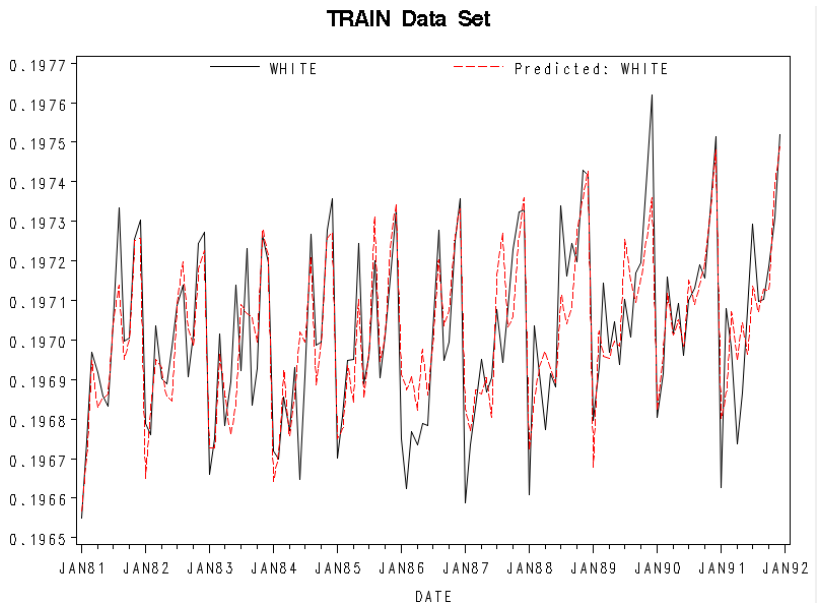
46. Click the right arrow (second button from the right) to see the actual and predicted red wine consumption values over the holdout (i.e. validation) period.



The TDNN predicted the consumption pattern over the three-year holdout period remarkably well, both in terms of direction of change and magnitude of change.

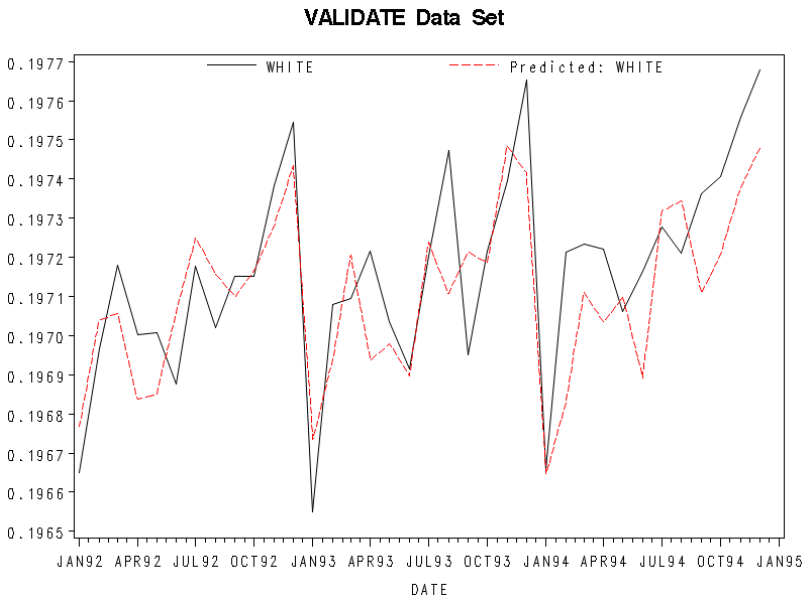
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

47. Click the right arrow (second button from the right) a third time to see the actual and predicted white wine consumption values on the training data.



Again the TDNN's predicted white wine consumption over the training period matched the actual consumption levels quite closely.

48. Click the right arrow one last time to see the TDNN's performance on the white wine validation data.



Again the neural network fairly closely matched the actual white wine consumption levels over the holdout period, reliably predicting (often even anticipating) the complex series of rises and falls.

49. Close the SAS Code node graph and Results windows.

50. Close the "tdnn" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

5.01 Poll

Neural networks cannot model both the autoregressive (AR) and moving average (MA) components of the ARMA model.

- ❍ True
- ❍ False

12

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

5.2 Surrogate Models

Objectives

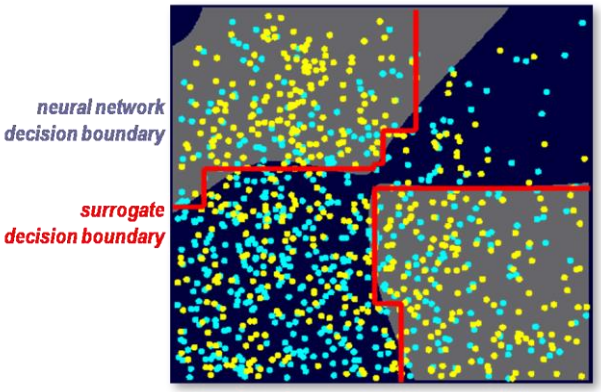
- Define a surrogate model.
- Illustrate how a surrogate model can be used to help understand a neural network's predictions.

16

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Surrogate Models

A surrogate model approximates an inscrutable model's predictions/decisions in order to facilitate interpretation.



17

A frequent criticism of neural networks is the difficulty understanding the basis of their predictions. The criticism stems from the complex parameterization in the model. Usually, little insight can be gained by directly analyzing a neural network's parameters. However, much can be gained by analyzing the neural network's predictions/decisions.

The idea behind a surrogate model is a simple one: use an interpretable model (such as a decision tree) to describe the relationship that a more opaque model (such as a neural network) has discovered.

A decision tree is an ideal surrogate model for a neural network. Not only are the if-then rules generated by a decision tree easy to interpret, in theory decision trees retain any nonlinear input-output relationships that are present in the neural network. There are also sound theoretical reasons for believing that a neural network's connectivity can be adequately summarized by if-then rules (Tsukimoto 2000).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Interpreting a Neural Network with a Continuous Target

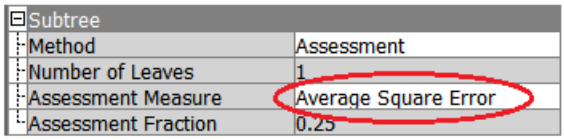
c5s2d1

This demonstration illustrates the use of a surrogate model (i.e. a decision tree) to help understand the basis on which a neural network is making its decisions. The **HOUSING** data (previously defined) will again be used. The goal is to interpret the Neural Network node's ***prediction*** of housing prices as a measure of willingness to pay for clean air.

1. Right-click **Diagrams**, and select **Create Diagram** from the pop-up menu.
2. Enter **surrogate** (**continuous)** in the **Diagram Name** field, and click **OK**.
3. Click Data **Sources** and drag and drop the **HOUSING** data onto the diagram.
4. Add a **Transform Variables** node (Modify tab) and connect it to the **HOUSING** data source.
5. In the Default Methods section, change the Interval Inputs value to **Maximum Normal**.
6. In order to use early stopping to help determine network complexity, add a **Data Partition** node (Sample tab) to the diagram and connect it to the **Transform Variables** node.
7. Change the Data Set Allocations values to **Training 70.0**, **Validation 30.0**, and **Test 0.0**.
8. Add a **Neural Network** node (Model tab) and connect it to the **Data Partition** node.
9. To fit the Poisson-distributed target, click the ellipsis (**…**) next to **Network**.
10. Change the Target Layer Activation Function property to **Exponential**.
11. Also change the Target Layer Error Function property to **Poisson**.
12. Right-click the **Neural Network** node and select **Run** from the pop-up menu.
13. Confirm the run and, when training is complete, click **OK** (not Results).
14. Add a **Metadata** node (Utility tab) and connect it to the output of the **Neural Network** node.
15. To change the training metadata, click the ellipsis (**.,.**) next to **Train** in the Variables section of the Metadata node's properties panel.
16. Because you want to model the neural network's prediction of the target, not the target itself, set the new role of the original target variable **MEDV** to **Rejected**. Also change the new role of the Neural Network node's prediction (**P_MEDV**) to **Target**.
17. Click **OK** to save.
18. Add a **Decision Tree** node (Model tab) and connect it to the **Metadata** node. Your final flow should look similar to the one shown below.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

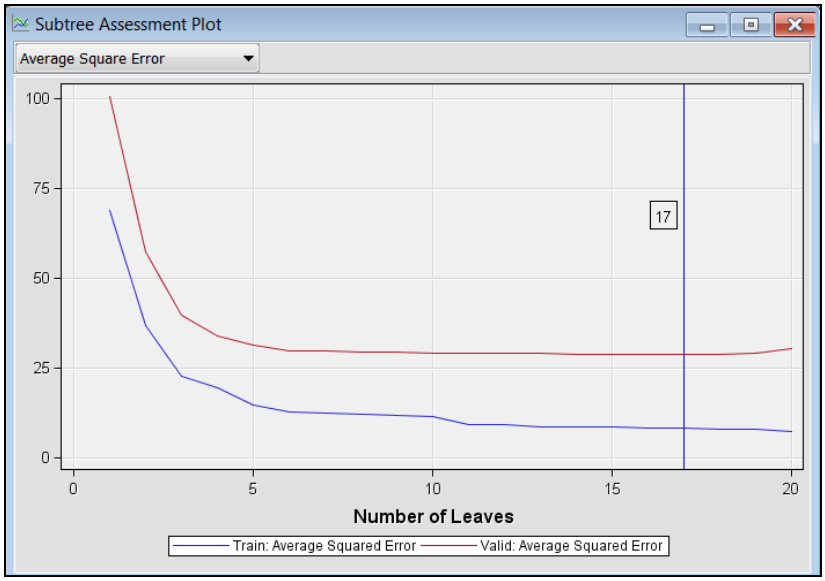
19. Activate the Decision Tree node's properties panel, and scroll down to the Subtree section. Change the Assessment Measure property to **Average Square Error**.



20. Right-click the default **Decision Tree** node and select **Run** from the pop-up menu.

21. Confirm the run, and examine the results when training is complete.

22. From the menu at the top of page, select **View** ⇨ **Model** ⇨ **Subtree Assessment Plot**.

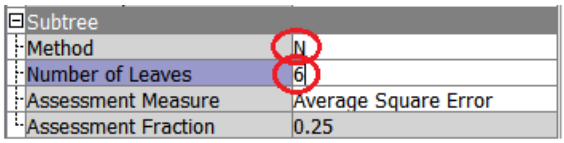


The plot shows that the decision tree generating be smallest average squared error on validation data (top line) had 17 leaves. But the plot also suggests that the improvement in average squared error was minimal after the fifth or sixth leaf was added. In other words, a 5- or 6-leaf tree seems to produce approximately the same fit as the more complex 17-leaf tree.

23. Close the Decision Tree node's results.

24. In the Decision Tree node's properties panel, again scroll down to the Subtree section. This time, change the Method value to **N** (which specifies the exact number of leaves to be generated).

25. Also change the Number of Leaves value to **6**.

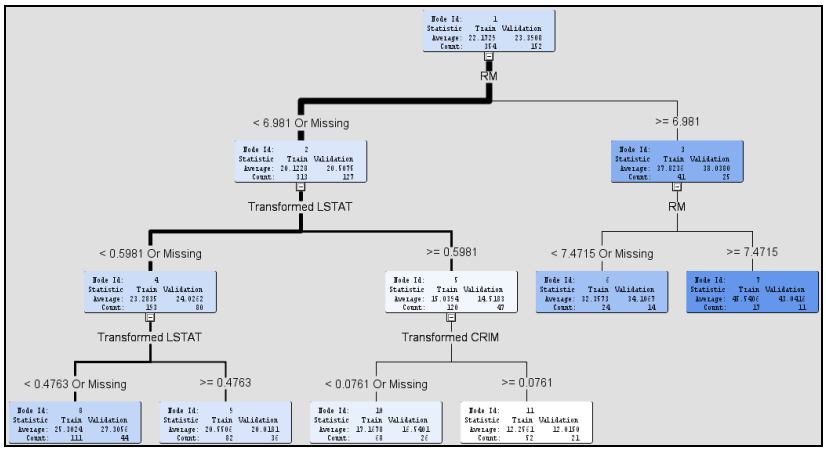


26. Right-click the default **Decision Tree** node and select **Run** from the pop-up menu.

27. Confirm the run, and examine the results when training is complete.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

28. Maximize and examine the derived decision tree (reproduced below).



The first split is based in the number of rooms (**RM**) the property contained. Properties missing the number of rooms value, or those with fewer than 6.981 rooms, branch to the left. Not surprisingly, these properties were predicted to generate a lower median value (**MEDV**) than properties with more 6.981 rooms (which branch to the right).

To further refine its left-branch (smaller houses) estimates, the neural network node next considered the **LSTAT** variable, which reports the percentage of the population that is "lower status." Again the results are not surprising. Small houses that are in areas with a smaller percentage of "lower status" individuals were predicted to have a larger market value than small houses in high percentage "lower status" individuals. As a final refinement to the left branch, the neural network considered crime rate (**CRIM**). Small properties in areas with less crime, tended to have a higher market value than those in high crime rate areas.

The only refinement to the estimated market value of the larger homes (the right branch) was to again subdivide the market value based on the number of rooms that houses contained. Specifically, the neural network predicted that homes with between 6.981 and 7.4715 rooms would generate less market value than homes with 7.4715 or more rooms.

29. Close the Decision Tree node's results.

30. Close the "surrogate (continuous)" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Interpreting a Neural Network with a Categorical Target (Self-Study)

c5s2d2

The **BANKING** data (detailed in Chapter 3), consisting of 2,000 banking customers who were offered a new banking product, is used again. The target, **ACQUIRE**, is a binary indicator of whether the prospect acquired the product. Decision processing was required to correct for oversampling. And the profit matrix was set to the inverse of the sample probabilities (the conforming profit matrix).

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **surrogate (categorical)** in the **Diagram Name** field and click **OK**.
3. Drag and drop the already-defined **BANKING** data source onto the diagram.
4. Add a **Data Partition** node (Sample tab) to the diagram and connect it to the **BANKING** node.
5. Set the split percentages to **70.0** (70%) for **Train**, **30.0** (30%) for **Validation**, and **0.0** (0%) for **Test**.
6. Add a **Neural Network** node (Model tab) to the diagram and connect it to the **Data Partition** node.
7. Right-click the **Neural Network** node, and select **Run** from the pop-up menu.
8. Click **OK** (not Results) when it is finished.
9. Add a **Metadata** node (Utility tab). Connect it to the output of the **Neural Network** node.
10. Click the ellipsis (**...**) next to the **Train** field (Variables section) in the Metadata node's properties panel, and set the new role of the current target variable (**ACQUIRE**) to **Rejected**.
11. Also set the new role of the decision variable **D_ACQUIRE** to **Target**. Save your changes.

Name	Hidden	Hide	Role	New Role	Level	New Level	New Order	New Report	Model
ACQUIRE	N	**Default**	Target	**Rejected**	Binary	**Default**	**Default**	**Default**	Neural
ADBDDA	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
ATMCT	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
ATRES	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
BP_ACQUIRE	N	**Default**	Assessment	**Default**	Interval	**Default**	**Default**	**Default**	
CP_ACQUIRE	N	**Default**	Assessment	**Default**	Interval	**Default**	**Default**	**Default**	
DDADEP	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
DDATOT	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
D_ACQUIRE	N	**Default**	Decision	**Target**	Nominal	**Default**	**Default**	**Default**	
EP_ACQUIRE	N	**Default**	Assessment	**Default**	Interval	**Default**	**Default**	**Default**	
F_ACQUIRE	N	**Default**	Classification	**Default**	Nominal	**Default**	**Default**	**Default**	
INCOME	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
INVEST	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
I_ACQUIRE	N	**Default**	Classification	**Default**	Nominal	**Default**	**Default**	**Default**	
P_ACQUIRE0	N	**Default**	Prediction	**Default**	Interval	**Default**	**Default**	**Default**	
P_ACQUIRE1	N	**Default**	Prediction	**Default**	Interval	**Default**	**Default**	**Default**	
R_ACQUIRE0	N	**Default**	Residual	**Default**	Interval	**Default**	**Default**	**Default**	
R_ACQUIRE1	N	**Default**	Residual	**Default**	Interval	**Default**	**Default**	**Default**	
SAVBAL	N	**Default**	Input	**Default**	Interval	**Default**	**Default**	**Default**	
U_ACQUIRE	N	**Default**	Classification	**Default**	Nominal	**Default**	**Default**	**Default**	
WARN	N	**Default**	Assessment	**Default**	Nominal	**Default**	**Default**	**Default**	
dataobs	N	**Default**	ID	**Default**	Interval	**Default**	**Default**	**Default**	

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

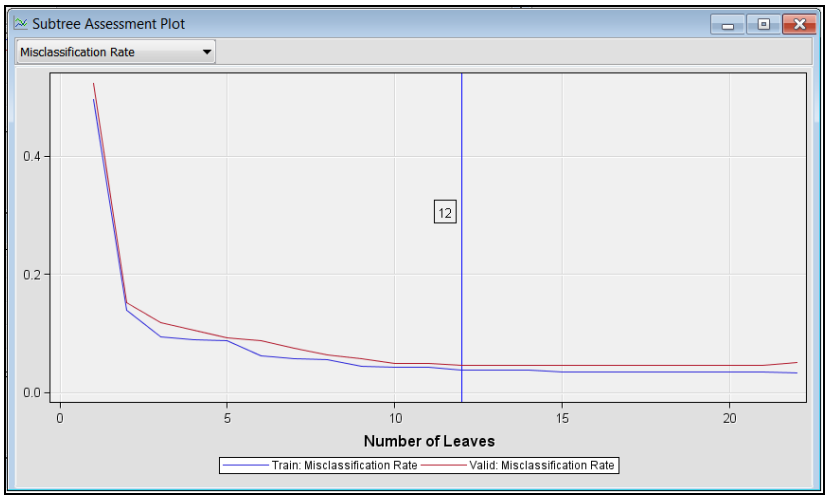
12. Finally, add a **Decision Tree** node (Model tab) and attach it to the output of the **Metadata** node. Your completed diagram should now look as follows:

13. Right-click the **Decision Tree** node and select **Run** from the pop-up menu.

14. Confirm the run, and select **Results** when training is complete.

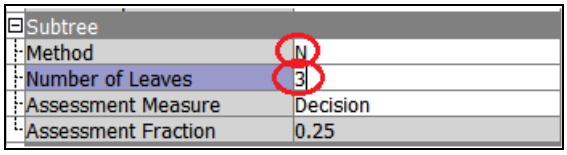
15. At the top of the window, select **View** ⇨ **Model** ⇨ **Subtree Assessment Plot**. Maximize the plot.

16. Click the down arrow in the upper left corner of the plot, and change the display criterion to **Misclassification Rate**. (See below.)



Notice that the proportion of cases misclassified drops drastically on the first split, a little further after the next split (into three leaves), and then changes very little after that. This suggests that a three-leaf tree largely describes the Neural Network node's decisions.

17. Scroll down to the Subtree section of the Decision Tree node's properties panel, and change the Method value to **N**. Also change the Number of Leaves value to **3**.



18. Re-run the tree by right-clicking the **Decision Tree** node and selecting **Run** from the pop-up menu.

19. Select **Results** when training is complete.

20. Maximize the Output window (bottom right corner).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

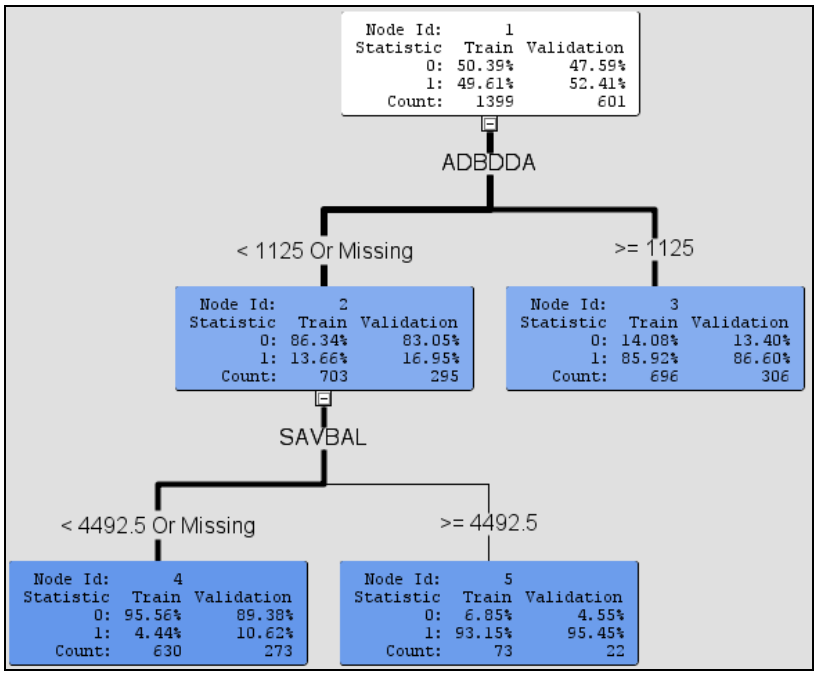
21. Go to line **145** to see the classification matrix results (reproduced below).

```
Data Role=VALIDATE Target=D_ACQUIRE Target Label=Decision: ACQUIRE

  False        True        False        True
Negative     Negative     Positive     Positive

   29           244          42           286
```

Remarkably, these results show that (244+286)/601= 88.2% of the validation decisions made by the neural network can be summarized by the simple three-leaf decision tree.

22. Minimize the Output window.

23. Maximize the Tree panel (bottom left corner) and examine the recommended three-leaf tree.



The surrogate model suggests that the neural network recommends contacting candidates who had an average daily balance in their checking account (**ABDDDA**) of $1,125 or more, or candidates with a smaller average daily balance but who had a savings balance (**SAVBAL**) of $4,492.50 or more.

24. Close the Decision Tree node's Tree panel and the Results window.

25. Close the "surrogate (categorical)" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

5.02 Poll

The purpose of a surrogate model is to remove the need for a complex model by replacing it with a simpler model.

- ❍ True
- ❍ False

20

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

5.3 Chapter Summary

Often neural networks can be augmented to good effect. Two such models are examined in this chapter: time delayed neural networks and surrogate models.

Feeding time-lagged inputs into a neural network produces a nonlinear autoregression model. Such models are known as time delay neural networks (TDNNs). Neural network time series models can achieve performance levels that are as good as classical approaches (Weigend and Gershenfeld 1994).

Surrogate networks can be used to help understand what a neural network is considering when it makes its predictions or decisions. Due to their interpretability, decision trees are often used.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

5.4 Solutions

Solutions to Student Activities (Polls and Quizzes)

5.01 Poll – Correct Answer

Neural networks cannot model both the autoregressive (AR) and moving average (MA) components of the ARMA model.

- True
- False

13

5.02 Poll – Correct Answer

The purpose of a surrogate model is to remove the need for a complex model by replacing it with a simpler model.

- True
- False

21

This is false. The purpose of a surrogate model is to explain, not replace, the inscrutable model.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Chapter 6 The HP Neural Node

6.1 Introduction to High-Performance Analytics 6-3

6.2 Overview of the HP Neural Node 6-9

Demonstration: Comparing One- and Two-Layer Networks 6-16

6.3 Chapter Summary 6-28

6.4 Solutions 6-29

Solutions to Student Activities (Polls and Quizzes) 6-29

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

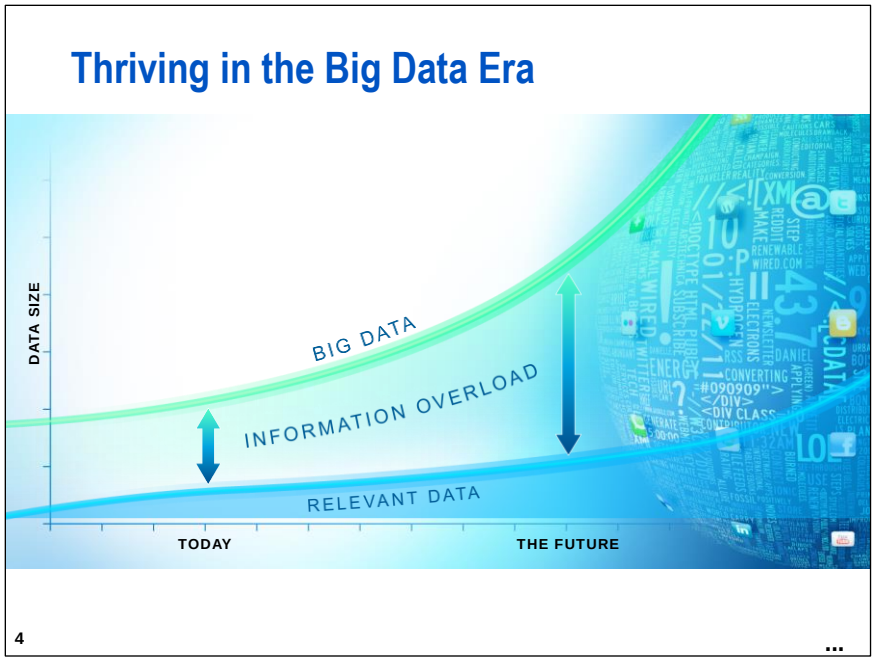
6.1 Introduction to High-Performance Analytics

Objectives

- Outline the challenge of big data.
- Introduce SAS High-Performance Analytics.

3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Big data is a popular term used to describe the exponential growth, availability, and use of information, both structured and unstructured. Here are examples of big data:

- Around 10,000 payment card transactions are made every second around the world.
- Walmart handles more than 1 million customer transactions an hour.
- More than 5 billion people are calling, texting, tweeting, and browsing websites on mobile phones.
- Radio Frequency ID systems generate up to 1,000 times the data of conventional bar code systems.
- Facebook has more than 901 million active users generating social interaction data.
- There are 340 million tweets sent per day. That is nearly 4,000 tweets per second.

Many organizations are concerned that the amount of amassed data is becoming so large that it is difficult to find the most valuable pieces of information. Until recently, organizations have either been limited to using subsets of their data, or they were constrained to simplistic analyses because the sheer volumes of data overwhelmed their processing platforms. But what is the point of collecting and storing terabytes of data if you cannot analyze it in full context, or if you have to wait hours or days to get results?

Now it is possible to apply high-performance analytics to analyze the massive amounts of data using technologies such as grid computing, in-database processing, and in-memory analytics. These big data technologies not only support the ability to collect large amounts of data, they provide the ability to understand it and use it for optimized decision making.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Performance Challenges

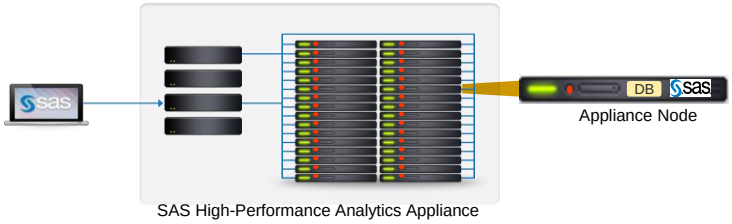
- Massive amounts of data are stored in databases. Extraction requires massive data movement, which results in very poor performance.
- Many data mining algorithms require multiple passes of the data for training models. This exacerbates the performance problem.
- Computationally intensive model fitting or simulations can cause slow execution.

5

Historically, data has to be “moved” before it can be analyzed. When the data set to be analyzed is large, moving data means that performance can be extremely slow. Exacerbating the problem, many data mining algorithms (e.g. logistic regression, neural networks) require multiple passes through the data. The SAS High-Performance solution minimizes data movement, substantially reducing computing time, especially for algorithms that require multiple passes through the data.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

How High-Performance Analytics Works



- All SAS High-Performance Analytics procedures are written using threaded kernel extensions.
- Each processing request is parsed into multiple parallel requests, and distributed to the appliance nodes.
- Appliance nodes operate independently.
- The results are consolidated and returned to the client.

6

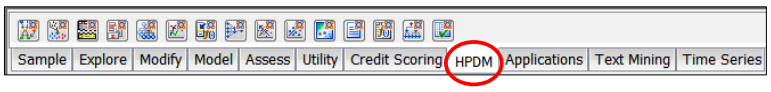
With the data distributed among the blades in the system, the SAS analytic program or application (client) sends instructions to a single blade (root node). The root node then sends the instructions for processing to all of the other blades in the system (worker nodes). Each node retrieves the data that it needs from the local hard drive and stores it in local memory. The analytic processes are then run in parallel on all of the data in the system. It runs high-performance-enabled capabilities from the following SAS solutions:

- Base SAS and SAS/STAT (for high-performance data summarization and statistical analysis)
- SAS/ETS (for high-performance econometrics and time series forecasting)
- SAS/OR (for high-performance optimization)
- SAS Enterprise Miner (for high-performance data mining)
- SAS Text Miner (for high-performance text mining)

Results are passed back through the root node to the client, and each node's memory is released. Each node retrieves only the data that it needs to perform the analysis and keeps it in memory only as long as it needs to perform the analysis.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

High-Performance Data Mining Nodes



- HP Bayesian Network Classifier
- HP Cluster
- HP Data Partition
- HP Explore
- HP Forest
- HP GLM
- HP Impute
- **HP Neural**
- HP Principal Components
- HP Regression
- HP SVM
- HP Text Miner
- HP Transform
- HP Tree
- HP Variable Selection

7

The HP nodes are found under the HPDM (High-Performance Data Mining) tab in the SEMMA tools palette. The available HP nodes, along with a brief summary, are listed below.

- HP Bayesian Network Classifier — creates a directed, acyclic graphical model in which the nodes represent random variables, and the links between the nodes represent conditional dependency between two random variables.
- HP Cluster — performs disjoint cluster analysis using Euclidean distances.
- HP Data Partition — generates an identifier variable identifying the observations to be used for training and for validation.
- HP Explore — computes summary statistics in the high-performance environment.
- HP Forest — creates random forest models in the high-performance environment.
- HP GLM — generates a generalized linear model that allows the population mean to depend on a linear predictor through a nonlinear link function (e.g. generalized or cumulative logit).
- HP Impute — imputes missing values using high-performance procedures.
- HP Neural — creates neural network models in the high-performance environment.
- HP Principle Components — calculates eigenvalues and eigenvectors from the covariance matrix or the correlation matrix of input variables.
- HP Regression — creates high-performance linear and logistic regression models
- HP SVM — implements Vapnik's (1995) support vector machine algorithm. It supports only binary classifications, including polynomial, radial basis function, and sigmoid nonlinear kernels.
- HP Text Miner — performs text mining in the high-performance environment.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

- HP Transform — creates transformation variables using high-performance procedures.
- HP Tree — enables you to create and visualize a tree model in the high-performance environment, and to determine input variable importance.
- HP Variable Selection — performs variable selection using high-performance procedures.

The icons for these nodes are the same as the non-high-performance nodes, except for the additional red icon in the upper right hand corner.

General Comments

- HPDM nodes are meant to supplement, not replace, their traditional counterpart nodes.
- The functionality of HPDM nodes will grow over time.

8

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

6.2 Overview of the HP Neural Node

Objectives

- Describe the HP Neural node's interface.

10

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

HP Neural Node

.. Property	Value
General	
Node ID	HPNNA
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Use Inverse Priors	No
Create Validation	No
Network Options	
Input Standardization	Range
Architecture	One Layer
Number of Hidden Neurons	3
Number of Hidden Layers	3
Hidden Layer Options	
Direct Connections	No
Target Standardization	Range
Target Activation Function	Identity
Target Error Function	Normal
Number of Tries	2
Maximum Iterations	300
Use Missing as Level	No
Report	
Maximum Number of Links	1000
Status	
Create Time	6/6/16 10:49 AM
Run ID	
Last Error	
Last Status	
Last Run Time	
Run Duration	
Grid Host	
User-Added Node	No

11

The HP Neural node was designed with two goals in mind:

- perform efficient, high-speed training of neural networks
- create accurate, generalizable models in an easy to use manner.

Thus, most settings (parameters) are selected automatically. This includes standardizing input and target variables, specifying the activation and error functions, and setting the termination criteria.

The HP Neural node requires one or more input variables, and one or more target variables. The inputs and targets can be binary, nominal, or interval**.** ***All ordinal variables are ignored during training.***

If an observation has missing values for any of the specified target variables, the observation is not used for training or for computing validation error. If your input data set contains a frequency variable, then the frequency variable must be an interval variable and all observations must be positive integers.

If you are running the HP Neural node in a grid environment, and using group processing, you cannot specify **Bagging** or **Boosting** for the Mode property of the Start Groups node.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

General Properties

.. Property	Value
General	
Node ID	HPNNA
Imported Data	
Exported Data	
Notes	

- The **Node ID** field identifies the current node.
- The **Imported Data** field specifies the set of tables imported by (read in) the HP Neural node.
- The **Exported Data** field identifies the tables exported (output) by the HP Neural node.
- The **Notes** field enables you to enter node-specific information or comments.

12

These general properties are associated with the HP Neural node:

- **Node ID** displays the ID that is assigned to the node. It is especially useful for distinguishing between nodes of the same type in a process flow diagram.
- **Imported Data** contains a list of the data sets that are imported by the node, and the ports that provide them. Clicking the ellipsis button to access the Imported Data window displays this list. If data exists, then you can select a row in the list and browse the data set, explore the data in a data set, or view the table and variable properties of a data set
- **Exported Data** contains a list of the data sets that are exported by the node and the ports to which they are provided. Clicking the ellipsis button to access the Exported Data window displays this list. If data exists, then you can select a row in the list and browse the data set, explore the data in a data set, or view the table and variable properties of a data set
- **Notes** accesses a window that you can use to store notes of interest. Click the ellipsis button to access the Notes window.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Train Properties

Train	
Variables	
Use Inverse Priors	No
Create Validation	No
Network Options	
Input Standardization	Range
Architecture	One Layer
Number of Hidden Neurons	3
Number of Hidden Layers	3
Hidden Layer Options	
Direct Connections	No
Target Standardization	Range
Target Activation Function	Identity
Target Error Function	Normal
Number of Tries	2
Maximum Iterations	300
Use Missing as Level	No

- The Train properties panel is the core of the HP Neural network construction process.
- It enables you to specify and configure the proposed neural network architecture.

13

These are the available train properties:

- **Variables** specifies the properties of each variable in the data source that you want to use. Click the ellipsis button to the right of the Variables property to open a variables table. You can set the variable status to either **Use** or **Don't Use** in the table, and you can set a variable's report status to **Yes** or **No**.
- **Use Inverse Priors** specifies whether inverse priors are used to weight training observations in the presence of a class target. This is especially helpful when you have rare events. When you specify **Yes**, a weight is calculated as the inverse of the fraction of time that the target class occurs in the input data set. It is applied to the prediction error of each nominal target variable. The default value for this property is **No**.
- **Create Validation** specifies whether a validation data set is created from the incoming training data set. When you specify **Yes**, every fourth observation is used to generate the validation data set. When you specify **No**, the entire data set is used for training.

  Training continues until the optimizer can no longer improve the total training error or the number of iterations specified in **Maximum Iterations** is reached. If a validation data set was created with the HP Data Partition node, validation is performed with that data set, and this property is ignored.
- **Input Standardization** specifies the method used to standardize the input variables. Permissible values are **Range**, **Z-Score**, and **None**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

- **Architecture** — The following network architectures are available:
  - **User-Defined**

    If you specify **User Defined**, then the **Number of Hidden Layers**, **Hidden Layer Options**, and **Direct Connections** properties become available.
  - **Logistic**

    Logistic regression, i.e. no hidden layers.
  - **One Layer**

    A single fully connected hidden layer.
  - **One Layer with Direct**

    A single fully connected hidden layer with direct connections, i.e. a skip-layer network.
  - **Two Layers**

    Two fully connected hidden layers.
  - **Two Layers with Direct**

    A fully connected two hidden layer network with direct connections between the input and output layers.
- **Number of Hidden Neurons** — specifies the number of hidden units in the network. If two hidden layers are specified, the number of hidden units is evenly distributed among the hidden layers. If the number of hidden neurons specified is odd, the extra hidden unit goes into the first hidden layer. The default value is 3.
- **Number of Hidden Layers** — specifies the number of hidden layers in the network. Up to 10 hidden layers can be specified.
- **Hidden Layer Options** — enables you to, by clicking the ellipsis (**…**), specify the number of hidden units in each layer (maximum=9999), and each hidden layer's activation function (i.e. tanh, sine, or identity).
- **Direct Connections** — specifies whether direct connections are to be generated. This applies only when a **User Defined** network is specified.
- **Target Standardization** — specifies the method used to standardize interval target variables. Permissible values are **Range**, **Z-Score**, and **None**.
- **Target Activation Function** — specifies the interval target variable activation function. The available functions are: Identity, Exponential, Sine, and Tanh.

  If the **Target Standardization** property is either **None** or **Z-Score**, then **Identity** must be specified.
- **Target Error Function** — specifies the error function to be used for interval targets. The available error function are **Normal**, **Poisson**, and **Gamma**.

  If either the **Poisson** or **Gamma** error functions are specified, the **Target Activation Function** must be **Exponentia**l. If the specified error function is **Normal**, then the **Identity**, **Sine**, or **Tanh** must be used.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

- **Number of Tries** specifies the number of times, or tries, that the network is to be retrained, using a different set of initial weights with each try. Because training involves optimizing a nonlinear objective function, this provides one way to be reasonably sure that a good set of weights is found. The default value is 2.
- **Maximum Iterations** specifies the maximum number of iterations allowed within each try. The default value is 50.
- **Use Missing as Level** specifies whether missing values should be considered as their own classification level.

Report Property

Report	
Maximum Number of Links	1000

- The **Report** property specifies the maximum number of links to include in the Link Graph or Weight plot.

14

There is only one report property available:

Maximum Number of Links specifies the maximum number of links to include in the Link Graph and Weights plots. (See note below.) When this number is exceeded, the HP Neural node keeps input links with the greatest absolute weight.

The Link Graph plot displays the network architecture graphically. All inputs are listed on the far left of the Link Graph. The target is at the far right of the Link Graph. Hidden units are found between the input variables and the target variable. The width and color of the lines indicate the magnitude of the weight for that particular connection. Thinner blue values indicate a smaller magnitude link weight, and thicker red values indicate a larger magnitude weight.

The weights plot displays the value of the weights for each connection. As in the Link Graph, the colors represent the magnitude of the weight.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Status Properties

Status	
Create Time	6/6/16 1:43 PM
Run ID	
Last Error	
Last Status	
Last Run Time	
Run Duration	
Grid Host	
User-Added Node	No

- The **Status** panel shows selective information about the last run status of the node.

15

These status properties are associated with the HP Neural node:

- **Create Time** displays the time at which the node was created.
- **Run ID** displays the identifier of the node run. A new identifier is created every time the node runs.
- **Last Error** displays the error message from the last run.
- **Last Status** displays the last reported status of the node.
- **Last Run Time** displays the time at which the node was last run.
- **Run Duration** displays the length of time of the last node run.
- **Grid Host** displays the grid server that was used during the node run.
- **User-Added Node** specifies whether the node was created by a user as a SAS Enterprise Miner extension node.

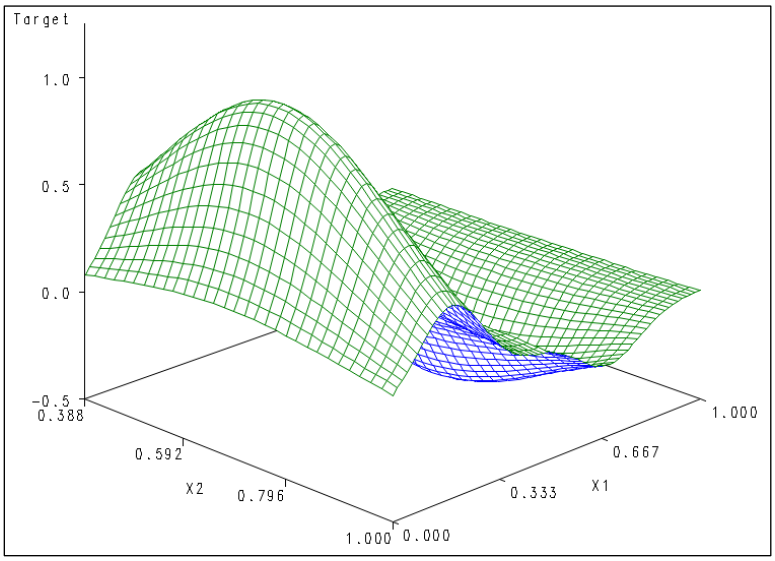
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Comparing One- and Two-Layer Networks

As was noted previously, for continuous input-output relationships, a one-hidden-layer neural network is a universal approximator. But, even when the relationship is continuous, it can be beneficial to consider using two-hidden-layer neural networks. A two-hidden-layer neural network will often produce a better fit to the data with fewer hidden units. The following demonstration illustrates this. It takes advantage of the fact that the HP Neural GUI enables you to construct two-hidden-layer networks.

This program requires access to the HPA nodes that are available only in SAS Enterprise Miner version 12 or later. To compare the performance of one- and two-hidden-layer networks in earlier versions of Enterprise Miner, a SAS Code node running PROC NEURAL code can be used in place of the HP Neural node. (See Page 4-7.)

The **hill_valley** data set is used. It embodies the following pattern:



The variables in the **hill_valley** data set are listed below.

Variable	Description	Role	Level
x1	First coordinate	Input	Interval
x2	Second coordinate	Input	Interval
y	Response value	Target	Interval

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **hidden layers** in the **Diagram Name** field, and click **OK** to save the change.
3. Right-click **Data Sources** in the project panel and select **Create Data Source**.
4. Click **Next** to create a SAS data table.
5. Select the **hill_valley** data table (in the **dmnn41** library). Click **OK**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

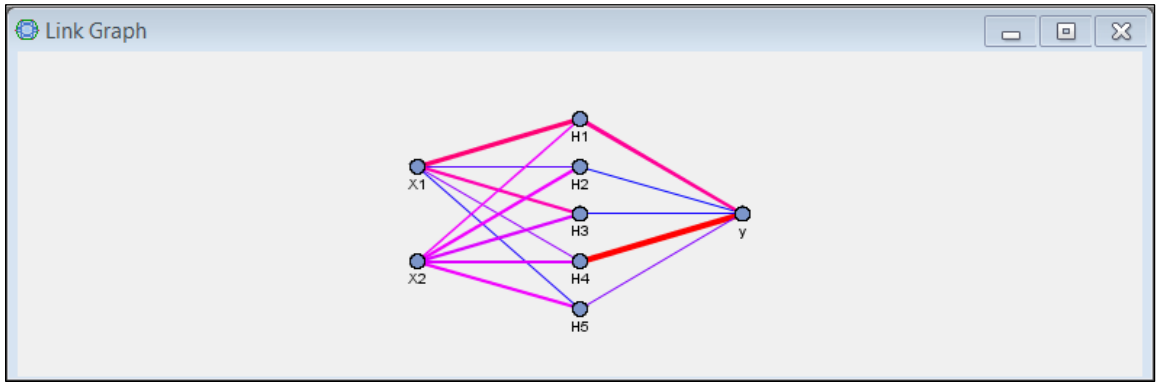
6. Click **Next** to display information about the **hill_valley** data. There are 2046 observations and 3 variables (listed previously) in the data set.
7. Click **Next** again to start the metadata definition process.
8. The **Basic** setting suffices in this case, so click **Next** to proceed.
9. Change the role of the **y** variable to **Target**.
10. Click **Next** three times, and then click **Finish** to complete the data source definition.
11. Drag and drop the newly created **hill_valley** data source onto the diagram.
12. Add an **HP Neural** node (eighth from the left in the HPDM tab), and connect it to the **hill_valley** data source.
13. Click the **HP Neural** node to activate its properties panel.
14. Change **Number of Hidden Neurons** to **5**. Also change **Number of Tries** to **5**. Finally, increase **Maximum Iterations** to **1000**. All other settings remain at their default values.

Train	
Variables	
Use Inverse Priors	No
Create Validation	No
Network Options	
Input Standardization	Range
Architecture	One Layer
Number of Hidden Neurons	5
Number of Hidden Layers	3
Hidden Layer Options	
Direct Connections	No
Target Standardization	Range
Target Activation Function	Identity
Target Error Function	Normal
Number of Tries	5
Maximum Iterations	1000
Use Missing as Level	No

15. Right-click the **HP Neural** node and select **Run** from the pop-up window.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

16. Examine the results when they are available. First examine the Link Graph plot in the upper right corner of the Results window .



The Link Graph confirms the structure of the requested network. There are two inputs, five hidden units in a single hidden layer, and one target unit. The width and color of the line indicate the resulting magnitude of the weight for that connection.

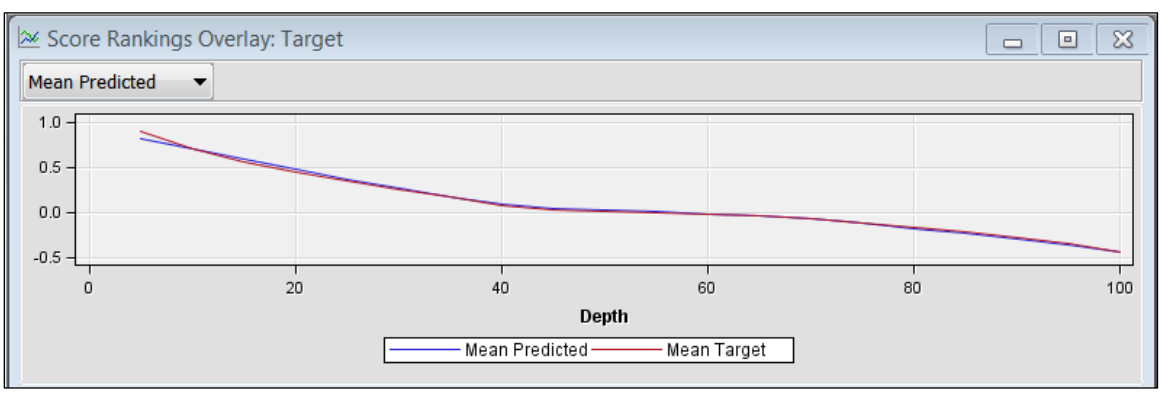
17. Next examine the Fit Statistics panel, located just below the Link Graph.

Fit Statistics

Target	Target Label	Fit Statistics	Statistics Label	Train	Validation	Test
y	Target	ASE	Average Squared Error	0.001703	.	.
y	Target	DIV	Divisor for ASE	2046	.	.
y	Target	MAX	Maximum Absolute Error	0.136712	.	.
y	Target	NOBS	Sum of Frequencies	2046	.	.
y	Target	RASE	Root Average Squared Error	0.041264	.	.
y	Target	SSE	Sum of Squared Errors	3.483809	.	.

It shows that the one-hidden-layer MLP network generated a sum of squared error value of 3.483809, and an average squared error value of 0.001703.

18. The Score Rankings Overlay plot displays the mean predicted values for the network.

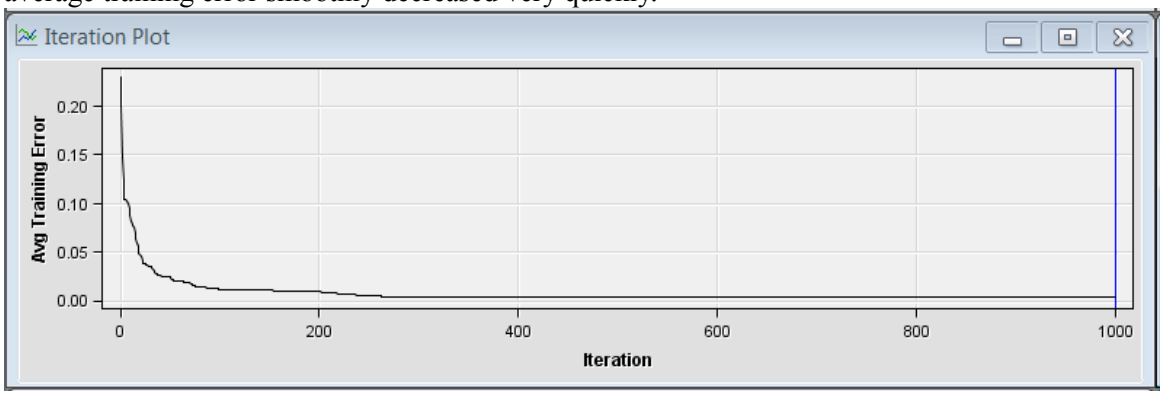


The mean predicted value seems to correspond quite closely to the actual target mean, across the entire depth of the network. The Max Predicted and Min Predicted plots (not shown), selected by clicking the down arrow, also reflect a fairly high degree of agreement with the actual values.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

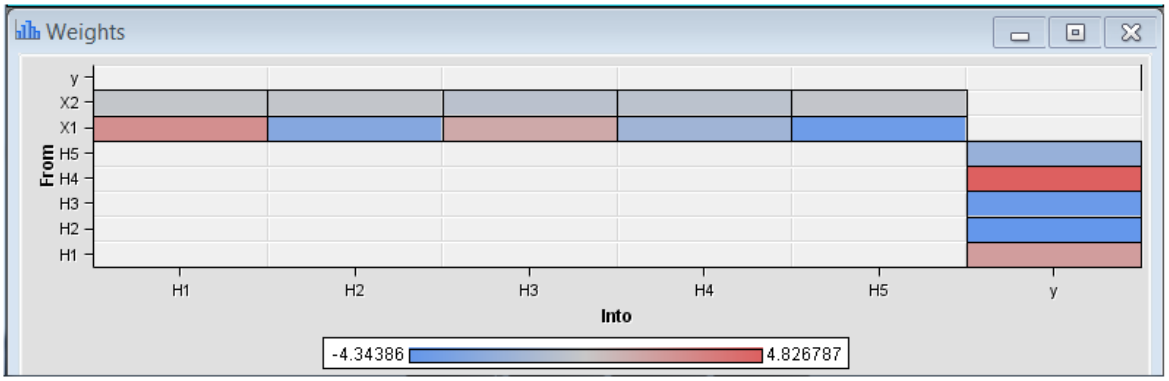
In a classification rule, *depth* is defined as the total proportion of cases that were allocated to class one, sorted into descending order. However, here the target is continuous. So in this case, *depth* means sort the observations by descending predicted value.

19. The Iteration plot, which is directly below the Score Rankings Overlay plot, shows that the average training error smoothly decreased very quickly.



The greatest reduction in training error occurred in the first 30 iterations. And there appeared to be little improvement in training error after iteration 100.

20. The final graphic is the Weights plot. It is a stylized version of a Hinton diagram that displays a color-coded report of the final weight value of each connection in the network.



21. The Output panel (bottom right corner) reports much of the same information that was presented graphically in the plots. It begins with a listing of the output predicted and decision variables.

```
Predicted and decision variables

Type          Variable    Label

TARGET          y         Target
PREDICTED       P_y       Predicted: y
RESIDUAL        R_y       Residual: y
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Next, the architecture is described. Scroll down to around line **35**.

```
The HPNEURAL Procedure

      Performance Information

Execution Mode       Single-Machine
Number of Threads    4


            Data Access Information

Data                Engine    Role     Path

EMWS31.IDS_DATA     V9        Input    On Client


                Model Information

Data Source                   EMWS31.IDS_DATA
Architecture                  MLP
Number of Input Variables     2
Number of Hidden Layers       1
Number of Hidden Neurons      5
Number of Target Variables    1
Number of Weights             21
Optimization Technique        Limited Memory BFGS
```

Notice that the optimization technique used is the Limited Memory BFGS algorithm. It is the only method available in HP Neural.

The Limited Memory BFGS (i.e., Broyden, Fletcher, Goldfarb and Shanno) algorithm approximates the BFGS using only a few vectors to approximate the inverse Hessian. Thus it is particularly well-suited to optimization problems with a large number of variables.

The number of observations read and used (which excludes observations with missing values) is then reported, as well as the observations' distribution across the training and validation data sets..

```
Number of Observations Read            2046
Number of Observations Used            2046
Number Used for Training               2046
Number Used for Validation                0
```

This is followed by a listing of the fit statistics (not shown), and the percent average and maximum error for the target variable, **y**.

```
    Error Summary for y

Description             Value

% Average Error     2.03922%
% Maximum Error     9.13581%
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Next the results of the (default) five requested preliminary training steps are displayed.

Training Table

Try	Iterations	Avg Training Error	Avg Validation Error	Reason for Stopping	Best?
1	1000	0.006017	.	MAXITER	
2	1000	0.003041	.	MAXITER	Y
3	1000	0.006098	.	MAXITER	
4	1000	0.006165	.	MAXITER	
5	1000	0.004030	.	MAXITER	

As indicated by the ‘Y’ under the ‘Best?’ column, in this case, the second starting point generated the smallest average training error (0.003041), after the 1000 requested iterations. The weights generated from the second location will be used as the initial weight values during training.

Next training begins. A partial listing of the training iterations is shown below. In all, 921 iterations were required before convergence was declared.

Iteration Table

Iteration	Avg Training Error	Avg Validation Error	Best?
0	0.230676	.	
1	0.201807	.	
2	0.147250	.	
3	0.124144	.	
4	0.104050	.	
5	0.103432	.	
6	0.102725	.	
7	0.102397	.	
8	0.099273	.	
9	0.094978	.	
10	0.088239	.	
11	0.081187	.	
12	0.078617	.	
13	0.076871	.	
14	0.075803	.	
15	0.069491	.	
...			
999	0.003042	.	
1000	0.003041	.	Y

Procedure timing information is listed next.

Procedure Task Timing

Task	Seconds	Percent
Data read and variable levelization	0.01	0.57%
Training	1.75	99.43%
Scoring	0.00	0.00%

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Next the Fit Statistics information is again displayed (to two decimal places), confirming that the model generated a sum of squared errors of 3.48 and an average squared error less than 0.01.

```
Fit Statistics

Target=y Target Label=Target

   Fit
Statistics    Statistics Label                Train

  _ASE_       Average Squared Error            0.00
  _DIV_       Divisor for ASE               2046.00
  _MAX_       Maximum Absolute Error           0.14
  _NOBS_      Sum of Frequencies            2046.00
  _RASE_      Root Average Squared Error       0.04
  _SSE_       Sum of Squared Errors            3.48
```

This is followed by the mean target and predicted mean by depth information, which is the basis of the Score Rankings Overlay plot.

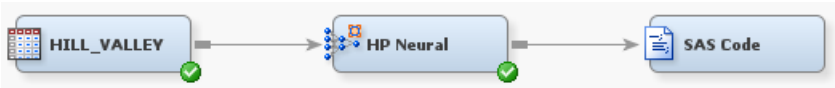
```
Assessment Score Rankings

Data Role=TRAIN Target Variable=y Target Label=Target

            Number of       Mean         Mean
Depth    Observations     Target     Predicted

   5          103         0.88761      0.81293
  10          102         0.70248      0.69698
  15          102         0.56065      0.59140
  20          103         0.45025      0.48086
  25          102         0.35277      0.37059
  30          102         0.24938      0.26250
  35          103         0.16588      0.17320
  40          102         0.07268      0.08853
  45          102         0.01716      0.04552
  50          102         0.00880      0.02407
  55          103        -0.00733      0.00146
  60          102        -0.01740     -0.02317
  65          102        -0.03382     -0.04485
  70          103        -0.07322     -0.08044
  75          102        -0.12198     -0.12580
  80          102        -0.16986     -0.17779
  85          103        -0.22023     -0.23449
  90          102        -0.28026     -0.29855
  95          102        -0.35435     -0.36985
 100          102        -0.44543     -0.44823
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The final information displayed in the Output window is the Assessment Score Distribution table.

```
Assessment Score Distribution

Data Role=TRAIN Target Variable=y Target Label=Target

   Range for          Mean         Mean        Number of       Model
   Predicted         Target     Predicted    Observations      Score

0.805 -  0.873       0.93385      0.83979          57         0.83908
0.737 -  0.805       0.81407      0.77288          57         0.77097
0.669 -  0.737       0.71345      0.70715          62         0.70285
0.601 -  0.669       0.62302      0.64011          65         0.63473
0.533 -  0.601       0.54292      0.57131          68         0.56661
0.464 -  0.533       0.46894      0.49844          67         0.49850
0.396 -  0.464       0.40249      0.42803          65         0.43038
0.328 -  0.396       0.33974      0.35765          61         0.36226
0.260 -  0.328       0.27607      0.29034          66         0.29414
0.192 -  0.260       0.21007      0.22244          77         0.22602
0.124 -  0.192       0.15246      0.15805          77         0.15791
0.056 -  0.124       0.06496      0.08346         109         0.08979
-0.012 -  0.056      0.00502      0.02176         299         0.02167
-0.081 -  -0.012    -0.03347     -0.04243         256        -0.04645
-0.149 -  -0.081    -0.11025     -0.11461         147        -0.11457
-0.217 -  -0.149    -0.17365     -0.18220         123        -0.18268
-0.285 -  -0.217    -0.23410     -0.25029         117        -0.25080
-0.353 -  -0.285    -0.30118     -0.31841          95        -0.31892
-0.421 -  -0.353    -0.37061     -0.38481          91        -0.38704
-0.489 -  -0.421    -0.45256     -0.45426          87        -0.45516
```

22. Close the HP Neural node's results.
23. Add a **SAS Code** node (Utility tab) and connect it to the **HP Neural** node. Your data flow diagram should look similar to the one shown below.



24. Activate the SAS Code node's properties panel, and click the ellipsis (**...**) next to **Code Editor**.
25. Right-click in the Training Code editor window, and select **Open**.
26. Read in the program **c6s1d1.sas** (in d:\workshop\winsas\dmnn41). The program simply displays the proposed solution (plot 1), and the residual (plot 2).

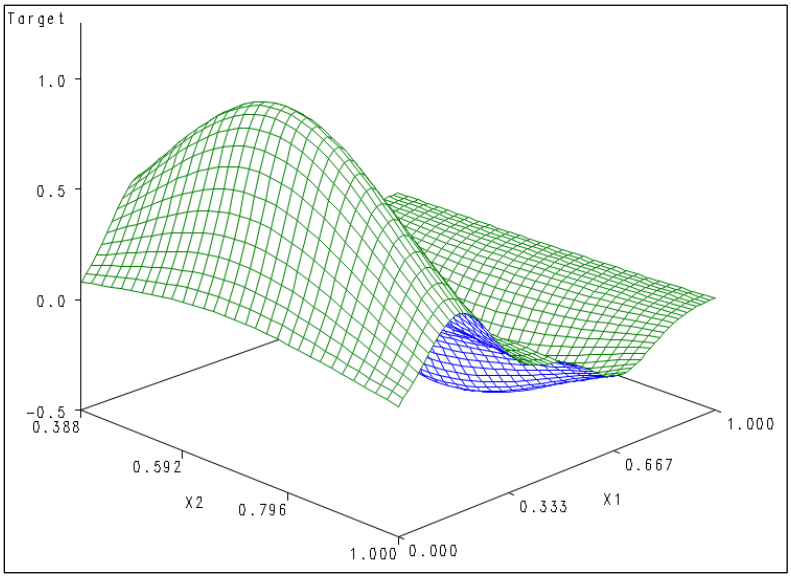
```
options nodate nonumber;
goptions reset=all device=actximg;

proc g3d data=&em_import_data;
   plot x1*x2=p_y/rotate=315 description="plot 1";
   plot x1*x2=r_y/rotate=315 description="plot 2";
run;
quit;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

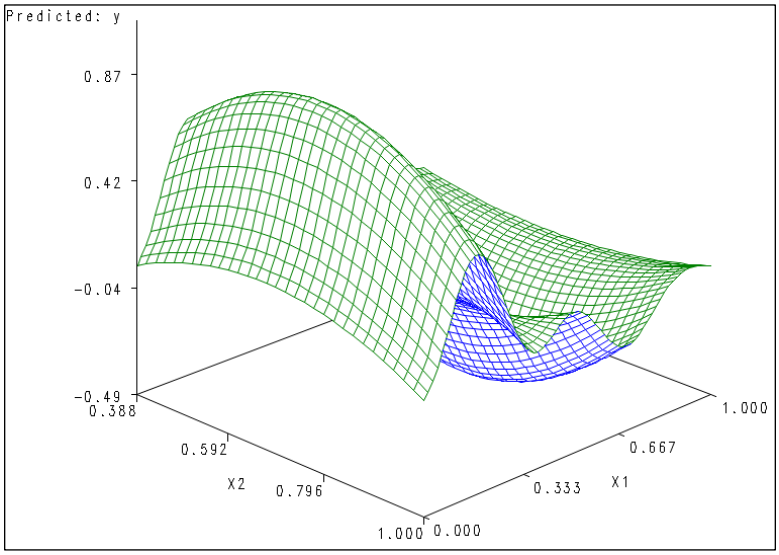
27. Run the flow from within the SAS Code node by clicking the running person icon at the top of the window. Confirm the run when asked, and open the results when they become available.

28. To see the plots, select **View** ⇨ **SAS Results** ⇨ **Train Graphs** at the top of the window.

    The first plot shows the actual target distribution.



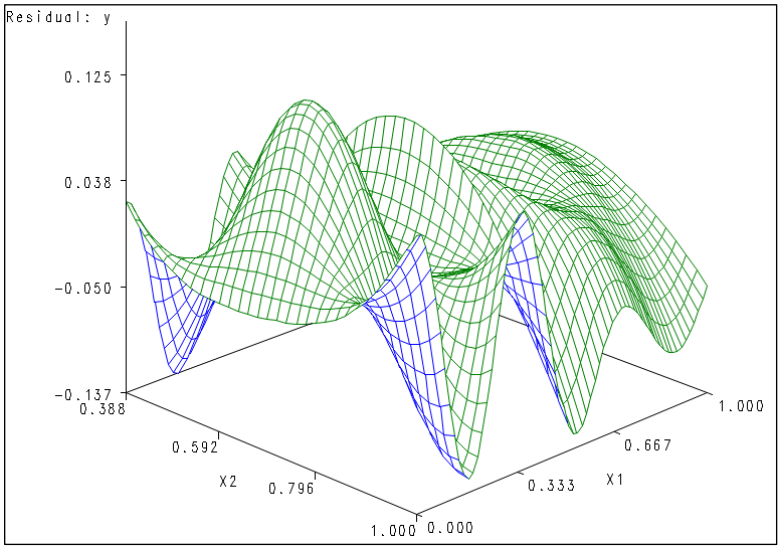
29. Click the right arrow (second button from the right) to see the residuals predicted distribution.



The single-hidden-layer network accurately captured at the hill and the adjacent valley. However, there is an extra hill (fold) that is not present in the original data.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

30. Click the right arrow again to see the residuals. The residuals, which varied greatly across the x1 and x2 values, ranged from 0.125 to -0.137.



31. Close the SAS Code node results.

32. Again activate the HP Neural node's properties panel, and change **Architecture** from **One Layer** to **Two Layers**. (See below.) No other changes are required.

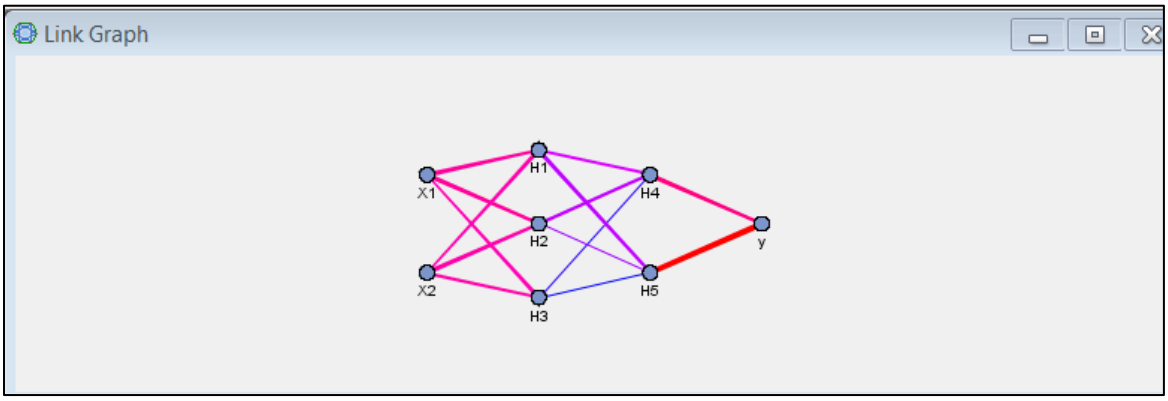
Train	
Variables	
Use Inverse Priors	No
Create Validation	No
Network Options	
Input Standardization	Range
Architecture	Two Layers
Number of Hidden Neurons	5
Number of Hidden Layers	3
Hidden Layer Options	
Direct Connections	No
Target Standardization	Range
Target Activation Function	Identity
Target Error Function	Normal
Number of Tries	5
Maximum Iterations	1000
Use Missing as Level	No

33. Right-click **HP Neural**, and re-run the flow from the HP Neural node.

34. Examine the results when they are available.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

35. Maximize the Link Graph (upper right corner) to confirm the requested architecture.



The five hidden units are now spread across the two hidden layers (i.e. three hidden units in the first hidden layer, and two hidden units in the second hidden layer).

36. Restore the Link Graph, and maximize the Fit Statistics panel.

Fit Statistics

Target	Target Label	Fit Statistics	Statistics Label	Train	Validation	Test
y	Target	ASE	Average Squar...	.0001414	.	.
y	Target	DIV	Divisor for ASE	2046	.	.
y	Target	MAX	Maximum Abs...	0.053493	.	.
y	Target	NOBS	Sum of Freque...	2046	.	.
y	Target	RASE	Root Average ...	0.01189	.	.
y	Target	SSE	Sum of Squar...	0.289247	.	.

The sum of squared errors value has dropped from 3.483809 in the one hidden-layer network to 0.289247, and the two hidden-layer average squared error is 0.0001414 (it was 0.001703), a factor of 10 decrease from the value generated by the one-hidden-layer network.

37. Close the HP Neural node's results.

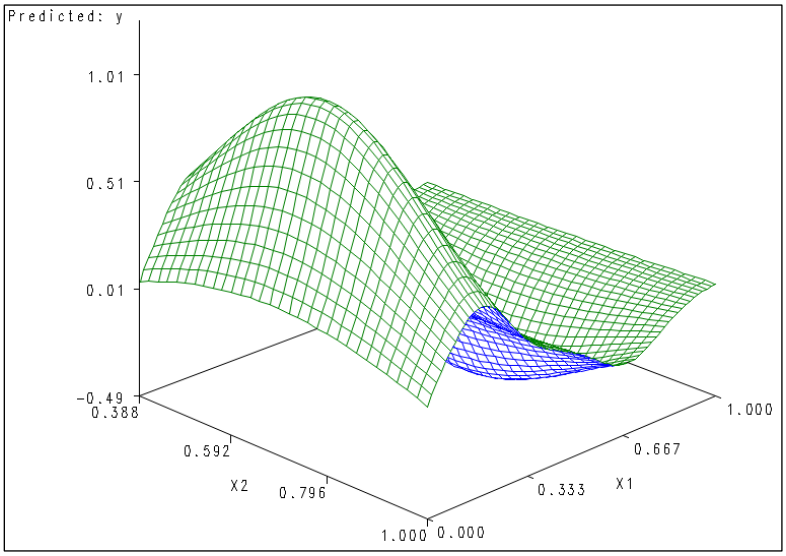
38. Right-click the **SAS Code** node, and select **Run** from the pop-up menu.

39. Examine the results when they become available.

40. To see the plots, select **View** ⇨ **SAS Results** ⇨ **Train Graphs** at the top of the window. The plot of the actual target distribution is displayed first (not shown).

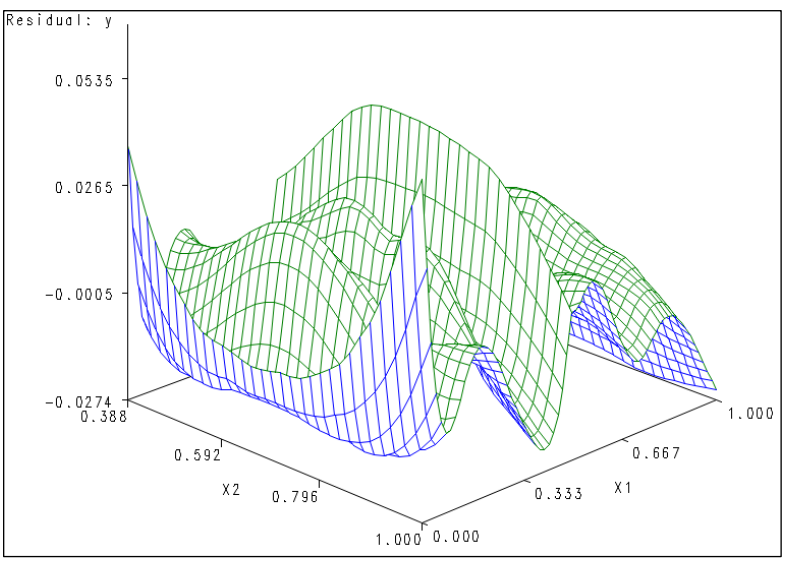
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

41. Click the right arrow (second button from the right) to see the residuals predicted distribution.



As the fit statistics suggested, the two-hidden-layer network seems to almost perfectly model the **hill_valley** data. The "fold" that was present in the one-hidden-layer network's predicted values plot has completely disappeared. ***This was achieved with no increase in the number of hidden units used, or in the number of training iterations requested!***

42. Click the right arrow again to see the residuals.



The residuals, which again varied greatly across the x1 and x2 values, seem to have improved little over the one layer network results. But in fact they are not as severe as they were when a single hidden layer was used. The now range only from 0.0535 to -0.0274.

43. Close the SAS Code node results window.

44. Close the "hidden layers" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

6.01 Poll

The HP Neural node is a multithreaded (parallel) version of the Neural Network node.

- ❍ True
- ❍ False

17

6.3 Chapter Summary

Big data is a popular term used to describe the exponential growth, availability, and use of information, both structured and unstructured. Many organizations are concerned that the amount of amassed data is becoming so large that it is difficult to find the most valuable pieces of information. Now it is possible to apply high-performance analytics to analyze the massive amounts of data using technologies such as grid computing, in-database processing, and in-memory analytics. These big data technologies provide the ability to understand large amounts of data, and to use it for optimized decision making.

The HP nodes are found under the HPDM (High-Performance Data Mining) tab in the SEMMA tools palette. The available HP nodes are: HP Data Partition, HP Explore, HP Forest, HP Impute, HP Neural HP Regression, HP Text Miner, HP Transform, and HP Variable Selection.

The HP Neural node was designed with two goals in mind:

- The HP Neural node aims to perform efficient, high-speed training of neural networks.
- The HP Neural node attempts to create accurate, generalizable models in an easy-to-use manner.

Most parameters in HP Neural are selected automatically. This means that the neural network modeler has much less control when using the HP Neural interface. However, HP Neural is meant to supplement, not replace, the existing NEURAL procedure.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

6.4 Solutions

Solutions to Student Activities (Polls and Quizzes)

6.01 Poll – Correct Answer

The HP Neural node is a multithreaded (parallel) version of the Neural Network node.

- ❍ True
- ❍ False

18

The HP Neural node has markedly different functionality than does the Neural Network node. In the HP Neural node, many more of the parameters that you are allowed to alter in the Neural Network node are set automatically. Moreover, some of the features of the HP Neural node have been specifically selected to facilitate parallel processing (e.g. limited-memory BFGS optimization).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Chapter 7 The AutoNeural Node (Self-Study)

7.1 Introduction **7-3**

7.2 Architectures **7-12**

Demonstration: Comparing AutoNeural Architectures 7-18

7.3 Chapter Summary **7-23**

7.4 Solutions **7-24**

Solutions to Student Activities (Polls/Quizzes) 7-24

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

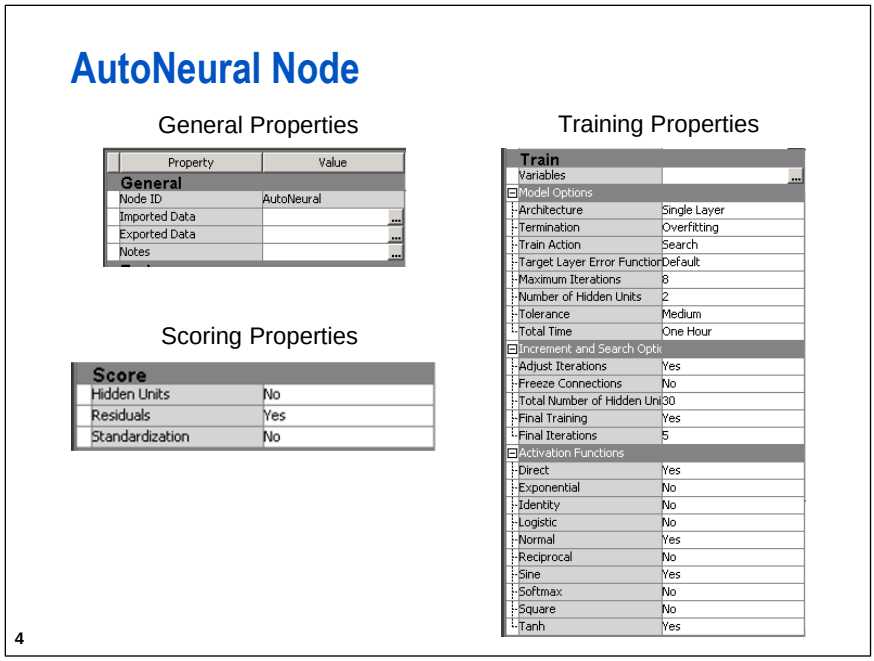
7.1 Introduction

Objectives

- Detail the options available on the AutoNeural node's properties panel.

3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



The AutoNeural node can be used to automatically construct a neural network. The code that it generates can also serve as the starting point for the development of custom neural network architectures.

Here are some of the key features of the AutoNeural node:

- Multiple hidden layer MLP networks can be constructed. Hidden layers are added one at a time.
- Preliminary training is optionally performed.
- All specified combination and activation functions are tried.
- Incremental training. (Subsequent training is initialized with the current weight estimates.)
- Limited searches for the best network configuration.
- Complete control over the target layer error function is applied.
- Layers can be frozen during subsequent training.
- In each training iteration, one estimate vector and one fit vector are retained.

The AutoNeural properties panel is divided into sections. First is the ***General*** properties section, which contains the following information:

Node ID	The first AutoNeural node added to a diagram will have a Node ID value of AutoNeural. The second AutoNeural node added will have a Node ID of AutoNeural2, and so on.
Imported Data	The Imported Data - AutoNeural window contains a list of the ports that provide data sources to the AutoNeural node.
Exported Data	The Exported Data - AutoNeural window contains a list of the output data ports that the AutoNeural node creates data for when it runs.
Notes	Select the button to the right of the Notes property to open a window that you can use to store notes of interest, such as data or configuration information.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Next is the ***Train*** properties section of the AutoNeural node's properties panel. It has three subsections:

(a) Model Options

The Model Options subsection enables you to configure the general properties of your network. The available settings are described below.

Architecture	specifies the neural network architecture that you want to use to build the network. The default is the single layer architecture.
Termination	identifies the method that you want to use to end training. The default is to stop when overfitting is detected.
Train Action	specifies the training method that you want to use. The default is to use all selected activation functions until the stopping criterion is achieved.
Target Layer Error Function	specifies the error function in a user-defined network. For discrete target variables, the default error function is multiple Bernoulli. For interval targets, the default error function is a normal distribution.
Maximum Iterations	specifies the maximum number of iterations permitted during training. Permissible values are any integer between 1 and 50. The default is 8.
Number of Hidden Units	specifies the number of hidden units that you want to use. Permissible values are integers between 1 and 8. The default is 2.
Tolerance	configures the extent of the preliminary search. The default is Medium, which invokes preliminary training but does not apply the ABSCONV=.001 convergence criteria.
Total Time	specifies the amount of time allowed for training. The default is one hour.

(b) Increment and Search

When the Search Train option is specified, nodes are added according to the architecture. The best network is retained as the final model. The following options are available:

Adjust Iterations	If the Train Action option is set to Search or Increment, and this option is set to Yes (default), the Maximum Iterations value is adjusted higher if the selected iteration equals the previously used Maximum Iterations value.
Freeze Connections	This option specifies whether connections (weights) should be frozen. The default setting for the Freeze Connections property is No.
Total Number of Hidden Units	This option specifies the hidden unit ceiling when Train Action is set to Search. Permissible values are 5, 10, 20, 30 (default), 40, 50, 75, or 100.
Final Training	This option indicates whether the final model should be trained again to allow the model to converge. If this is set to Yes, the number of iterations used will correspond to the value set in the Final Iterations property.
Final Iterations	This option indicates the number of iterations to use if the Final Training option is set to Yes. The Final Iterations property is unavailable if the Final Training option is set to No.

(c) Activation Functions

Activation functions alter and constrain a neuron's output range. By default, only the Direct, Normal, Sine, and Tanh activation functions are set to Yes (active).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Direct	Use the Direct activation function during training.
Exponential	Use the Exponential activation function during training if you want to constrain the output range of a neuron to nonnegative values (0:∞)
Identity	Use the Identity (no transformation) activation function during training when you do not want to constrain the neurons activation range (-∞:∞).
Logistic	Use the Logistic activation (logit-link) function during training. The Logistic activation function returns the probability (0:1) of each target class.
Normal	Use the Normal (Gaussian) activation function during training. Its range is infinite (-∞:∞) although, in practical terms, it ranges from about -4.5 to 4.5.
Reciprocal	Use the Reciprocal activation function, ranging from -∞ to ∞, during training.
Sine	Use the Sine activation function during training. Its range is -1 to 1.
Softmax	Use the Softmax activation function (inverse generalized logit link). The Softmax activation function, described earlier, generates probability values (0:1) for each target class. It further ensures that the probabilities across the target classes sums to 1.
Square	Use the Square activation function during training. Like the exponential activation function, the square function (which squares its input) generates nonnegative values.
Tanh	Use the Hyperbolic Tangent activation function during training. The hyperbolic tangent is a sigmoidal activation function (like logistic) that ranges between -1 and 1.

The final section in the AutoNeural panel is the ***Score*** properties section, which enables you to specify the outputs to be generated. The available options are as follows:

Hidden Units	specifies whether you want to create hidden unit variables (default is No).
Residuals	specifies whether you want to create residual variables (default is Yes).
Standardization	specifies whether you want to create standardization variables (default is No).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Available Train Options

You can choose from the following training actions:

- **Train**: Selected functions are trained until stopping.
- **Increment**: Nodes are added one at a time. It is retained only if a layer lowers the average error.
- **Search**: Nodes are added according to the specified architecture. The best network is retained. (This is the default.)

5

The *Train Action* option in the Train section of the AutoNeural properties panel specifies the method that you want to use to train the network. There are three options:

1. If **Train** is chosen, the specified hidden units are trained using all selected functions until stopping. There is no search. As a result, this option usually runs the fastest.
2. The **Increment** option adds nodes one at a time from each of the specified activation functions. If the added node lowers the average error, it is retained. If not, it is dropped. No function is reused.
3. When the **Search** (default) training action is selected, nodes are added according to the specified architecture. If there are hidden units, preliminary training is performed. Only the network generating the best fit performance is retained, where *best performance* is defined as per the following table:

Target Level	Validation Data?	Selection Criteria
Continuous	No	Schwarz-Bayesian Criterion (_SBC_)
	Yes	Validation Average Squared Error (_VAVERR_)
Discrete	No	Training Misclassification Rate (_MISC_)
	Yes	Validation Misclassification Rate (_VMISC_)

When **Search** is selected, the total number of hidden units is calculated on each run. If the value is greater than the specified (in the Total Number of Hidden Units property) number of hidden units, then training stops and the final model is selected. Due to the extra steps involved in conducting a search for the best performing network, the Search option usually runs the slowest.

Although AutoNeural's predictions are adjusted for priors, the model selection process is based only on the fit metric used, ***without prior adjustment***. This could lead to unexpected results if the primary and secondary outcome proportions are not equal.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Tolerance Options

Tolerance options configure the extent of the preliminary search.

- **Low** requires no preliminary training.
- **Medium** performs preliminary training (default).
- **High** performs preliminary training and applies a convergence criteria (ABSCONV=0.001).

6

The Tolerance option is used to configure preliminary training.

Specifying **Low** for the Tolerance option turns off preliminary training altogether. Because preliminary training helps avoid the worst local minima, this option should probably be used sparingly.

A tolerance level of **Medium** (default) or **High** requests that preliminary training be performed. If **High** is specified, the algorithm also applies the ABSCONV < 0.001 convergence criteria.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Error Functions

- Normal (default for interval targets)
- Cauchy
- Logistic
- Huber
- Biweight
- Wave
- Gamma
- Poisson
- Bernoulli distribution
- Entropy
- MBernoulli (default for discrete targets)
- Multinomial
- Mentropy

7

In addition to fitting normally distributed targets, the AutoNeural node offers a rich set of error (deviance) functions that enable you to fit non-normal targets directly, ***without the need for target transformation***. The available error functions are the same set as previously defined for the Neural Network node:

Normal	used with any type of target variable to predict the conditional mean. It is most often used with interval targets for which the noise distribution is approximately normal with constant variance. With categorical targets it is used for robust estimation of posterior probabilities.
Cauchy	used with any type of target variable. However, it is most often used when you want to predict the approximate conditional mode instead of the conditional mean.
Logistic	used with any type of target variable, but it is most often used with interval targets where the noise distribution might contain outliers.
Huber	suitable for unbounded interval targets that have outliers. It can also be used for categorical targets to predict the mode rather than the posterior probability.
Biweight	most often used with interval targets where the noise distribution might contain severe outliers although it can be used with any type of target variable.
Wave	used with any type of target variable, but it is most often used with interval targets where the noise distribution might contain severe outliers.
Gamma	most often used when the standard deviation of the noise is proportional to the mean of the target. It can be used only with positive interval target variables.
Poisson	suitable for skewed, nonnegative interval targets, especially counts of rare events, where the conditional variance is proportional to the conditional mean.
Bernoulli	only suitable for a target with the values 0 and 1.
Entropy	for independent interval targets with values between 0 and 1 inclusive.
Mbernoulli	suitable for all categorical targets, including multinomial.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Multinomial can be used only with two or more interval target variables with nonnegative values, where each case represents a number of trials that are equal to the sum of the target values. The activation function must force the outputs to sum to 1 (e.g., softmax).

Mentropy should normally be used only with two or more interval targets with values that lie between 0 and 1 inclusive, and that sum to 1 for each case. The activation function must force the outputs to sum to 1 (e.g., softmax).

For a categorical target, the default error function is multiple Bernoulli (MBernoulli). For interval targets, the default error function assumes that the target is normally distributed (Normal).

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Termination Criteria

Training automatically terminates when the following occurs:

- **Overfitting** is detected (default).
- Maximum **training time** has elapsed.
- Reduction in **training error** is < 0.001.

8

Training always stops when the maximum run time is exceeded. The default termination criterion is **Overfitting**, which, as its name implies, stops training when overfitting is detected.

Overfitting can be falsely detected for some models. Therefore, two additional options are available. If **Training Error** is specified, training will stop when the reduction in error is less than 0.001.

Specifying **Training Time** causes the models to be trained until the allotted time has elapsed.

7.01 Poll

The AutoNeural node can build neural networks with more than one hidden layer.

- ❍ True
- ❍ False

9

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

7.2 Architectures

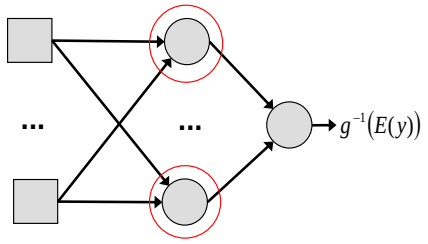
Objectives

- Describe the four AutoNeural architectures:
  - Single Layer
  - Block Layers
  - Funnel Layers
  - Cascade

12

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Single-Layer Architecture



- Hidden layers/neurons are added in parallel.
- Direct connections (not shown) might also be present.
- The algorithm chooses the number of hidden units and the type of activation function.

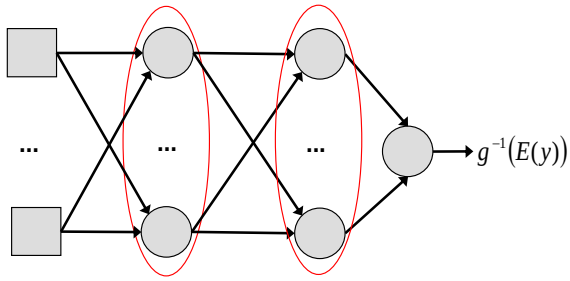
13

The Architecture property is used to specify the form of the neural network that you want to construct. Four predefined architectures are available. By default, a *single layer* architecture is created, in which ***the specified number of hidden units is added in parallel to the existing hidden layer***.

Setting the Number of Hidden Units property to 1 builds the network one neuron at a time. If the specified value is greater than 1, groups of neurons are added.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Block Layers Architecture



- Hidden layers/neurons are added as additional layers.
- Each layer has the same number of neurons.
- Direct connections (not shown) might also be present.
- AutoNeural chooses number of the hidden units and the type of activation function.

14

In the *block layers* architecture, ***hidden units are added as additional layers***. Each layer that is added has exactly the same number of hidden units.

The number of hidden units added with each layer is given by the value specified for the Number of Hidden Units property.

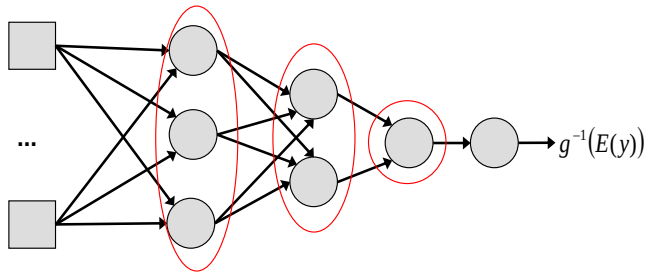
Adding more hidden layers creates new levels of nesting, rather like combining separate MLP networks. Although a single-hidden-layer MLP models any continuous input-output relationship, additional hidden layers might be required when the relationship is discontinuous. Also, somewhat counterintuitively, the addition of an extra hidden layer can actually often help speed up convergence on a solution.

Despite these advantages, the block layers architecture is ***not*** commonly used. A key reason for its lack of popularity is that the number of weights grows quickly with the number of hidden layers. Specifically, assuming that there are k inputs, and that h_1 and h_2 are the number of units in the first and second hidden layers, respectively, then the number of weights that must be set is given by the following equation:

$$h_1(k+1)+h_2(h_1+1)+h_2+1$$

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Funnel Layers Architecture



- The layers form a funnel. Only a single node feeds into the output layer.
- Direct connections (not shown) might also be present.
- The algorithm chooses the number of hidden neurons and type of activation function.

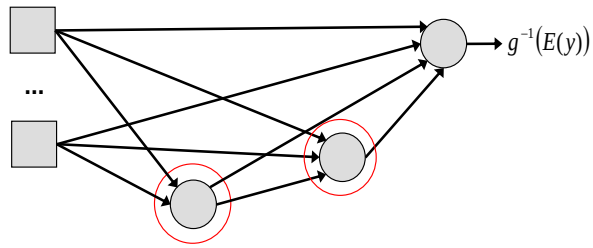
15

As in the block layers architecture, in the *funnel layers* architecture, multiple hidden layers are created. However, ***each time a new layer is added, a single neuron is added to the previous layer***. This forms a funnel pattern, and a single neuron feeds into the output layer. (See diagram above.)

The funnel layers architecture is also rarely used.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Cascade Architecture



- Hidden nodes are added in a cascading fashion.
- All trained weights are frozen.
- The algorithm chooses the number of hidden neurons and type of activation function.

16

The *cascade* architecture (Fahlman and Lebiere 1990, Prechelt 1997) is a multilayer perceptron in which ***each hidden unit is connected to all preceding input and hidden units in a cascading fashion***, giving the algorithm its name. For example, the output of a two-input (x_1, x_2), two-hidden-unit (H_1, H_2) network that uses the hyperbolic tangent (tanh) activation function is given by this equation:

$$g_0^{-1}(E(y)) = w_0 + w_1x_1 + w_2x_2 + w_3H_1 + w_4H_2$$

where

$$H_1 = \tanh(w_{01} + w_{11}x_1 + w_{21}x_2)$$

$$H_2 = \tanh(w_{02} + w_{12}x_1 + w_{22}x_2 + w_{32}H_1)$$

Notice that hidden unit H_1 is included in the input set to hidden unit H_2. If a third hidden unit (H_3) is added, its activation would be given by this equation:

$$H_3 = \tanh(w_{03} + w_{13}x_1 + w_{23}x_2 + w_{32}H_1 + w_{43}H_2)$$

And the corresponding output equation would be as follows:

$$g_0^{-1}(E(y)) = w_0 + w_1x_1 + w_2x_2 + w_3H_1 + w_4H_2 + w_5H_3$$

Regardless of the Number of Hidden Units value that is specified, **the Cascade architecture always constructs its networks one hidden unit at a time**.

Details

The Cascade network construction methodology is described below.

1. Fit a generalized linear model.
2. Add a hidden unit and connect it both to all input units and to all of the previous hidden units. This, in effect, turns all previous hidden units into inputs to the new hidden unit.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

3. To initialize the new hidden unit to reasonable parameter values, freeze all of the weights and biases (except for the output layer biases) in the old network at their present value, and train.
4. Thaw the frozen parameters, allowing all of the weights to again change, and then re-train the entire network. This integrates the new hidden unit's weights with the rest of the network.
5. If convergence has not been achieved, then return to step 2.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Comparing AutoNeural Architectures

c7s2d1

This demonstration explores the performance of the AutoNeural node's single layer, block layers, funnel layers, and cascade (correlation) architectures. It compares the impact of changing the Train Options setting within each architecture from **Train** to **Increment**, and then to **Search**. The architectures are assessed against the **TITANIC** data table, described in Chapter 2.

1. Right-click **Diagrams** in the project panel and select **Create Diagram**.
2. Enter **autoneural** in the **Diagram Name** field, and click **OK**.
3. Because the **TITANIC** data source has already been defined, just drag and drop it onto the diagram.

Single Layer Architecture (Default)

4. Add an **AutoNeural** node (leftmost node in the Model tab) to the diagram.
5. Connect it to the **TITANIC** data source.

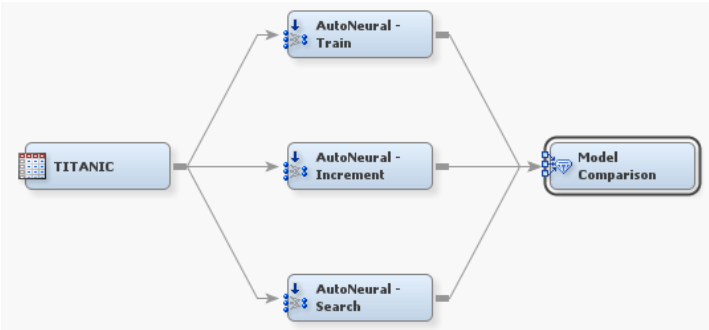
6. Activate the AutoNeural node's properties panel, and make the following changes:
   a. Change the Train Action property to **Train**.
   b. Set the Maximum Iterations property (Model Options section) to **50**, which is the maximum allowed. This means that, in total, up to 50 groups of hidden units could be added.
   c. Set the Number of Hidden Units property (Model Options section) to **1**. This means that "groups" of only one hidden unit will be added to the specified architecture on each iteration.

      AutoNeural always builds its cascade networks one hidden unit at a time, regardless of the value specified for the Number of Hidden Units property.

   d. Change the Tolerance value (Model Options section) to **High**. This will not only perform preliminary training, it also applies the ABSCONV=0.001 convergence criteria.
   e. Because the addition of a skip layer seldom helps improve a neural network's fit, under the Activation Functions section of the properties panel, set the Direct property to **No**.
   f. In order to simplify the subsequent discussion, also set Sine to **No**. Only the Tanh (hyperbolic tangent sigmoid) and the Normal (Gaussian) activation functions will be used.
7. Right-click the **AutoNeural** node again and select **Rename** from the pop-up menu. Rename the node **AutoNeural - Train**.
8. Right-click the **AutoNeural** node and select **Copy** from the pop-up menu.
9. Right-click anywhere in the diagram white space and select **Paste** from the pop-up menu.
10. Connect the new node to the input data source.
11. Rename the copied node **AutoNeural - Increment**.
12. Change its Train Action value to **Increment**.
13. Add another copy by again right-clicking in the diagram white space and selecting **Paste**.
14. Connect the new node to the input data source.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

15. Right-click the new node, and rename it **AutoNeural - Search**.
16. Change its Train Action value to **Search**.
17. Lastly, under the Assess tab, add a **Model Comparison** node to the diagram. The completed data flow diagram is shown below.



18. Run the diagram from the Model Comparison node, and open the results when asked.
19. Maximize the Output panel (lower right corner). The fit statistics appear at line **29**.

Fit Statistics

Model Selection based on Train: Misclassification Rate (_MISC_)

Selected Model	Model Node	Model Description	Train: Misclassification Rate	Train: Average Squared Error	Train: Roc Index	Train: Gini Coefficient
Y	AutoNeural3	AutoNeural - Search	0.19404	0.14523	0.839	0.678
	AutoNeural	AutoNeural - Train	0.20932	0.15426	0.819	0.637
	AutoNeural2	AutoNeural - Increment	0.21008	0.15503	0.815	0.630

The Search variant of the Single Layer architecture generated the lowest misclassification rate and average squared error. It also had the largest ROC Index and Gini coefficient.

20. Close the Multiple Comparison node Results window, and return to the diagram.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Block Layer Architecture

21. Activate the **AutoNeural - Train** node's properties panel, and change the Architecture property to **Block Layers**. ***Make the same change in the other two AutoNeural nodes.***
22. Re-run the diagram from the Model Comparison node, and examine the results when they are ready.
23. Maximize the Output panel, and examine the performance information at line **29** (shown below).

```
Fit Statistics
Model Selection based on Train: Misclassification Rate (_MISC_)

                                                                  Train:
                                                     Train:      Average  Train:
Selected                                        Misclassification Squared    Roc   Train: Gini
 Model    Model Node     Model Description           Rate         Error    Index  Coefficient

   Y      AutoNeural3  AutoNeural - Search         0.22002       0.15961   0.798     0.596
          AutoNeural2  AutoNeural - Increment      0.22154       0.15966   0.798     0.596
          AutoNeural   AutoNeural - Train          0.22231       0.15968   0.798     0.596
```

When the block layers architecture is specified, the Search variant again outperformed both the Train and Increment options with respect to misclassification rate and average squared error. However, the Train option had the largest ROC Index and Gini Coefficient values. In all cases, the difference in the goodness-of-fit scores for the three training actions (i.e. Train, Increment, and Search) was minimal

24. Close the results window and return to the diagram.

Funnel Layer Architecture

25. In the AutoNeural - Train node's properties panel, change the Architecture property to **Funnel Layers**. Make the same change in the other two AutoNeural nodes.
26. Re-run the diagram from the Model Comparison node, and examine the results when selection is complete. (The Search variant might take some time to finish.)
27. As before, maximize the Output panel and examine the fit statistics at line **29** (reproduced below).

```
Fit Statistics
Model Selection based on Train: Misclassification Rate (_MISC_)

                                                                  Train:
                                                     Train:      Average  Train:
Selected                                        Misclassification Squared    Roc   Train:
Gini
 Model    Model Node     Model Description           Rate         Error    Index
Coefficient

   Y      AutoNeural3  AutoNeural - Search         0.19939       0.15155   0.825     0.651
          AutoNeural   AutoNeural - Train          0.20397       0.15088   0.825     0.649
          AutoNeural2  AutoNeural - Increment      0.20550       0.15239   0.811     0.623
```

Again the Search variant outperformed the Train and Increment variants on most of the measures. The Train did tie Search on the ROC Index, and it had the lowest average squared error of the three. As in the Block Layer Architecture, the difference in the fit of the three training actions was minimal.

28. Close the Results window and return to the diagram.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Cascade Architecture

29. Lastly, in the AutoNeural - Train node's properties panel, change the Architecture property to **Cascade**. Make the same change in the other two AutoNeural nodes.
30. Rerun the diagram from the Model Comparison node, and examine the results.
31. Maximize the Output panel and examine the fit statistics at line **29** (reproduced below).

Fit Statistics

Model Selection based on Train: Misclassification Rate (_MISC_)

Selected Model	Model Node	Model Description	Train: Misclassification Rate	Train: Average Squared Error	Train: Roc Index	Train: Gini Coefficient
Y	AutoNeural3	AutoNeural - Search	0.15966	0.12135	0.889	0.777
	AutoNeural	AutoNeural - Train	0.20474	0.15229	0.823	0.645
	AutoNeural2	AutoNeural - Increment	0.22078	0.15784	0.810	0.619

Again the search option generated the best model, as defined by the smallest misclassification rate on the training data. In fact, in all four architectures, the search train option generated the smallest misclassification rate. And in three of the four architectures, it also has the smallest average squared error value, and the largest ROC index and Gini coefficients. Thus, the "Search" option would seem to be the best choice**.** The cascade architecture's Search variant had the smallest misclassification rate overall. It also had the smallest average squared error and the largest ROC index and Gini values.

32. Close the Results window and return to the diagram.
33. Close the AutoNeural node's Results window.
34. Close the "autoneural" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

7.02 Multiple Choice Poll

Which of the following use multiple hidden layers?

a. the block layers architecture

b. the funnel layers architecture

c. the cascade architecture

d. all of the above

18

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

7.3 Chapter Summary

The AutoNeural node provides a convenient way to automatically determine an appropriate neural network architecture for a given data table. Four predefined architectures are available:

- By default a single (hidden) layer is created, in which the hidden units are added in parallel.
- If the block layer architecture is specified, hidden layers are added as additional layers. Thus, the block architecture enables you to construct multiple hidden-layer networks.
- In a funnel architecture, the output layer has only one unit, the preceding layer has two hidden units, the layer before that has three hidden units, and so on.
- In the cascade architecture, each new layer (a single hidden unit because units are always added one at a time) is connected to all preceding layers, including the input layer. This, in essence, turns each new hidden unit into an input unit on any subsequent hidden unit additions.

There are also three training actions from which you can choose:

- The default training action is Train. When **Train** is selected, all hidden units are trained using all selected activation functions until stopping. There is no search, so this option is usually the fastest.
- An alternative to the default training action is **Increment**. The Increment option adds nodes from each of the selected activation functions, one at a time. No activation function is reused. If the added layer lowers the average error, it is retained. If not, it is dropped from the final model.
- As its name implies, the **Search** option searches for the best network, adding nodes according to the specified architecture. It is the most computationally intensive option, particularly if you do not use a validation data set. When requested, preliminary training is performed.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

7.4 Solutions

Solutions to Student Activities (Polls/Quizzes)

7.01 Poll – Correct Answer

The AutoNeural node can build neural networks with more than one hidden layer.

- ❍ True
- ❍ False

10

7.02 Multiple Choice Poll – Correct Answer

Which of the following use multiple hidden layers?

a. the block layers architecture
b. the funnel layers architecture
c. the cascade architecture
d. all of the above

19

Only the single-layer option is restricted to one hidden layer. Apart from building multiple hidden-layer networks using PROC NEURAL coding (as illustrated), currently the only way to build neural networks with more than one hidden layer is by means of the AutoNeural node.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Appendix A PROC DMDB and PROC NEURAL User's Guide

A.1 The DMDB Procedure **A-3**

A.2 The NEURAL Procedure **A-14**

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

A.1 The DMDB Procedure

Objectives

- Detail the syntax of the DMDB procedure.

3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

DMDB Procedure

```
PROC DMDB <options>;
      CLASS list-of-variables <option>;
      FREQ variable;
      ID list-of-variables;
      TARGET list-of-variables;
      VAR list-of-variables;
RUN;
```

4

SAS Enterprise Miner architecture is based on the creation of a *data mining database* (*DMDB*) that is a snapshot of the original data. The DMDB procedure creates this DMDB from the input data source. It also compiles and computes metadata information about the input data that is based on variable roles. This data is stored in a metadata catalog. The DMDB and metadata catalog facilitate subsequent data mining activities. These catalogs are designed to efficiently process and store large amounts of data.

The ASSOC, DMINE, DMNEURAL, DMSPLIT, DMREG, DMVQ, PMBR, NEURAL, SEQUENCE, and SVM procedures all depend on the catalogs that are created by the DMDB procedure.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

PROC DMDB Statement

PROC DMDB DATA=*data-set-name* <*options*>**;**

- The PROC DMDB statement invokes the DMDB procedure.

5

The PROC DMDB statement has one required argument:

DATA=*data-set-name*

specifies the SAS data set to add to the data mining database.

Here are the options for the PROC DMDB statement:

BATCH | NOMETAIN

creates a new metadata catalog as specified in the DMDBCAT= option. The default behavior is to update an existing metadata catalog. Any existing catalog with the name specified in the DMDBCAT= option is replaced with system-generated information.

CLASSOUT=*data-set-name*

creates a data set that contains the metadata for the CLASS variables that exist in the input data.

The output data set contains the following variables:

- NAME – indicates the name of the included variable. Each variable is observed once for each level of measurement.
- LEVEL – indicates the level of the observed variable.
- FREQUENCY – indicates the frequency of the observed level of measurement
- TYPE – indicates whether the variable values are character strings or numeric values.
- CRAW – displays the raw input form of any character variable values. This value is missing for numeric values.
- NRAW – displays the raw input for of any numeric variable values. This value is missing for character values.
- NMISSPERCENT – provides the frequency of the observed level of measurement as a percentage of observations without missing values.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

MISSINGONLY

provides frequencies and other applicable statistics that are based solely on the distinction between missing and nonmissing values.

MAXLEVEL=*number*

specifies the maximum number of class levels to process. The default value of *number* is the largest, available, machine integer. The value of *number* must be an integer greater than or equal to 3.

MAXOBS=*number*

specifies the maximum number of database records that are processed. Often, the total number of records in a database is not known. This option prevents the excessive download of data into SAS by limiting the imported data to *number* rows. If in-database processing is used, then this option is ignored.

NONORM

generates class-level values without normalization. A normalized value is left-justified, truncated to NORMLEN= length, and in uppercase. By default, the DMDB procedure generates metadata with normalized class levels. Normalization helps with unclean or inconsistent data. For example, the values "Yes," "YES," and "yes" all share the same normalized form, "YES." Also notice that class levels need to be unique within the default NORMLEN= length of 32 bytes. For example, the values "ABC Department store, nylon socks, blue" and "ABC Department store, nylon socks, black" would both share the normalized form "ABC DEPARMENT STORE, NYLON SOCKS" even though they are different values.

NORMLEN=*number*

specifies the normalized length of categorical variable values. The value of *number* must be an integer and defaults to 32.

OUT=*data-set-name*

names the DMDB. The DMDB contains each of the ID, VAR, and FREQ variables, copied as is from the input data set. The DMDB also converts each of the CLASS variables to a corresponding integer class level value.

PMML

creates PMML score code. The DMDB procedure produces the DataDictionary component of a PMML document. For more information about PMML, see the PMML Support in Enterprise Miner section in the SAS Enterprise Miner Help documentation.

VARDEF=DF | *number*

indicates the divisor that is used when the DMDB procedure calculates variance and standard deviation. You can specify DF to use the degrees of freedom and *number* to specify the number of observations. The default value is the degrees of freedom.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

VAROUT=*data-set-name*

creates a data set that contains the metadata for the VAR variables that exist in the input data set.

The output data set contains the following variables:

- NAME – indicates the name of the included variable. Each variable is observed once for each level of measurement.
- NMISS – indicates the number of missing values for the observed variable.
- FREQPERCENT – provides the frequency of the observed level of measurement as a percentage of all observations, including missing values.
- N – indicates the total number of observations for the observed variable, not including missing values.
- MIN – provides the smallest observed value.
- MAX – provides the largest observed value.
- MEAN – provides the average of the variables values, not including any missing values.
- STD – provides the standard deviation of the variable values, not including any missing values.
- SKEWNESS – indicates the skewness of the variable values, not including any missing values.
- KURTOSIS – indicates the kurtosis of the variable values, not including any missing values.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

CLASS Statement

CLASS *list-of-variables* <*options*>**;**

- The CLASS statement specifies a list of categorical variables that are used in later analysis.

6

The CLASS statement has one required argument:

list-of-variables

specifies a list of categorical variables that are used in later analysis. For each variable listed here, the metadata contains information about its class level value, its frequency, and its ordering information. The variables listed here can be character or numeric. These variables ***cannot*** be stated in the ID or VAR statements.

This option can be used in the CLASS statement:

sorting-order

determines how each variable is sorted. (An example that illustrates how to use this option appears at the end of this section.)

Here are the valid values of *sorting-order*:

- ASC | ASCENDING – Class levels are arranged from the lowest to the highest order of the unformatted values. This is the default value.
- DESC | DESCENDING – Class levels are arranged from the highest to the lowest order of the unformatted values.
- ASCFMT | ASCFORMATTED – Class levels are arranged from the lowest to the highest order of the formatted values.
- DESFMT | DESFORMATTED – Class levels are arranged from the highest to the lowest order of the formatted values.
- DATA | DSORDER – Class levels are arranged according to their order in the input data set.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

FREQ Statement

FREQ *variable*;

- The FREQ statement specifies a variable whose value represents the frequency of the observation.

7

The FREQ statement has one required argument:

variable

specifies a numeric variable whose value represents the frequency of the observation. If you specify this statement, then this variable is used by all other SAS Enterprise Miner procedures in the current data mining project. If the variable value is zero or missing, then the observation is omitted in the DMDB and is not included in statistical calculations.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

ID Statement

ID *list-of-variables*;

- The ID statement specifies a list of variables that contain unique identifiers for each input observation.
- These variables can be either character or numeric.

8

The ID statement has one required argument:

list-of-variables

specifies a list of variables that contain unique identifiers for each observation in the input data set. These variables can be either character or numeric.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

TARGET Statement

TARGET *list-of-variables*;

- The TARGET statement lists the target variables in the data set. The statement is optional.
- The variables in the TARGET statement must also be specified in either the VAR or the CLASS statements.

9

The TARGET statement has one required argument:

list-of-variables

lists the target variables in the data set. These variables must be specified in either the VAR or the CLASS statement. These variables can be character or numeric variables.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

VAR Statement

VAR *list-of-variables*;

- The VAR statement lists any numeric variables that are used in the analysis.

10

The VAR statement has one required argument:

list-of-variables

lists any numeric analysis variables. These variables appear in the output data set in the same order in which you specify them here. If you omit this statement, then the DMDB procedure analyzes all numeric variables that are not listed in any other statement.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Details

The DMDB is maintained as a SAS data set. The metadata information that is associated with the DMDB is maintained in a SAS catalog. Metadata includes overall data set information as well as statistical information for the variables according to their roles. For each CLASS variable, the metadata contains information about class-level values, frequencies, and ordering information. In the DMDB, the CLASS variables are stored as integers that are mapped to different class-level values.

The statistics stored for VAR variables are as follows:

- number of observations with nonmissing values
- number of observations with missing values
- minimum observed value
- maximum observed value
- sum of all nonmissing values
- corrected sum of squares
- uncorrected sum of squares
- standard deviation
- measure of the skewness of the data
- measure of the kurtosis of the data

Data mining databases are created only for training data and should not be used for validation or testing data.

Example

This example demonstrates how to create a DMDB data set and catalog. The example uses the **SAMPSIO.HMEQ** data set, which includes information about 5960 fictitious mortgages.

Each case represents an applicant for a home equity loan, and all applicants have an existing mortgage. The binary target variable **bad** indicates whether an applicant eventually defaulted or was ever seriously delinquent. There are 10 numeric input variables and two class input variables. The code below creates the DMDB data set and catalog for the **hmeq** data set.

```
proc dmdb batch data=sampsio.hmeq out=DMhmeq dmdbcat=CAThmeq;
   var loan derog mortdue value yoi delinq clage ninq clno debtinc;
   class bad(desc) reason(ascending) job;
   target bad;
run;
```

The data set **DMhmeq** contains the DMDB data set, and **CAThmeq** contains the DMDB catalog. Both should be found in your Work directory. The numeric variables are specified in the VAR statement and the categorical variables are specified in the CLASS statement. You cannot specify a variable in both of these statements. Notice that this data set does not contain an ID or a FREQ variable. The TARGET statement is used to identify the variable **bad** as the target variable.

Notice that the variable **bad** is sorted in descending order, but **reason** is sorted in ascending order. You can specify different sorting methods for each of the variables in the CLASS statement. Because nothing is specified for **job**, it is automatically sorted in ascending order.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

A.2 The NEURAL Procedure

Objectives

- Detail the syntax of the NEURAL procedure.

12

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

ARCHI Statement

ARCHI *architecture-name* <**HIDDEN=***n*> <**DIRECT**>**;**

- The ARCHI statement enables you to construct a neural network with either zero or one hidden layer.
- It automatically sets the appropriate combination and activation functions for the specified architecture.

13

The ARCHI statement enables you to construct a neural network with, ***at most, one hidden layer***. The *architecture-name* parameter names the neural network architecture that you want to apply. The specified *architecture-name* parameter in the ARCHI statement automatically assigns the appropriate hidden-unit combination (COMBINE=) and activation (ACT=) functions. The following architectures are available:

- generalized linear model (GLIM)
- multilayer perceptron (MLP)
- ordinary radial basis function with equal widths (ORBFEQ)
- ordinary radial basis function with unequal widths (ORBFUN)
- normalized radial basis function with equal widths and heights (NRBFEQ)
- normalized radial basis function with equal heights and unequal widths (NRBFEH)
- normalized radial basis function with unequal heights and equal widths (NRBFEW)
- normalized radial basis function with equal volume (NRBFEV)
- normalized radial basis function with unequal heights and unequal widths (NRBFUN)

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

The model that is selected affects certain options in the INPUT, TARGET, RANOPTIONS, and HIDDEN statements, as summarized in this table:

Architecture	INPUT Arguments	TARGET Arguments	HIDDEN Arguments
GLIM	STD=NODE	STD=NONE	No Hidden Layers
MLP			ACT=TANH COMBINE=LINEAR
ORBFEQ			ACT=EXP COMBINE=EQRADIAL
ORBFUN			ACT=EXP COMBINE=EHRADIAL
NRBFEQ		STD=NOBIAS	ACT=SOFTMAX COMBINE=EQRADIAL
NRBFEH		STD=NOBIAS	ACT=SOFTMAX COMBINE=EHRADIAL
NRBFEW		STD=NOBIAS	ACT=SOFTMAX COMBINE=EWRADIAL
NRBFEV		STD=NOBIAS	ACT=SOFTMAX COMBINE=EVRADIAL
NRBFUN		STD=NOBIAS	ACT=SOFTMAX COMBINE=XRADIAL

The ARCHI statement has only two options:

HIDDEN=*n*

specifies the number of hidden units *n* for all architectures other than GLIM. There is no default number of hidden units. A value for *n* must be specified.

DIRECT

requests that direct connections from the input layer to the output layer are added.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

CODE Statement

```
CODE FILE=filename
      <ERROR | NOERROR>
      <CATALOG= | CAT= | C=>
      <FORMAT=format>
      <GROUP=name>
      <LINESIZE | LS=integer-value>
      <RESIDUAL | NORESIDUAL>
      <DUMMIES | NODUMMIES>
      <LOOKUP=lookup-algorithm>
      <PMML | XML>;
```

- The CODE statement writes SAS DATA step (scoring) code to the specified file or catalog entry.

14

The CODE statement writes SAS DATA step (scoring) code to a file or catalog entry. This code can later be included in a DATA step that uses a SET statement and reads a data table to be scored.

The ***required*** FILE= argument specifies where to write the code.

FILE=*filename*

specifies where to write the code (for example, FILE='c:\mydir\scorecode.sas').

The CODE statement options are listed below.

ERROR | NOERROR

specifies whether the error function is to be computed. The default is NOERROR.

CATALOG= | CAT= | C=*library.catalog.entry.type*

specifies where to write the code. The default location is determined by the USER= setting.

FORMAT=*SAS format*

formats weights or other numeric values that do not have a format that is given in the input data set. The default format is BEST20.

GROUP=*identifier*

specifies the identifier for group processing. The identifier should be a maximum of 16 characters.

LINESIZE= | LS=*number*

specifies a line size for generated code. Permissible values range from 64 to 254. The default is 72.

RESIDUAL | NORESIDUAL

generates residual values. The default is NORESIDUAL.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

DUMMIES | NODUMMIES

specifies whether to keep dummy variables. The default is NODUMMIES.

LOOKUP=*method*

specifies the method that you want to use to look up CLASS levels. The available methods are as follows:

- AUTO (default) selects the fastest method for each variable that is portable.
- BINARY uses a binary search. It is fast, but it might produce incorrect results.
- LINEAR uses a linear search with IF statements with categories in the order specified in the data mining database catalog. This method is slow if there are many categories.
- LINFREQ uses a linear search. This method is fast if the distribution of class frequencies is uneven, but it is slow if there are many categories with approximately equal frequencies.
- SELECT uses a SELECT statement. This method is slow if there are many categories.

PML | XML

specifies PMML to produce scoring code in Predictive Modeling Markup Language, an XML-based standard for representing data mining results. For more information, see the PMML Support in Enterprise Miner section in the Enterprise Miner 5.3 Java Help.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

CONNECT Statement

CONNECT *ID-list </ options>*;

- The CONNECT statement defines the connection between the neurons that are identified in the *ID-list*.
- The *ID-list* argument is defined using the ID= option in an INPUT, HIDDEN, or TARGET statement.

15

By default, each layer is fully connected to the next layer. If you do not want this architecture, you can use one or more CONNECT statements to explicitly define network connectivity.

The ***required*** *ID-list* argument lists the identifiers of the two or more layers to connect. The identifiers must be previously defined using the ID= option in an INPUT, HIDDEN, or TARGET statement.

The CONNECT statement options are listed below.

RANDIST=CAUCHY | CHIINV | NORMAL | UNIFORM

specifies the distribution that is used to generate random initial weights and perturbations. The default setting is NORMAL.

RANDIST=	RANLOC=	RANSCALE=	RANDF=
CAUCHY	median=0	scale=1	
CHIINV		scale=1	df=1
NORMAL	mean=0	std=1	
UNIFORM	mean=0	halfrange=1	

RANLOC=*number*

specifies the seed for the random number generator.

RANSCALE=*number*

specifies the scale parameter for random numbers.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

CUT Statement

CUT *ID-list* **| ALL;**

- The CUT statement removes connections and the corresponding weights.
- The *ID-list* specifies the layers to be disconnected.
- Specifying ALL disconnects all layers.

16

If the connection between two layers is not contributing to the predictive ability of the network, you can use a CUT statement to remove that connection and the corresponding weights.

The *ID-list* argument specifies the identifiers of the layers to disconnect. The ALL argument disconnects all layers from the neural network.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

DECISION Statement

DECISION DECDATA=<*libref.*> *SAS-data-set*
<DECVARS=*decision-variable(s)*> <*options*>**;**

- The DECISION statement specifies information that is used for decision processing.
- DECISION statement options enable you to apply prior probability and cost (profit) considerations.

17

The DECISION statement specifies information that is used for decision processing in PROC NEURAL. Its ***required*** DECDATA= argument and the available options are described below.

DECDATA=<*libref.*> *SAS-data-set*

specifies the input data set that contains the decision matrix. The DECDATA= data set must contain the target variable. It can also contain decision variables that are specified by means of the DECVARS= option, and prior probability variables that are specified by the PRIORVAR= or OLDPRIORVAR= option. The matrix can be of different types (PROFIT, LOSS, or REVENUE). The default is PROFIT.

DECVARS=*decision-variable(s)*

specifies the decision variables in the DECDATA= data set that contain the target-specific consequences for each decision.

COST=*costs-list*

specifies numeric constants that give the cost (negative profit) of a decision, or variables in the DATA= data set that contain the case-specific costs, or any combination of constants and variables (default = 0). There must be the same number of cost constants and variables as there are decision variables specified in the DECVARS= option. The DECDATA= data must be TYPE=REVENUE.

PRIORVAR=*variable*

specifies the variable in the DECDATA= data set that contains the prior probabilities to use for making decisions. The default is none.

OLDPRIORVAR=*variable*

specifies the variable in the DECDATA= data set that contains the prior probabilities that were used when originally fitting the model. The default is none. If you specify OLDPRIORVAR=, then you must specify PRIORVAR=.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

DELETE Statement

DELETE *ID-list*;

- The DELETE statement enables you to remove layers that are not contributing to the neural network.
- It also removes all of the associated weights.

18

If an input or hidden layer is not contributing to the predictive ability of the network, you can remove that layer with a DELETE statement. The DELETE statement also removes all associated weights.

The ***required*** *ID-list* argument specifies the identifiers of layers and neurons to delete.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

FREEZE Statement

```
FREEZE weight-list /
        <VALUE=number>
        <EST=<libref.>SAS-data-set>;
```

- Freezing retains the weights at their current value until a corresponding THAW statement is executed.
- This causes training to proceed faster, and it requires less computer memory.

19

Normally, during training, all weights are updated. If you freeze one or more weights, those weights retain their frozen values until a corresponding THAW statement is executed. Thus, freezing weights generally causes training to proceed more quickly. It also requires less computer memory.

The *weight-list* argument lists the weights to be frozen. It consists of zero or more repetitions of *wname* → *wname-2*, where *wname* is a unit name, a layer identifier, a BIAS, or an ALTITUDE, and *wname-2* is a unit name or a layer identifier.

The FREEZE statement options are described below.

VALUE=*number*

specifies the numeric value to which weights are to be frozen.

EST=<*libref.*>*SAS-data-set*

specifies the SAS data set that contains the values to which weights are to be frozen.

You can specify either the VALUE= or EST= option, but not both. If neither option is specified, the weights are frozen to their current values.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

FREQ Statement

FREQ *variable*;

- The FREQ statement specifies the frequency variable to be used for training.
- PROC NEURAL automatically looks for this variable in the DATA=, VALIDATA=, and TESTDATA= data sets.
- If the frequency variable is not found in any of these data sets, a value of 1 is used for all observations.

20

The FREQ statement contains the frequency of each observation.

PROC NEURAL searches for the name of the specified FREQ variable in the DATA=, VALIDATA=, and TESTDATA= data sets. If the name does not appear in any of these data sets, the NEURAL procedure issues a warning message but continues processing using a value of 1 for all observations.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

HIDDEN Statement

HIDDEN *number* **/ ID=***name*
<**ACT=***activation-function*>
<**BIAS|NOBIAS**>
<**COMBINE=***combination-function*>**;**

- Each HIDDEN statement defines a hidden layer, which consists of one or more hidden units.

21

Each HIDDEN statement defines a hidden layer. You can declare as many HIDDEN statements as you want, up to the limits imposed by the computer. By adding a hidden layer that consists of only a single neuron, the HIDDEN statement can be used to build neural networks one neuron at a time.

The following two arguments are ***required***:

- *number*, which gives the number of units in the hidden layer
- ID=, which specifies the identifier for the layer

The HIDDEN statement options are as follows:

ACT=*activation-function*

specifies the activation function to be used. The default depends both on the number of units in the layer and on the combination function that was specified.

- When COMBINE=ADD, the default is ACT=IDENTITY.
- For COMBINE=LINEAR or EQSLOPES, the default is ACT=TANH.
- When COMBINE=EHRADIAL, EQRADIAL, EVRADIAL, EWRADIAL, or XRADIAL, the default is ACT=EXP if there is only one hidden unit in the layer. Otherwise, the default is ACT=SOFTMAX.

BIAS | NOBIAS

specifies whether to use a bias term in the hidden layer neuron (or neurons). The default is BIAS.

COMBINE=*combination-function*

specifies the combination function to be used. For the MLP, the available combination functions are EQSLOPES, ADD, and LINEAR (the default). For an RBF, the available combination functions are EHRADIAL, EQRADIAL, EWRADIAL, EVRADIAL, and XRADIAL.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

INITIAL Statement

```
INITIAL <BIADJUST=adjustment-value>
        <INEST=<libref.>SAS-data-set>
        <INFAN=number>
        <OUTEST=<libref.>SAS-data-set>
        <RANDBIAS|NORANDBIAS>
        <RANDOUT|NORANDOUT>
        <RANDOM=integer>
        <RANDSCALE | NORANDSCALE>;
```

- The INITIAL statement gives initial values to all the weights and biases in the network.
- If an INITIAL statement is not provided, the network is initialized using the default specifications.

22

After a network is defined, all weights and biases must be given initial values before any training is performed. ***There are no required arguments.*** If you train a network without supplying an INITIAL statement, the network is initialized using the default specifications.

The INITIAL statement offers an extensive set of options.

BIADJUST=*adjustment-value*

specifies how to adjust the random biases for units with a linear combination function. The bias is adjusted by multiplying by a function of the weights indicated by *adjustment-value* and dividing by the scale (RANSCALE=) of the distribution from which the random bias was drawn.

The *adjustment-value* can be any one of the following:

- NONE is no bias adjustment. (This is the default.)
- SUM adjusts random initial biases for the sum of the absolute connection weights leading to the unit. It is typically used with STD=MIDRANGE for inputs and RANDIST=UNIFORM.
- USS adjusts random initial biases for the square root of the sum of squared connection weights leading to the unit. This is typically used with STD=STD for inputs and RANDIST=NORMAL.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

INEST=<*libref.*>*SAS-data-set*

specifies an input data set that contains some or all of the weights. An INEST= data set typically is created by using the OUTEST= option in a SAVE or a TRAIN statement from a previous execution of the NEURAL procedure. Any weights in the INEST= data set that have missing values are assigned random values.

INFAN=*number*

divides random connection weights by (*fan-in of unit*)*******number*, where the *fan-in* of a unit is the number of other units feeding into that unit, not counting the bias or altitude. The default value for INFAN is 0 for radial combination functions. Otherwise, it is 0.5.

OUTEST=<*libref.*>*SAS-data-set*

specifies the output data set that contains all the initial weights.

RANDBIAS | NORANDBIAS

specifies whether to randomize output biases. The default is NORANDBIAS, which sets the bias to the inverse activation function of the target mean.

RANDOM=*integer*

specifies the random number seed. The default is 0.

RANDOUT | NORANDOUT

specifies whether to randomize the output connection weights. The default is NORANDOUT, which sets the weights to 0.

RANDSCALE | NORANDSCALE

specifies whether to randomize target scale estimates. The default, NORANDSCALE, sets each scale estimate to the standard deviation of the corresponding target variable.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

INPUT Statement

INPUT *variable-list* / **ID**=*name*
<**LEVEL**=*value*>
<**STD**=*method*>**;**

- The INPUT statement groups input variables that have common levels (nominal, ordinal, or interval).
- You can specify as many INPUT statements as needed.

23

The INPUT statement enables you to group input variables with common levels. You can specify as many INPUT statements as you want, up to the limits imposed by the computer.

Like the HIDDEN statement, there are two ***required*** arguments:

- *variable-list*, which specifies the input variables
- ID=*name*, which specifies an identifier for the layer

The available options are as follows:

LEVEL=*value*

specifies the measurement level, where *value* can be one of the following:

- NOMINAL | NOM nominal (unordered discrete variable)
- ORDINAL | ORD ordinal (ordered discrete variable)
- INTERVAL | INT interval (continuous variable)

STD=*method*

specifies the standardization method, where *method* is one of the following:

- STD Variables are rescaled to have a mean of 0 and a standard deviation of 1. This is the default.
- RANGE | RAN Variables are rescaled to a minimum of 0 and a range of 1.
- MIDRANGE | MID Variables are rescaled a midrange of 0 and a half range of 1. That is, they are rescaled to a minimum of -1 and a maximum of 1.
- NONE | NO Variables are not altered.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

NETOPTIONS Statement

NETOPTIONS *options*;

- The NETOPTIONS statement is used to set various network options, including the following items:
  - the initial random weight values
  - the objective function
  - the weight decay parameter

24

The NETOPTIONS statement is used to set ***network options***. The available options are as follows:

DECAY=*number*

specifies the weight decay. The *number* argument must be greater than or equal to 0. The default value depends on the optimization algorithm that is used.

- For quick-prop (QPROP) optimization, the default weight decay value is 0.0001.
- For all others optimization techniques, the default value is 0.

INVALIDTARGET=*action*

specifies the action to be taken during training if an out-of-range target value is found, where *action* can be one of the following values:

- STOP — If INVALIDTARGET=STOP is specified, an error is issued, and training is terminated. This is the default action.
- OMITCASE | OMIT — If INVALIDTARGET=OMITCASE is specified, a warning is given and the observation is discarded, but training continues.

OBJECT=*objective-function*

specifies the objective function, where *objective-function* can be one of the following:

- DEV — requests deviance (for ERROR=NORMAL, this is least squares).
- LIKE — requests negative log-likelihood.
- MEST — requests M estimation.

The default objective function that is used depends on the error function that is specified.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

RANDF=*number*

specifies the degrees of freedom parameter for random numbers.

RANDIST=*name*

specifies the type of distribution to be used for random initial weights and perturbations.

RANDIST=	RANLOC=	RANSCALE=	RANDF=
CAUCHY	median=0	scale=1	
CHIINV		scale=1	df=1
NORMAL	mean=0	std=1	
UNIFORM	mean=0	halfrange=1	

RANDOM=*integer*

specifies the random number seed.

RANLOC=*number*

specifies the location parameter for random numbers.

RANSCALE=*number*

specifies the scale parameter for random numbers.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

PROC NEURAL Statement

PROC NEURAL DATA=<*libref.*>*SAS-data-set*
DMDBCAT=*catalog* <*options*>**;**

- The PROC NEURAL statement invokes the neural network procedure.
- Options include the ability to read in saved networks and to assign validation and test data sets.
- The SAS (DATA=) data set must already be cataloged by means of the DMDB procedure.

25

The PROC NEURAL statement invokes the neural network procedure. The DMDBCAT= argument points to the location in which the DATA= data set was cataloged (using PROC DMDB).

The following PROC NEURAL statement options are available:

GRAPH

plots the objective function, validation error, and test error that was achieved during training.

NETWORK=*screen-specification*

constructs a network according to a description that was saved by using a SAVE statement during a previous execution. The default is none.

RANDOM=*integer*

specifies the random number seed used in network weight initialization. The default is 12345. When 0 or a negative value is entered, the system uses the system clock to generate the seed.

RANSCALE=*decimal*

determines the number of standard deviations that are used in the initial random weight generation process. ***Specifying a small value (for example, 0.1) encourages the training algorithm to move slowly away from a linear model.***

RANLOC=*number*

specifies the mean that is used in the initial random weight generation process.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

STOPFILE='*file path*'

enables you to stop training when you are running a large job. When you want to stop the job, create the specified file (for example, STOPFILE='c:\mydir\haltneural'). Initially this file should not exist. The NEURAL procedure checks for the existence of this file between iterations. When you want to stop, create the specified file, and PROC NEURAL stops. The file does not need to have any contents.

TESTDATA=<*libref.*>*SAS-data-set*

specifies a data set that is used to compute the average ***test*** error during training. At selected iterations, controlled by ESTITER=, each observation in the TESTDATA= data set is read in and scored, and the error is computed. The average error is then written to the output of the OUTEST= data set. The data set must contain all of the input and target variables.

VALIDATA=<*libref.*>*SAS-data-set*

specifies a data set that is used to compute the average ***validation*** error during training. At selected iterations, controlled by ESTITER=, each observation in the VALIDATA= data set is read in and scored, and the error is computed. The average error is then written to the output of the OUTEST= data set. The data set must contain all of the input and target variables.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

NLOPTIONS Statement

NLOPTIONS <*options*>**;**

- The NLOPTIONS statement sets nonlinear optimization options, including the following choices:
  - initial Hessian matrix estimates and scale
  - radius diameter for the trust region, double dogleg, and Levenberg-Marquardt optimization methods
  - convergence criteria

26

The NLOPTIONS statement is used to set the nonlinear ***optimization options***. The extensive list of available nonlinear options is itemized below.

ABSCONV=*number*

specifies the absolute function convergence criterion. The default is the negative square root of the largest double-precision value. The value of *number* must be greater than 0.

ABSFCONV=*number*

specifies the absolute function convergence criterion. The value of *number* must be greater than 0. The default is 0.

ABSGCONV=*number*

specifies the absolute gradient convergence criterion. (The default is 1E-5.) The value of *number* must be greater than 0.

ABSXCONV=*number*

specifies the absolute parameter convergence criterion. The value of *number* must be greater than 0. The default is 0.

DAMPSTEP=*number*

specifies that the initial step size value for each line search used by the QUANEW, CONGRA, or NEWRAP optimization technique cannot be larger than the product of *number* and the step size value used in the former iteration. The value of *number* must be greater than 0. The default is 2.

DIAHES

forces the optimization (TRUREG, NEWRAP, or NRRIDG) to take advantage of the diagonality.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

FCONV=*number*

specifies the function convergence criterion. The value of *number* must be greater than 0. The default is 0.0001.

FSIZE=*number*

specifies the relative function and relative gradient termination criteria. The value of *number* must be greater than or equal to 0. The default is N/A.

GCONV=*number*

specifies the relative gradient convergence criterion. The value of *number* must be greater than 0. The default is 1E-8.

HESCAL=0 | 1 | 2 | 3

specifies the scaling version of the Hessian or cross-product Jacobian matrix used in NRRIDG, TRUREG, LEVMAR, NEWRAP, or DBLDOG optimization. The default value is 1 for the Levenberg-Marquardt minimization technique, and 0 for all other methods.

INHESSIAN=*number*

specifies how to define the initial estimate of the approximate Hessian for the quasi-Newton techniques QUANEW and DBLDOG. The value of *number* must be greater than 0. The default is to use a Hessian based on the initial weights.

INSTEP=*number*

specifies the initial radius of the trust region used in the TRUREG, DBLDOG, and LEVMAR algorithms. The value of *number* must be greater than 0. The default is 1.

LCEPSILON | LCEPS=*number*

specifies the range for active constraints. The value of *number* must be greater than 0.

LCSINGULAR=*number*

specifies the tolerance for dependent constraints. The value of *number* must be greater than 0.

LINESEARCH=*number*

specifies the line-search method for the CONGRA, QUANEW, and NEWRAP optimization techniques. The range of valid *number* values is from 1 to 8 (inclusive). The default is 2.

LSPRECISION=*number*

specifies the degree of accuracy to be obtained by the second and third line-search algorithms. The default depends on the line-search techniques that are specified.

MAXFUNC=*number*

specifies the maximum number of optimization function calls. The value of *number* must be greater than 0. The default is 2147483647.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

MAXITER=*number*

specifies the maximum number of iterations in the optimization process. The value of *number* must be greater than 0. The default is 100 for TRUREG, NRRIDG, NEWRAP, and LEVMAR; 200 for QUANEW and DBLDOG; and 400 for CONGRA.

MAXSTEP=*number*

specifies the upper bound for the step length of the line-search algorithms. The value of number must be greater than 0. The default is the largest double-precision value.

MAXTIME=*number*

specifies the upper limit of CPU time (in seconds) for the optimization process. The value of *number* must be greater than 0. The default is 604800.

MINITER=*number*

specifies the minimum number of iterations in the optimization process. The value of *number* must be greater than or equal to 0. The default is 0.

NOPRINT

suppresses all output. Only errors, warnings, and notes are printed in the log file.

PALL

prints all optional output except the output generated by PSTDERR , LIST, or LISTCODE.

PSUMMARY

restricts the amount of default printed output to a short form of iteration history, warnings, and errors.

PHISTORY

prints the optimization history. If PSUMMARY or NOPRINT is not specified, then the PHISTORY option is set automatically and the iteration history is printed by default.

RESTART=*number*

specifies that the QUANEW or CONGRA algorithm is restarted with a steepest descent or ascent search direction after the specified number of iterations is completed. If CONGRA and UPDATE ≠ PB, the default value of *number* is given by the number of parameters. For QUANEW, *number* is the largest integer. In all cases, *number* must be greater than or equal to 1. For the CONGRA technique when UPDATE=PB, *number* is not used (because a restart is always performed).

SINGULAR=*number*

specifies an absolute singularity criterion for the computation of the inertia of Hessian and cross-product Jacobian and their projected forms. The value of *number* must be greater than 0. The default is 1E-8.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

TECHNIQUE=*method*

specifies an optimization technique, where *method* is one of the following values:

- CONGRA specifies the conjugate gradient optimization technique. This is the default value when the number of parameters is greater than or equal to 400.
- DBLDOG specifies the double-dogleg optimization technique.
- NEWRAP specifies the Newton-Raphson with Line Search optimization technique.
- NRRIDG specifies the Newton-Raphson with Ridging optimization technique. This is the default value when there are fewer than or as many as 40 parameters.
- QUANEW specifies the quasi-Newton optimization technique. This is the default value when there are between 40 and 400 parameters.
- TRUREG specifies the trust-region optimization technique.

UPDATE=*update-type*

specifies an update method, where *update-type* value is one of the following:

- BFGS for TECHNIQUE=QUANEW performs the BFGS (Broyden-Fletcher-Goldfarb-Shanno) update of the Cholesky factor of the Hessian matrix.
- CD for TECHNIQUE=CONGRA performs a conjugate descent update of Fletcher.
- DBFGS for TECHNIQUE=DBLDOG or QUANEW performs the dual BFGS update of the Cholesky factor of the Hessian matrix.
- DDFP for TECHNIQUE=DBLDOG or QUANEW performs the dual DFP (Davidon-Fletcher-Powell) update of the Cholesky factor of the Hessian matrix.
- DFP for TECHNIQUE=QUANEW performs the original DFP update of the inverse Hessian.
- FR for TECHNIQUE=CONGRA performs the Fletcher-Reeves update.
- PB for TECHNIQUE=CONGRA performs the automatic restart update method of Powell and Beale. This is the default for TECHNIQUE=CONGRA.
- PR for TECHNIQUE=CONGRA, UPDATE performs the Polak-Ribière update.

VERSION=1 | 2 | 3

specifies the version of the hybrid quasi-Newton optimization technique or the version of the quasi-Newton optimization technique with nonlinear constraints. The default is 2.

XCONV=*number*

specifies the parameter convergence criterion. The value of *number* must be greater than 0. The default is 0.

XSIZE=*number*

specifies the number of successive iterations that the criterion must satisfy before the optimization process can be terminated. The value of *number* must be greater than 0. The default is 0.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

PERFORMANCE Statement

```
PERFORMANCE <COMPILE | NOCOMPILE>
            <CPUCOUNT=number / ACTUAL>
            <MULTIPASS>
            <THREADS | NOTHREADS>;
```

- The PERFORMANCE statement requests that threaded (parallel) computation be used.
- This statement is ignored unless the computation is performed on a multi-processor or multi-core machine.

27

The PERFORMANCE statement requests that threaded (that is, parallel) processing be performed. This statement is ignored unless the computation is performed on a multi-processor or a multi-core machine.

COMPILE | NOCOMPILE

requests that code be generated and compiled for a specific network. The compiled code is then used in subsequent network computation. For complex networks, this can help improve efficiency. The default is NOCOMPILE.

CPUCOUNT=*number* | ACTUAL

requests that the number of threads equals either the ACTUAL number of processors present or the number of processors specified by the *number* parameter (when CPUCOUNT=*number* is used).

MULTIPASS

requests that a utility file ***not*** be written for the training data table. This can increase the CPU time that is available for training the network.

Training a neural network can require reading the training data multiple times. Various flags that are associated with the input and target values must be computed for each observation. CPU time can be saved if the training data is read only once, and the computed flags are written to an internal utility file, which is then used in place of the training file during training.

THREADS | NOTHREADS

determines whether thread (parallel) processing is to be performed. THREADS=YES, or simply THREADS, requests that thread processing be performed. THREADS=NO (or equivalently NOTHREADS) turns off multithreading. The default is NOTHREADS.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

PERTURB Statement

PERTURB *weight-list* / <**RANDF=***number*>
<**OUTEST=***<libref.>SAS-data-set*>
<**RANDIST=***name*>
<**RANDOM=***integer*>
<**RANLOC=***number*>
<**RANSCALE=***number*>**;**

- The PERTURB statement perturbs the weights.
- This sometimes enables the network to escape a local error minimum.

28

Perturbing a network's weights can sometimes enable you to escape a local error minimum.

The PERTURB statement perturbs the weights. The ***required*** *weight-list* argument identifies the weights to be perturbed, and the available options, listed below, control the perturbation that is applied.

RANDF=*number*

specifies the degrees of freedom parameter for random numbers. For the CHIINV distribution, the default is 1. There is no default for the NORMAL, UNIFORM, and CAUCHY distributions.

OUTEST=*<libref.>SAS-data-set*

specifies the output data set that contains all the weights. The default is none.

RANDIST=*name*

specifies the type of distribution for random numbers.

RANDOM=*integer*

specifies the random number seed. The default is 0.

RANLOC=*number*

specifies the location parameter for random numbers. For the NORMAL, UNIFORM, and CAUCHY distributions, the default is to place the mean or median at 0. There is no default location for CHIINV.

RANSCALE=*number*

specifies the scale parameter for random numbers. The default is 1.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

PRELIM Statement

```
PRELIM integer MAXITER=integer
               <ACCELERATE=number>
               <DECELERATE=number>
               <INEST=<libref.>SAS-data-set>
               <LEARN=number>
               <MAXLEARN=number>
               <MAX | MAXMOMENTUM=number>
               <MINLEARN=number>
               <MOM | MOMENTUM=number>
               <OUTEST=<libref.>SAS-data-set>
               <PREITER=integer>
               <PRETECH=name>
               <PRETIME=number>
               <RANDBIAS|NORANDBIAS>
               <RANDOUT|NORANDOUT>
               <RANDOM=integer>;
```

29

Preliminary training reduces the risk of falling into bad local optima.

The PRELIM statement repeatedly (by default, five times) trains a network for a small number of iterations (by default, 10) using different starting locations. The weights of the best performing of these partially trained networks are used as the initial weights in subsequent training.

In addition to the ***required*** *integer* argument, which specifies the number of preliminary optimizations, a large number of options is available. These options are listed below.

ACCELERATE | ACCEL=*number*

specifies the rate of increase of learning for the RPROP optimization technique. The value of *number* must be greater than 1. The default is 1.2.

DECELERATE | DECEL=*number*

specifies the rate of decrease of learning for the RPROP optimization technique. The value of *number* must be greater than 0 and less than 1. The default is 0.5.

INEST=<*libref.*>*SAS-data-set*

specifies the input data set that contains some or all weights. Any weights in the INEST= data set that have missing values are assigned initial random values according to the RANDOM=, RANDOUT, and RANDBIAS options, as well as the options that pertain to random number distributions that you specify in the RANDOM statements.

LEARN=*number*

specifies the learning rate for BPROP or the initial learning rate for QPROP and RPROP. The value of *number* must be greater than 0. The default is 0.1.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

MAXITER=

specifies the number of steps from each starting point.

MAXLEARN=*number*

specifies the maximum learning rate for RPROP. The value of *number* must be greater than 0. The default value is given by the reciprocal of the square root of the machine epsilon.

Machine epsilon is the machine's precision or unit round-off.

MAXMOMENTUM | MAXMOM=*number*

specifies the maximum momentum for BPROP. The value of *number* must be greater than 0. The default is 1.75.

MINLEARN=*number*

specifies the minimum learning rate for RPROP. The value of *number* must be greater than 0. The default value is given by the square root of the machine epsilon.

MOMENTUM | MOM=*number*

specifies the momentum for BPROP and RPROP. The value of *number* must be greater than or equal to 0 and less than 1. The default is dependent on the algorithm that is used. For BPROP, the default is 0.9. For RPROP, it is 0.1.

OUTEST=*<libref.>SAS-data-set*

specifies the output data set that contains all the weights.

PREITER=*integer*

specifies the maximum number of iterations in each preliminary optimization. The default is 10.

PRETECH | TECHNIQUE=*name*

specifies the optimization technique. The default is the same as the TECHNIQUE= defaults in the TRAIN statement. (See the TRAIN statement syntax.)

PRETIME=*number*

specifies the amount of time after which training stops.

RANDBIAS | NORANDBIAS

specifies whether to randomize output biases. The default is NORANDBIAS, which sets bias to the inverse activation function of the target mean.

RANDOM=*integer*

specifies the random number seed. The default is 0.

RANDOUT | NORANDOUT

specifies whether to randomize the output connection weights. The default is NORANDOUT, which sets the output connection weights to 0.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

QUIT Statement

```
QUIT;
```

- The QUIT statement, as its name suggests, stops the NEURAL procedure's execution.

30

The QUIT statement stops the NEURAL procedure.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

RANOPTIONS Statement

RANOPTIONS *connection-list* /
<**RANDF=***number*>
<**RANDIST=***name*>
<**RANDOM=***integer*>
<**RANLOC=***number*>
<**RANSCALE**=*number*>;

- The RANOPTIONS statement specifies the distribution of the random initial weights.

31

The RANOPTIONS statement specifies the distribution of the initial random weights. The ***required*** *connection-list* argument identifies the connections to be randomized.

The statement's options are described below.

RANDF=*number*

specifies the degrees of freedom parameter for random numbers. The default is 1.

RANDIST=*name*

specifies the type of distribution for random numbers. The default is NORMAL.

RANDOM=*integer*

specifies the random number seed. The default is 0.

RANLOC=*number*

specifies the location parameter for random numbers. The default mean (or median) value is 0 for the NORMAL, UNIFORM, and CAUCHY distributions. There is no default location for CHIINV.

RANSCALE=*number*

specifies the scale parameter for random numbers. The default is 1.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

SAVE Statement

SAVE <**OUTEST=**<*libref.*>*SAS-data-set*>
<**NETWORK=***screen-specification*>**;**

- The SAVE statement saves the network definition.
- It either writes the weights to a data set or a description of the network to a catalog entry.

32

The SAVE statement writes weights to a data set or a description of the network to a catalog entry. There is no single argument that is required, but at least one of the following options must be specified:

OUTEST=<*libref.*> *SAS-data-set*

saves the network weights in an output data set.

NETWORK=*screen-specification*

saves the definition of the network. The *screen-specification* argument is the name of a catalog entry.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

SCORE Statement

```
SCORE DATA=<libref.>SAS-data-set
        <DUMMIES | NODUMMIES>
        <HIDDEN>
        <OUT=<libref.>SAS-data-set>
        <OUTFIT=<libref.>SAS-data-set>
        <ROLE=role-option>;
```

- The SCORE statement creates a data set that contains predicted values, residuals, classifications, and other useful statistical information.
- Valid role options are Train, Validate, Test, and Score.

33

The SCORE statement creates an output data set that contains predicted values and possibly other results such as residuals, classifications, decisions, and assessment values. The DATA=<*libref.*>*SAS-data-set* option specifies the data set to be scored. If no DATA= option is given, the training data set is used.

The required SCORE statement argument is the following:

OUT=<*libref.*>*SAS-data-set*

specifies the data set that contains the outputs.

The SCORE statement options are listed below.

DUMMIES | NODUMMIES

specifies whether to write dummy variables to the OUT= data table. The default is NODUMMIES.

HIDDEN

asks that the hidden unit activation values also be written to output.

OUTFIT=<*libref.*>*SAS-data-set*

specifies the output data set that contains the fit statistics.

ROLE=*role-options*

specifies the role of the DATA= data table, primarily affecting which fit statistics are computed and what their names and labels are. Applicable *role-options* values are TRAIN, VALIDATE, TEST, or SCORE. The default is TEST.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

SET Statement

SET *weight-list number*;

- The SET statement sets the value of weights that are specified in the weight list to the value *number*.

34

The SET statement does ***not*** freeze weights, so subsequent training might change the values of the weights that are specified in the SET statement.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

SHOW Statement

SHOW <STATEMENTS> <WEIGHTS>**;**

- The SHOW statement prints the statements or weights that PROC NEURAL can use to reproduce the network.

35

The SHOW statement prints information about the network. You must specify one of the following options:

STATEMENTS

prints statements that can be used with the NEURAL procedure to reproduce the network.

WEIGHTS

prints the network weights.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

TARGET Statement

```
TARGET | OUTPUT variable-list /
        <ACT=activation-function>
        <BIAS|NOBIAS >
        <COMBINE=combination-function>
        <ERROR=keyword>
        <ID=name>
        <LEVEL=value>
        <MESTA=number>
        <MESTCON=number>
        <SIGMA=number>
        <STD=method>;
```

- The TARGET statement identifies the target variables.
- It is also used to specify the target layer activation and error functions.

36

The TARGET statement defines the target (output) layer. The required arguments are as follows:

- *variable-list* specifies the target variables.
- *ID=name* assigns a unique identifier to the output layer.

The TARGET statement offers the following options:

ACT=*activation-function*

specifies the activation function to be used. The default depends on the measurement level:

- If LEVEL=INTERVAL, then the default is ACT=IDENTITY.
- If LEVEL=ORDINAL, then the default is ACT=LOGISTIC.
- If LEVEL=NOMINAL, then the default is ACT=MLOGISTIC.

Use MLOGISTIC for ERROR=MBERNOULLI, MENTROPY, or MULTINOMIAL.

BIAS | NOBIAS

specifies whether to use a bias term in each output layer neuron. The default is BIAS.

COMBINE=*combination-function*

specifies the combination function to be used. For an MLP, the available combination functions are EQSLOPES, ADD, and LINEAR (the default). For an RBF, the available combination functions are EHRADIAL, EQRADIAL, EWRADIAL, EVRADIAL, and XRADIAL.

ERROR=*keyword*

specifies the error function to be applied. The default is NORMAL for LEVEL=INTERVAL. Otherwise, the default is MBERNOULLI.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

LEVEL=*value*

specifies the measurement level, where *value* can be NOMINAL (the default for character variables), ORDINAL, or INTERVAL (the default for numeric variables).

MESTA=*number*

specifies the scale constant for M-estimation. The default value is computed from the MESTCON= value to give consistent scale estimates for normal noise.

MESTCON=*number*

specifies the tuning constant for M-estimation. For HUBER, the default is 1.5. If BIWEIGHT is selected, the default is 9. For WAVE, the default is 2.1π.

SIGMA=*number*

specifies the fixed value of the scale parameter. By default, SIGMA is not used.

STD=*method*

specifies the standardization method to be applied to the target variables, where *method* can be the following:

- NONE | NO — Variables are not altered (default).
- STD — Variables are rescaled to a mean of 0 and a standard deviation of 1.
- RANGE | RAN — Variables are rescaled to a minimum of 0, and a range of 1.
- MIDRANGE | MID — Variables are rescaled to a midrange of 0 and a half-range of 1, that is a minimum of -1 and a maximum of 1.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

THAW Statement

THAW *weight-list*;

- The THAW statement thaws (unfreezes) weights.
- The required weight list itemizes the weights to be thawed.
- Specifying an unfrozen weight in the weight list does not cause a problem. The weight is ignored.

37

The THAW statement undoes the FREEZE action. That is, it thaws all of the weights that are specified in the (***required***) *weight-list* argument. If a specified weight is not frozen, the THAW statement is ignored.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

TRAIN Statement

```
TRAIN OUT=<libref.>SAS-data-set
      OUTEST=<libref.>SAS-data-set
      OUTFIT=<libref.>SAS-data-set
      <ACCEL|ACCELERATE=number>
      <DECEL|DECELERATE=number>
      <DUMMIES | NODUMMIES>
      <ESTITER=i>
      <LEARN=number>
      <MAX|MAXMOMENTUM=number>
      <MAXITER=integer>
      <MAXLEARN=number>
      <MAXTIME=number>
      <MINLEARN=number>
      <MOM|MOMENTUM=number>
      <TECHNIQUE=name>;
```

38

As its name suggests, the TRAIN statement trains the network. The following options are available:

ACCELERATE | ACCEL=*number*

specifies the rate of increase of learning for the RPROP optimization technique. The value of *number* must be greater than 1. The default is 1.2.

DECELERATE | DECEL=*number*

specifies the rate of decrease of learning for the RPROP optimization technique. The value of *number* must be greater than 0 and less than 1. The default is 0.5.

DUMMIES | NODUMMIES

specifies whether to write dummy variables to the OUT= data set. The default is NODUMMIES.

ESTITER=*i*

writes only initial and final weights to the OUTEST= data set when $i = 0$ (the default). If $i > 0$, PROC NEURAL writes weights after every *i* iterations, as well as the final weights, to OUTEST=.

LEARN=*number*

specifies either the learning rate for BPROP or the initial learning rate for QPROP and RPROP. (The default is 0.1.) The value of *number* must be greater than 0.

MAXITER=*integer*

specifies the maximum number of iterations. The default depends on the optimization method.

- For TRUREG, NRRIDG, NEWRAP, and LEVMAR, the default is 100 iterations.
- The QUANEW and DBLDOG default is 200 iterations.
- CONGRA sets the default at 400 iterations.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

- BPROP, RPROP, and QPROP default to 1000 iterations.

MAXLEARN=*number*

specifies the maximum learning rate for RPROP. The value of *number* must be greater than 0. The default is 1/MACSQRTEPS.

MAXMOMENTUM | MAXMOM=*number*

specifies the momentum for BPROP. The value of *number* must be greater than 0. The default is 1.75.

MAXTIME=*number*

specifies the amount of time after which training stops. The default is 604800 seconds.

MINLEARN=*number*

specifies the minimum learning rate for RPROP. The value of *number* must be greater than 0. The default is MACSQRTEPS.

MOMENTUM | MOM=*number*

specifies the momentum for BPROP. For RPROP, the default is 0.1. For BPROP, it is 0.9. The value of *number* must be greater than or equal to 0 and less than 1.

OUT=<*libref.*>*SAS-data-set*

specifies the output data set that contains the outputs.

OUTEST=<*libref.*>*SAS-data-set*

specifies the output data set that contains the network weights.

OUTFIT=<*libref.*>*SAS-data-set*

specifies the output data set that contains the fit statistics.

TECHNIQUE=*name*

specifies the optimization technique to be applied, where *name* is one of the following values:

- TRUREG (trust region)
- LEVMAR (Levenberg-Marquardt)
- DBLDOG (double dogleg)
- QUANEW (quasi-Newton)
- CONGRA (conjugate gradient)
- BPROP (standard backpropagation)
- RPROP (resilient backpropagation)
- QPROP (quick backpropagation)

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

USE Statement

USE *<libref.>SAS-data-set*;

- The USE statement sets the network's weights to the values that are provided in the specified SAS data set.
- Because it does not randomly generate weights, the data set must contain ***all*** of the required weights.

39

The USE statement sets all weights to values from a data set. The ***required*** *<libref.>SAS-data-set* argument specifies an input data set that contains all the weights.

Unlike the INITIAL statement, the USE statement does ***not*** generate any random weights. Therefore, the data set must contain all of the network weights and parameters.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Details of Activation and Combination Functions

Activation Function	Range	Definition as a Function of x
IDENTITY	(-∞ : ∞)	x
LINEAR	(-∞ : ∞)	x
EXPONENTIAL	(0 : ∞)	e^x
RECIPROCAL	(0 : ∞)	$\frac{1}{x}$
SQUARE	(0 : ∞)	x^2
LOGISTIC	(0 : 1)	$\frac{1}{1+e^{-x}}$
MLOGISTIC	(0 : 1)	$\frac{e^x}{\sum_j \text{exponentials}}$
SOFTMAX	(0 : 1)	$\frac{e^x}{\sum_j \text{exponentials}}$
GAUSS	(0 : 1]	e^{-x^2}
SINE	(0 : 1]	sin(x)
COSINE	(0 : 1]	cos(x)
ELLIOT	(-1 : 1)	$\frac{x}{1+\lvert x\rvert}$
TANH	(-1 : 1)	$1-\frac{2}{1+e^{2x}}$
ARCTAN	(-1 : 1)	$\frac{2}{\pi}\tan^{-1}(x)$

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Combination Function	Definition
ADD	$\sum_i x_i$
LINEAR	$bias_j + \sum_i w_{ij} x_i$
EQSLOPES	$bias_j + \sum_i w_i x_i$
XRADIAL	$F \cdot \log(alt_j) - bias_j^2 \cdot \sum_i \left(w_{ij} - x_i\right)^2$
EHRADIAL	$bias_j^2 \cdot \left(w_{ij} - x_i\right)^2$
EVRADIAL	$F \cdot \log(bias_j) - bias_j^2 \cdot \left(w_{ij} - x_i\right)^2$
EWRADIAL	$F \cdot \log(alt_j) - bias_j^2 \cdot \left(w_{ij} - x_i\right)^2$
EQRADIAL	$-bias_j^2 \cdot \left(w_{ij} - x_i\right)^2$
RADIAL	$bias_j^2 \cdot \left(w_{ij} - x_i\right)^2$

where

- alt_j is the altitude of the jth unit.
- $bias_j$ is the bias (width) of the jth unit.
- F is the number of other units that feed into the current unit, not adjusted for altitude or bias.
- w_{ij} is the weight connecting the ith incoming value to the jth unit.
- x_i is the ith incoming value.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Appendix B Empirical Partial Residuals

B.1 Empirical Partial Residuals B-3

Demonstration: Illustrating Empirical Partial Residuals B-4

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

B.1 Empirical Partial Residuals

Objectives

- Illustrate empirical partial residuals.

2

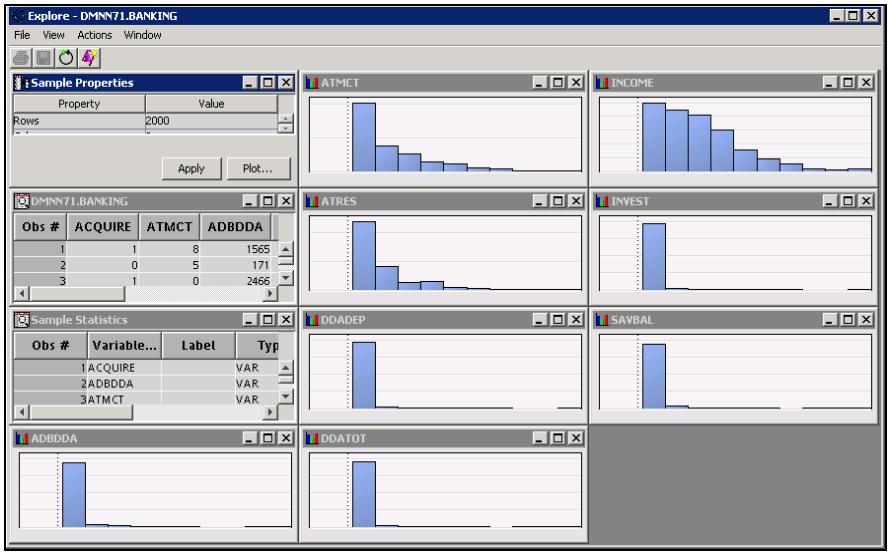
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Illustrating Empirical Partial Residuals

b1s1d1

This demonstration illustrates the generation of empirical partial residual plots to guide variable selection. A generalized additive neural network (GANN) modeling a discrete (binary) target is used. If the target variable is discrete, partial residuals can only be approximated. Empirical partial residuals are an alternative. The banking data table (**BANKING**), which consists of 2000 banking customers who had an opportunity to acquire a new investment product, is again used. This table was previously defined.

1. Right-click **Diagrams** and select **Create Diagram**.
2. Enter **appendix B** in the **Diagram Name** field and click **OK**.
3. Because **BANKING** has already been defined, drag and drop it onto the new diagram.
4. Activate the **BANKING** data set's properties panel, and click the ellipsis (**…**) next to **Variables**.
5. Highlight all the input variables and click **Explore**.



All of the input variables are skewed, some severely. Transformation is implied.

6. Close the Explore and variables windows.
7. Add a **Transform Variables** node (Modify tab) to the flow, and connect it to **BANKING**.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

8. Change the default transformation method for interval variables to **Maximum Normal**.

Default Methods	
Interval Inputs	Maximum Normal
Interval Targets	None
Class Inputs	None
Class Targets	None
Treat Missing as Level	No

9. Right-click the **Transform Variables** node and select **Run** from the pop-up menu. Access the results when they become available.

10. Maximize the Output window and scroll to line **22** to see the recommended transformations.

```
Computed Transformations
(maximum 500 observations printed)

Input              Input
 Name     Role      Level         Name        Level                    Formula

ADBDDA    INPUT    INTERVAL    LOG_ADBDDA    INTERVAL    log(max(ADBDDA--352, 0.0)/288807  + 1)
ATMCT     INPUT    INTERVAL    SQRT_ATMCT    INTERVAL    sqrt(max(ATMCT-0, 0.0)/30)
ATRES     INPUT    INTERVAL    LOG_ATRES     INTERVAL    log(max(ATRES-1, 0.0)/46  + 1)
DDADEP    INPUT    INTERVAL    LOG_DDADEP    INTERVAL    log(max(DDADEP-0, 0.0)/222744  + 1)
DDATOT    INPUT    INTERVAL    LOG_DDATOT    INTERVAL    log(max(DDATOT-0, 0.0)/269187  + 1)
INCOME    INPUT    INTERVAL    PWR_INCOME    INTERVAL    (max(INCOME-35, 0.0)/1465)**0.25
INVEST    INPUT    INTERVAL    LOG_INVEST    INTERVAL    log(max(INVEST-0, 0.0)/300000  + 1)
SAVBAL    INPUT    INTERVAL    PWR_SAVBAL    INTERVAL    (max(SAVBAL-0, 0.0)/259585)**0.25
```

11. Close the Transform Variables node's Results window.

12. Add a **SAS Code** node (Utility tab) and connect it to the **Transform Variables** node.

13. Activate the SAS Code node's properties panel and click the ellipsis (**...**) next to **Code Editor**.

14. In the (default) Training Code editor, right-click in the white space and select **Open**.

15. Select the program **b1s1d1.sas** (in d:\workshop\winsas\dmnn41).

    First, the GANN is defined. It is virtually identical to the Chapter 4 GANN except that the target is now binary. After initializing the options, the program defines two SYMBOL statements.

```
options nodate nonumber;
goptions reset=all device=actximg;

symbol1 c=b v=circle i=none;
symbol2 c=bl v=none i=join;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Then the PRLOGIT macro definition begins. The macro function takes only one argument (**dsn**), which specifies the name of the data set to be analyzed.

```
%macro prlogit(dsn=);
```

Next, two macro variables are defined. The first (**ninputs**) holds the count of the number of inputs in the **%EM_INTERVAL_INPUT** macro variable, which points to the interval inputs. The second creates the predictor variable from the target. In this case, the predictor created is **p_acquire1**.

```
data _null_;
   call symputx('ninputs',length("%cmpres(%EM_INTERVAL_INPUT)")-
         length(compress("%cmpres(%EM_INTERVAL_INPUT)"))+1);
   call symputx("predicted",%str("p_%EM_TARGET")||'1');
run;
```

In order to properly prune the mini-networks, you need to know which mini-network corresponds to which input variable. To see the order of the inputs as they appear in the **%EM_INTERVAL_INPUT** macro variable, a %PUT statement has been added. This will list the inputs in the LOG file.

```
   %put %EM_INTERVAL_INPUT;
```

Then the input data source pointed to by **dsn** is converted to a DMDB and cataloged. Because the target is discrete (binary/nominal), it is declared in a CLASS statement. Notice, too, that the DESC (descending) descriptor is added to show that the one event is to be modeled.

```
proc dmdb data=&dsn dmdbcat=cat;
   class %EM_TARGET(desc);
   var %EM_INTERVAL_INPUT;
run;
```

The generalized additive neural network (GANN) is created next. Given that the components of the GANN were described in detail in Chapter 4, they are not outlined again here.

```
title1 h=1.5 f=swissb 'Generalized Additive Neural Network';
proc neural data=&dsn ranscale=0.1 dmdbcat=cat;
   target %EM_TARGET/level=nom id=out;
   %do i=1 %to &ninputs;
      input %scan(%EM_INTERVAL_INPUT,&i)/level=int std=std id=x&i;
      hidden 1/act=identity comb=lin nobias id=f&i;
      connect x&i f&i out;
   %end;
   initial;
   %do i=1 %to &ninputs;
      freeze f&i->out/value=1;
   %end;
   train;
   hidden 1 / id=h1;
   connect x1 h1 f1;
   hidden 1 / id=h2;
   connect x2 h2 f2;
   hidden 1 / id=h3;
   connect x3 h3 f3;
   hidden 1 / id=h4;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
   connect x4 h4 f4;
   hidden 1 / id=h5;
   connect x5 h5 f5;
   hidden 1 / id=h6;
   connect x6 h6 f6;
   hidden 1 / id=h7;
   connect x7 h7 f7;
   hidden 1 / id=h8;
   connect x8 h8 f8;
   prelim 5 maxiter=10;
   train;
   score data=&dsn out=results;
run;
```

Because the **BANKING** data is oversampled, the misclassification rate will misrepresent the model's fit unless it is corrected using the prior probability information. An alternative is to assess the model based on the ROC index. The ROC index (C statistic) is insensitive to oversampling.

```
title2 'Area Under the ROC Curve (C Statistic)';
proc rank data=results out=rankings;
   var &predicted;
   ranks rp;
run;
proc sql;
   select sum(%EM_TARGET=1) as n1,
      (sum(rp*(%EM_TARGET=1))-.5*(calculated n1)*(calculated n1+1))
       /((calculated n1)*(count(%EM_TARGET)-(calculated n1))) as c
   from rankings;
quit;
```

The major new component of the program is the empirical partial residual calculation, shown below:

```
%do i=1 %to &ninputs;
   %let var=%scan(%EM_INTERVAL_INPUT,&i);
   proc rank data=results groups=100 out=binned;
      var &var;
      ranks bin;
   run;
   proc sql;
      create table bins as
      select count(%EM_TARGET) as tot,
         sum(%EM_TARGET) as m,
         mean(&predicted) as &predicted,
         mean(&var) as &var,
         mean(f&i.1) as f&i.1,
         log((calculated m+1)/(calculated tot-calculated m+1))
            as elogit,
         calculated elogit-log(calculated &predicted
            /(1-calculated &predicted))+calculated f&i.1 as parres
      from binned
      group by bin;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

```
   quit;
   title2 'Empirical Partial Residual Plot';
   proc gplot data=bins;
      plot parres*&var=1 f&i.1*&var=2/overlay;
   run;
%end;
```

First, a %DO loop pulls off each input and stores it as the macro variable (**var**). Then PROC RANK bins the input variable into 100 equal groups. Next, PROC SQL creates the **bins** table, containing the mean input variable (**&v**), mean univariate function (**&f**), and empirical partial residual for each bin (**parres**). The empirical logit (**elogit**) is calculated as the logit of the proportions of events in each bin, and the calculated partial residuals are plotted against the mean input.

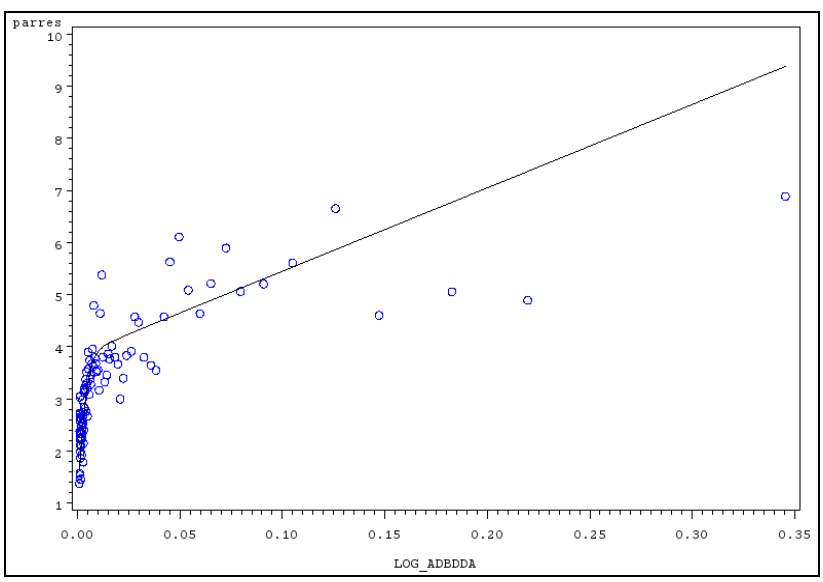
16. Right-click the running man icon to run the diagram from within the SAS Code node.
17. Confirm the run and save the program. When training is complete, access the results.
18. Scroll to the bottom of the Output window (line **364**) to see the ROC index value that results when all of the inputs have 4-degree-of-freedom mini-networks.

```
Generalized Additive Neural Network
Area Under the ROC Curve (C Statistic)

      n1          c
------------------
    1000     0.9168
```

The ROC index, which ranges from 0.5 to 1, is extremely high (92%), suggesting that the current GANN provides a very good fit to the **BANKING** training data.

19. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** to inspect the partial residual plots.

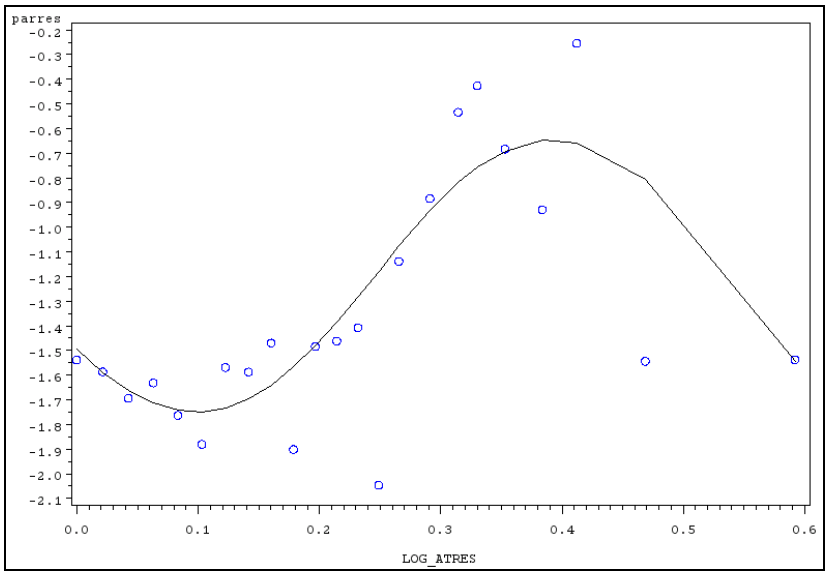
The first plot (below) displays the empirical partial residuals for the transformed **ADBDDA** variable.



The 4-degree-of-freedom mini-network appears to fit the data fairly well. No change to the 4-df mini-network would seem to be implied.

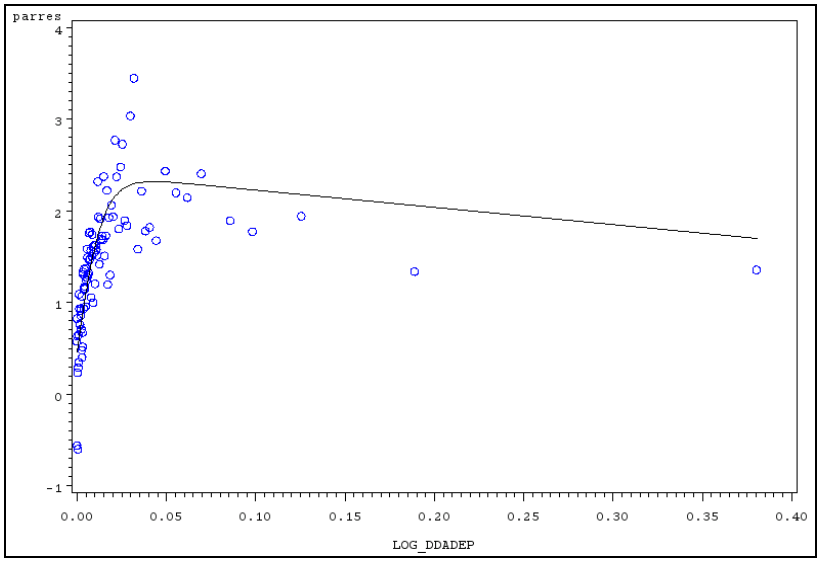
20. Click the right arrow to see the empirical partial residual plot of the transformed **ATRES** variable.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Again the fit appears to be a pretty good one. No change to its mini-network is implied.

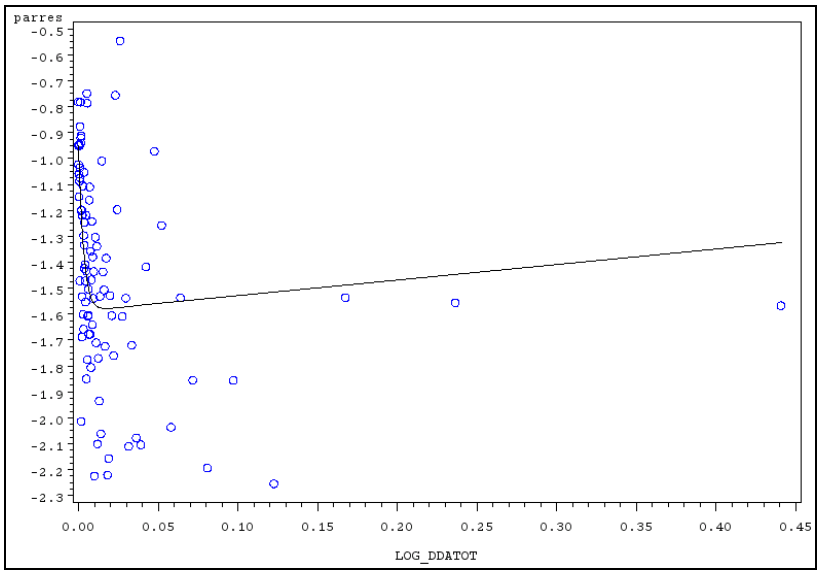
21. Click the right arrow to see the empirical partial residual plot of the transformed **DDADEP** variable.



This variable would seem to be a candidate for removal. Not only is the transformed value still heavily skewed, the slope of the fit at larger values of **DDADEP** is virtually zero (null model).

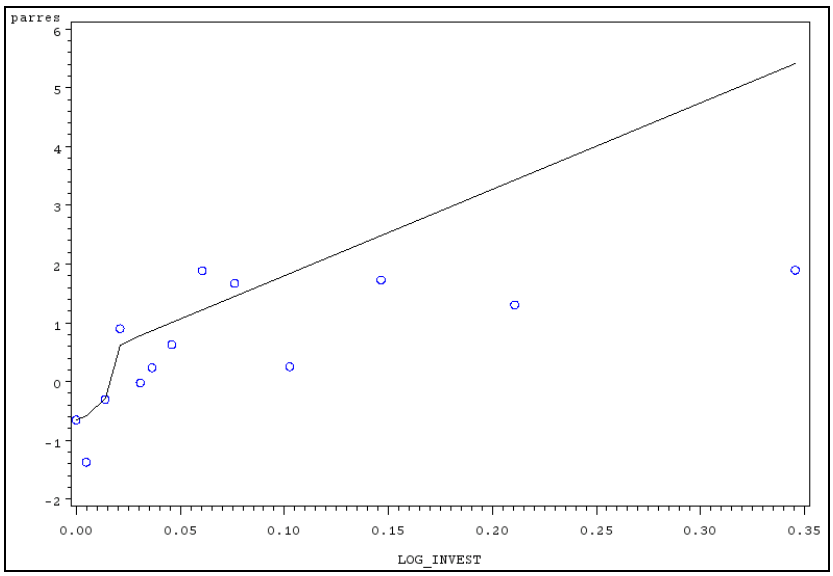
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

22. Click the right arrow button again to see the plot of the transformed **DDATOT** variable.



This variable would also seem to be a candidate for removal. As with the **DDADEP** variable, the transformed value is still heavily skewed. And the slope of the network's fit at higher values of the transformed **DDATOT** variable is virtually zero (null model).

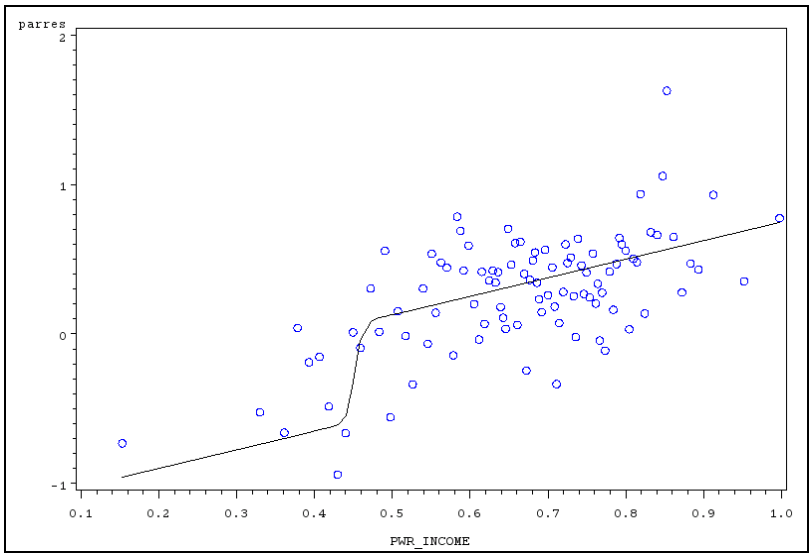
23. Click the right arrow to see the empirical partial residual plot of the transformed **INVEST** variable.



The transformed **INVEST** variable does not appear to bend enough to pass through the residuals at the larger magnitudes. Perhaps adding a second hidden unit to its mini-network will help.

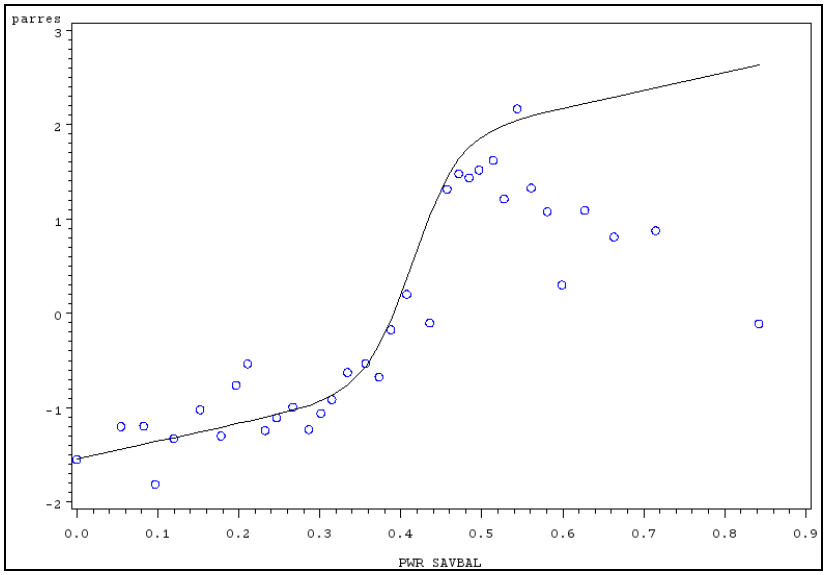
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

24. Click the right arrow again to see the plot of the **INCOME** variable.



There seems little need for the extra flexibility introduced by the hidden unit in the transformed **INCOME** variable's mini-network. Linearization (removal of the hidden unit) is implied.

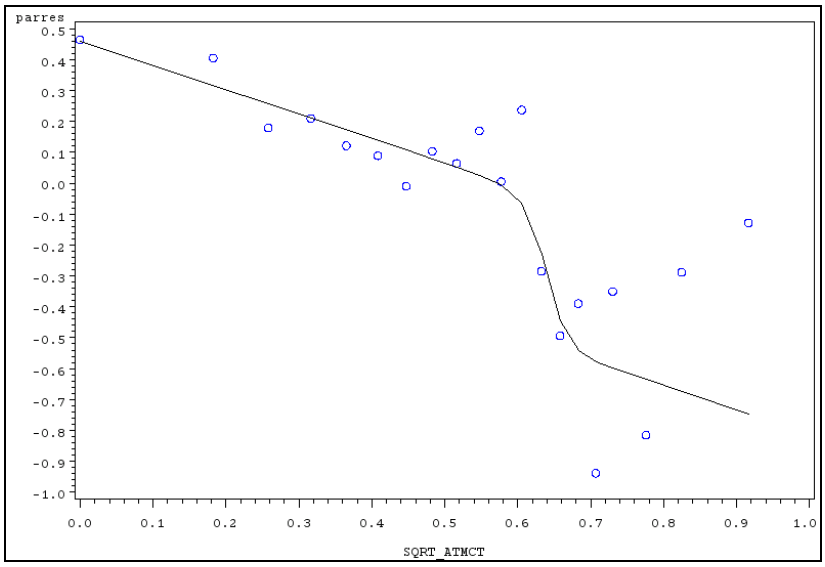
25. Click the right arrow again to see the plot of the transformed **SAVBAL** variable.



The current mini-network fails to catch the downward movement at larger values of the transformed **SAVBAL** variable. Augmenting its mini-network by adding a second hidden unit (increasing the mini-network to 7 degrees of freedom) should provide the needed flexibility.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

26. Click the right arrow one last time to see the plot of the transformed **ATMCT** variable.



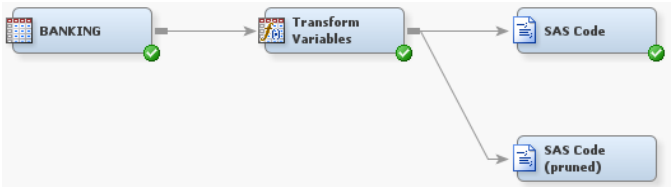
There seems little need for the extra flexibility introduced by the hidden unit in the transformed **ATMCT** variable's mini-network. Linearization (removal of the hidden unit) is implied.

27. Close the Train Graphs and the Results windows.

28. Also close the SAS Code node.

29. Add another **SAS Code** node (Utility tab), and connect it to the **Transform Variables** node.

30. Right-click the new node and rename it **SAS Code (Pruned)**. Your diagram should now look similar to the one shown below.



31. To remove the transformed **DDADEP** and **DDATOT** variables entirely, click the ellipsis (**…**) next to **Variables** in the new SAS Code (Pruned) node's properties panel, and set the Use status of both the **LOG_DDADEP** and **LOG_DDATOT** variables to **No**.

Name	Use	Report	Role	Level
ACQUIRE	**Default**	**No**	Target	Binary
LOG_ADBDDA	**Default**	**No**	Input	Interval
LOG_ATRES	**Default**	**No**	Input	Interval
LOG_DDADEP	**No**	**No**	Input	Interval
LOG_DDATOT	**No**	**No**	Input	Interval
LOG_INVEST	**Default**	**No**	Input	Interval
PWR_INCOME	**Default**	**No**	Input	Interval
PWR_SAVBAL	**Default**	**No**	Input	Interval
SQRT_ATMCT	**Default**	**No**	Input	Interval

32. Click **OK** to save the changes.

33. Click the ellipsis (**…**) next to the **Code Editor** field.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

34. In the (default) Training Code editor, right-click in the white space and select **Open**.

35. Read in the program **b1s1d1 (pruned)**.

    The program is identical to the one listed above except that hidden units **h7** and **h8**, corresponding to the two deleted inputs **(LOG_DDADEP** and **LOG_DDATOT**), and their direct connections have not been created. The **PWR_INCOME** and **SQRT_ATMCT** variables have also been linearized by commenting out the addition of hidden units **h4** and **h6** and the number of hidden units in the mini-networks associated with **LOG_INVEST** and **PWR_SAVBAL** was increased to **2**. (See below.)

```
      hidden 1/id=h1;      *LOG_ADBDDA;
      connect x1 h1 f1;

      hidden 1/id=h2;      *LOG_ATRES;
      connect x2 h2 f2;

      hidden 2/id=h3;      *LOG_INVEST;
      connect x3 h3 f3;

*     hidden 1/id=h4;      *PWR_INCOME;
*     connect x4 h4 f4;

      hidden 2/id=h5;      *PWR_SAVBAL;
      connect x5 h5 f5;

*     hidden 1/id=h6;      *SQRT_ATMCT;
*     connect x6 h6 f6;
```

36. Run the flow using the pruned GANN from within the SAS Code (pruned) node.

37. Confirm the run and examine the results when they become available.

38. In the Output window, scroll to line **333** to see the ROC index for the pruned GANN.

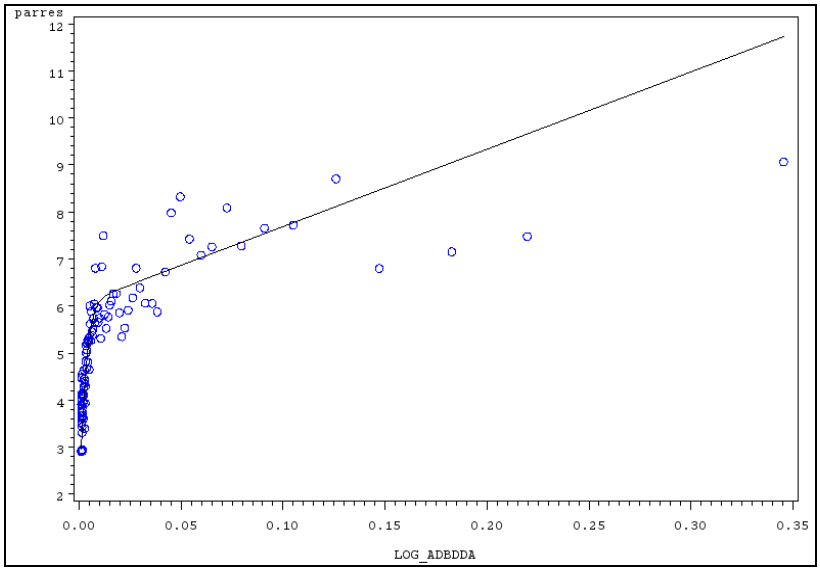
```
Generalized Additive Neural Network
Area Under the ROC Curve (C Statistic)

      n1         c
------------------
    1000  0.911176
```

    The ROC index has gone down, but only slightly (from 0.9168 to 0.911176). Considering the substantial simplification that the GANN underwent, this loss seems an equitable tradeoff.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

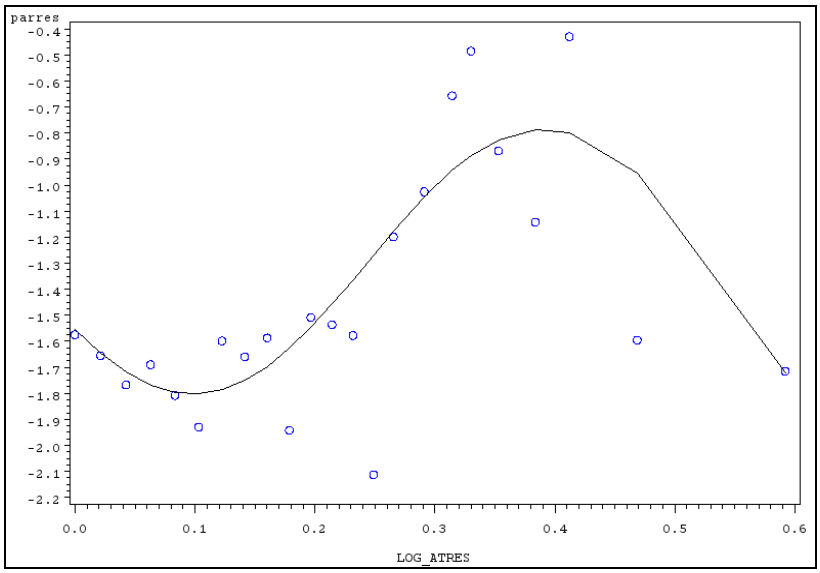
39. Select **View** ⇨ **SAS Results** ⇨ **Train Graphs** to inspect the resulting empirical partial residual plots.

    The first plot is for the transformed **ADBDDA** variable. Given that no change was made to its mini-network, it is not surprising that the fit is the similar to the one shown before.



The empirical partial residual plot for a given variable can change, even if its associated function (mini-network) is unaltered. That is, changing the mini-network associated with another variable can result in a change to the fit of a variable whose network is unaltered.

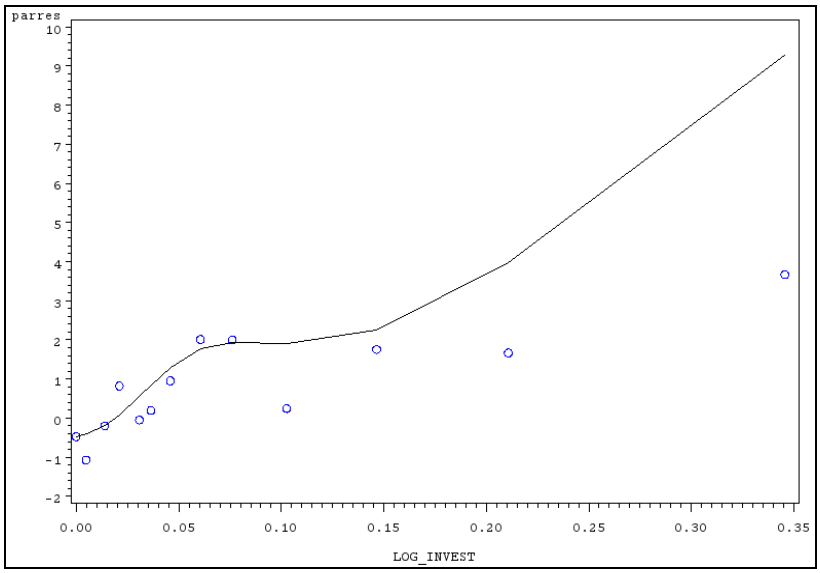
40. Click the right arrow (second from the right at the bottom of the page) to see the next plot.



Again, no change to the transformed **ATRES** variable's mini-network was proposed, so the resulting empirical partial residual plot is similar to the one shown previously.

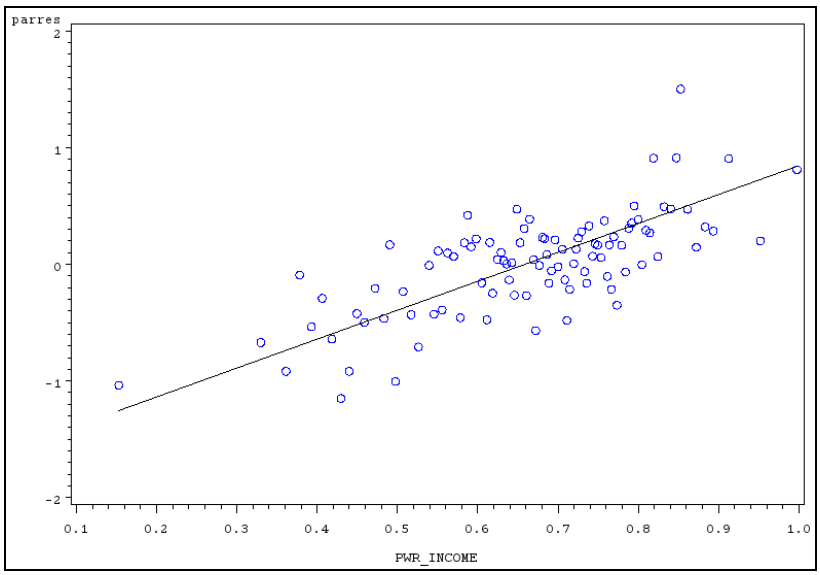
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

41. Click the right arrow to see the plot of the transformed **INVEST** variable.



The mini-network associated with this variable was augmented with a second hidden unit. The resulting model (arguably) seems to provide a better fit to the empirical partial residuals.

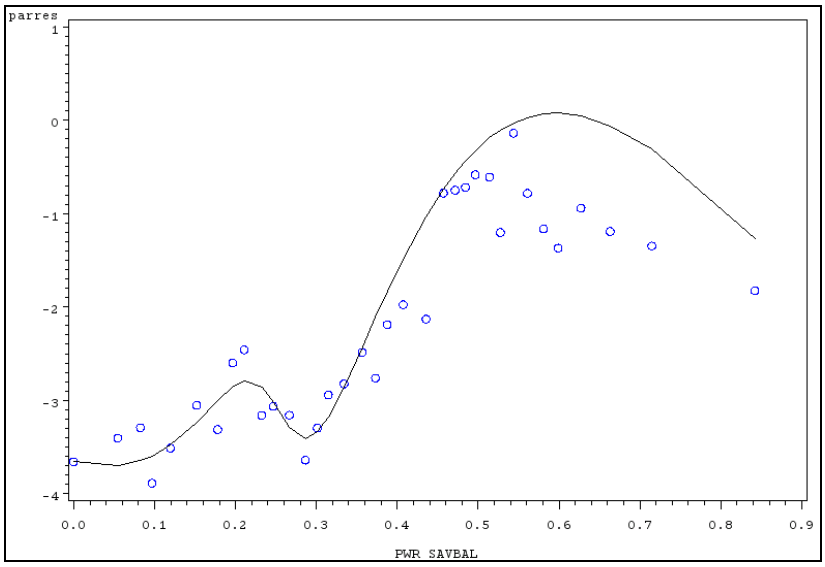
42. Click the right arrow again to display the plot of the linearized **INVEST** variable.



The decision to use only a direct (linear) connection to model the relationship between the transformed investment variable and banking product acquisition seems a good one.

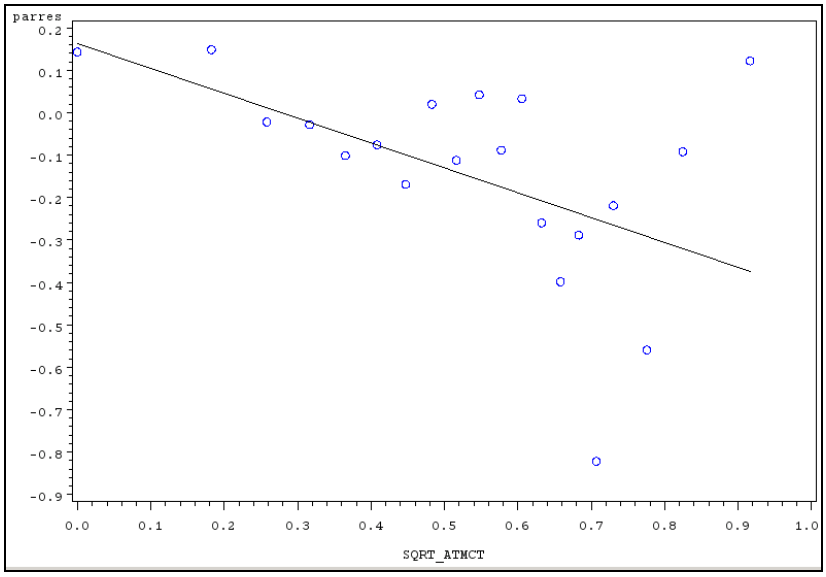
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

43. Click the right arrow again to see the partial residual plot of the augmented **SAVBAL** variable.



The addition of a second hidden unit to the **SAVBAL** variable's mini-network has resulted in a second rise in the network's fit line. As this second rise does seem to correspond to a rise in the data, it would appear that the addition of the second hidden unit was appropriate.

44. Click the right arrow again to display the pruned GANN's final empirical partial residual plot.



This is the plot of the transformed **ATMCT** variable. Its mini-network was linearized. Although the increase variation at large values of the **ATMCT** variable seems to suggest a degree of nonlinearity, given that the variation is both above and below the line, the choice of a linear model seems sound.

45. Close the Report Graphs window.

46. Close the "appendix B" diagram.

End of Demonstration

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Appendix C References

C.1 References C-3

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

C.1 References

Alpaydin, E. and F. Gürgen. 1996. "Comparison of Statistical and Neural Classifiers and Their Applications to Optical Character Recognition and Speech Classification." in *Neural Network Systems: Techniques and Applications* (Leondes, Ed.). San Diego, CA: Academic Press.

Banks, R. B. 1994. *Growth and Diffusion Phenomena: Mathematical Frameworks and Application*. New York: Springer-Verlag.

Bartlett, P. L. 1997. "For Valid Generalization, the Size of the Weights is More Important than the Size of the Network." in: *Advances in Neural Information Processing Systems Volume 9* (Mozer, Jordan, and Petsche, Eds.). Cambridge, MA: The MIT Press.

Berk, K. N. and D. E. Booth. 1995. "Seeing a Curve in Multiple Regression." *Technometrics* 37:4.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. New York: Oxford University Press.

Blake, C., E. Keogh, and C. J. Merz. 1998. UCI Repository of machine learning databases. Irvine, CA: University of California. Department of Information and Computer Science.

Box, G. E. P. and G. M. Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day Inc.

Broomhead, D. S. and David Lowe. 1988. *Radial basis functions, multi-variable functional interpolation and adaptive networks* (Technical report). RSRE. 4148.

Broyden, C. G. 1970 "The Convergence of a Class of Double-rank Minimization Algorithms." *Journal of the Institute of Mathematics and Its Applications* 6:76–90.

Bryson, A. E. and Y. C. Ho. 1969. *Neural Networks: Computers with Intuition*. Singapore: World Scientific.

Cai, Z. and C. L. Tsai. 1999. "Diagnostics for Nonlinearity in Generalized Linear Models." *Computational Statistics and Data Analysis* 29:445–469.

Carroll, R. J. and D. Ruppert. 1988. *Transformation and Weighting in Regression*. New York: Chapman & Hall.

Chatfield, C. and J. Faraway. 1998. "Time Series Forecasting with Neural Networks: A Comparative Study Using the Airline Data." *Applied Statistics* 47:231–250.

Chester, D.L.1990. "Why Two Hidden-Layers are Better than One" in IJCNN-90-WASH-DC. Lawrence Erlbaum. Vol. 1:265–268.

Cuellar, M.P., M. Delgado, and M.C. Pegalajar. 2006. "An Application of Non-Linear Programming to Train Recurrent Neural Networks in Time Series Prediction Problems." Enterprise Information Systems VII (Springer Netherlands) 95–102.

Cybenko, G. 1988. *Continuous Valued Neural Networks with Two Hidden Layers Are Sufficient*. Technical Report. Dept. of Computer Science. Tufts University: Medford, MA.

David Shepard Associates. 1999. *The New Direct Marketing: How to Implement a Profit-Driven Database Marketing Strategy*. New York: McGraw-Hill.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

De Vries, B. and J. C. Principe. 1992. “The Gamma Filter - A New Class of Adaptive IIR Filters with Restricted Feedback.” *IEEE Transactions.* Signal Processing. In Press.

De Vries, B. and J. C. Principe. 1992. “The Gamma Model - A New Model for Temporal Processing.” *Neural Networks* Vol.5:565–576.

De Vries, B. and J. C. Principe. 1992. *Short Term Memory Structures for Dynamic Neural Networks*. Princeton, NJ: David Sarnoff Research Center.

Dorffner, G. 1996. “Neural Networks for Time Series Processing.” *Neural Network World* 6:447–468.

Efron, B. 1983. “Estimating the Error Rate of a Prediction Rule: Improvement on Cross Validation.” *Journal of the American Statistical Association* 78:316–331.

Ellingsen, B.K. 1994. “A Comparative Analysis of Backpropagation and Counterpropagation Neural Networks.” Department of Information Science. University of Bergen: Bergen, Norway.

Elman, J.L. 1990. “Finding structure in Time.” *Cognitive Science* 14:179–221.

Ezekiel, M. 1924. “A method for handling Curvilinear Correlation for any Number of Variables.” *Journal of the American Statistical Association* 19:431–453.

Falhman, S. E. 1988 “Faster-learning variations on back-propagation: an empirical study.” In D. Touretzky, G. E. Hinton, and T. J. Sejnowski (Eds.). *Proceedings of the 1988 Connectionist Models Summer School*. pp. 31–51. San Mateo, CA. Morgan Kaufmann.

Fahlman, S. E. and C. Lebiere. 1990. “The Cascade-Correlation Learning Architecture.” in *Advances in Neural Information Processing Systems Volume 2*. (D Touretzky, Ed.). San Mateo, CA: Morgan Kaufmann.

Fan, J. and I. Gijbels. 1996. *Local Polynomial Modeling and its Applications*. New York: Chapman & Hall.

Fletcher, R. 1987. *Practical Methods of Optimization*. New York: Wiley.

Fletcher, R. 1970. “A New Approach to Variable Metric Algorithms.” *Computer Journal* 13: 317–322.

Friedman, J. H. 1991. “Multivariate Adaptive Regression Splines (with discussion).” *Annals of Statistics* 19:1–141.

Friedman, J. H. 1994. “An Overview of Predictive Learning and Function Approximation.” in *From Statistics to Neural Networks. Theory and Pattern Recognition Applications*. (Cherkasy, Friedman, Wechsler, eds.) New York: Springer-Verlag.

Friedman, J. H. 1997. “On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality.” *Data Mining and Knowledge Discovery* 1:55–77.

Friedman, J. H. and W. Stuetzle 1981. “Projection Pursuit Regression.” *Journal of the American Statistical Association* 76:817–823.

Goldfarb, D. 1970. “A Family of Variable Metric Updates Derived by Variational Means.” *Mathematics of Computation* 24: 23–24.

Harrison, D. and D. L. Rubinfeld. 1978. “Hedonic Prices in the Demand for Clean Air.” *Journal of Environmental Economics and Management* 5:81–102.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Hand, D. J. 1997. *Construction and Assessment of Classification Rules*. New York: Wiley.

Harrell F. E., K. L. Lee, and D. B. Mark. 1996. "Multivariate Prognostic Models: Issues in Developing Models, Evaluating Assumptions and Adequacy, and Measuring and Reducing Errors." *Statistics in Medicine* 15:361–387.

Hassibi, B. and D. G. Stork. 1993. "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon." in *Advances in Neural Information Processing Systems, Volume 5*. (Hansen, Cowan, and Giles, eds.) San Mateo, CA: Morgan Kaufmann.

Hastie, T. J. and R. J. Tibshirani. 1986. "Generalized Additive Models (with discussion)." *Statistical Science* 1:297–318.

Hastie, T. J. and R. J. Tibshirani. 1990. *Generalized Additive Models*. New York: Chapman & Hall.

Hebb, D. O. 1949. *The Organization of Behavior*. New York: Wiley.

Hecht-Nielsen, R. 1987. "Counterpropagation Networks." *Applied Optics* 26:4979–4984.

Hertz, J., A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley Publishing Co.

Hinton, G. E., S. Osindero. and Y. W. Teh. 2006. "A Fast Learning Algorithm for Deep Belief Networks." *Neural Computation 18*. MIT. 1527–1554.

Hinton, G. E., N. Shrivastava, and K. Swersky. 2013. "Overview of Mini-Batch Gradient Descent" Video from Coursera - University of Toronto - Course: Neural Networks for Machine Learning: Published on Nov 5, 2013. https://www.coursera.org/course/neura....

Hoerl, A. E. and R. W. Kennard. 1970. "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics* 12:55–67.

Hogg, R. V. 1979. "Statistical Robustness: One View of Its Use in Applications Today." *The American Statistician* 33:108–115.

Holden, K. 1995. "Vector Autoregression Modeling and Forecasting." *Journal of Forecasting* 14:159–166.

Holland, J. H. 1976. "Studies of Spontaneous Emergence of Self-Replicating Systems Using Cellular Automata and Formal Grammars." in *Automata, Languages, Development*. (A. Lindenmayer and G. Rozenberg, eds.) Amsterdam: North-Holland.

Holland, J. H. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, 2nd Edition*. Cambridge, MA: The MIT Press.

Huber, P. J. 1964. "Robust Estimation of a Location Parameter." *Annals of Mathematical Statistics* 35:73–101.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton. 1991. "Adaptive Mixture of Local Experts." *Neural Computation* 3:79–87.

Jaeger, H. 2001. "The 'echo state' approach to analyzing and training recurrent neural networks". *GMD Report 148*. GMD - German National Research Institute for Computer Science.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Jaeger, H. 2002. “Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach”. *GMD Report 159*. Fraunhofer Institute AIS.

Jaeger, H. 2007. “Echo state network.” *Scholarpedia* 2(9):2330.

Johnson, D. E. 1998. *Applied Multivariate Methods for Data Analysis*. Pacific Grove, CA: Druxbury Press. Cole Publishing Company.

Johnson, G. 1991. *In the Palaces of Memory*. Vintage Books: Random House, Inc. NY, NY.

Jordan, M. I. and R. A. Jacobs. 1994. “Hierarchical Mixture of Experts and the EM algorithm.” *Neural Computation* 6:181–214.

Landwehr, J. D., D. Pregibon, and A. C. Shoemaker. 1984. “Graphical Methods for Assessing Logistic Regression Models (with discussion).” *Journal of the American Statistical Association* 79:61–83.

Larsen, W. A. and S. J. McCleary. 1972. “The Use of Partial Residual Plots in Regression Analysis.” *Technometrics* 14:781–790.

Le Cun, Y., P. Y. Simard, and B. Pearlmutter. 1993. “Autonomic learning rate maximization by on-line estimation of Hessian’s eigenvectors” in *Advances in Neural Information Processing Systems* (S. J. Hanson, J. D. Cowan, and C. L. Giles, eds.) Vol. 5:156–163. San Mateo, CA: Morgan Kaufmann.

Leisch, F., A. Trapletti, and K. Hornik. 1999. “Stationarity and Stability of Autoregressive Neural Network Processes.” in *Advances in Neural Information Processing Systems 11. (*Kearns, Solla, Cohn, eds.) Cambridge, MA: The MIT Press.

Lukosevicius, M. and H. Jaeger. 2010. “Reservoir Approaches to Recurrent Neural Network Training.” *Computer Science Revue* 3(3):127–149.

Lukosevicius, M. 2012. “A Practical Guide to Applying Echo State Networks.” In *Neural Networks: Tricks of the Trade, 2nd ed*. (G. Montavon, G. B. Orr, and K. R. Müller, eds.) Springer LNCS 7700:659–686.

Maass W., T. Natschlaeger, and H. Markram. 2002. “Real-time computing without stable states: A new framework for neural computation based on perturbations.” *Neural Computation*. 14(11):2531–2560.

Masters, T. 1993. *Practical Neural Network Recipes in C++*. Boston: Academic Press Inc. Harcourt Brace & Co. Publishers.

Melssen, W., R. Wehrens, and L. Buydens. 2006. “Supervised Kohonen networks for classification problems.” *Chemometrics and Intelligent Laboratory Systems* 83:99–113.

McCullagh, P. and J. A. Nelder. 1989. *Generalized Linear Models, Second Edition*. New York: Chapman & Hall.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley.

Minsky, Marvin. 1986. *The Society of Mind.* New York: Simon and Schuster.

Mitra, S. 2000. “Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework.” *IEEE Transactions on Neural Networks*. Vol. II, No 3 (May).

Moody, J. 1994. “Prediction Risk and Architecture Selection for Neural Networks.” in *From Statistics to Neural Networks. Theory and Pattern Recognition Applications*. (Cherkasy, Friedman, Wechsler, eds.) New York: Springer-Verlag.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Ng, A. 2013. “Stochastic Gradient Descent” Video from Coursera - Stanford University - Course: Machine Learning: Published on Nov 1, 2013. https://www.coursera.org/course/ml.

Olah, C. “Understanding LSTM Networks.” Available http://colah.github.io/posts/2015-08-Understanding-LSTMs. Posted Aug 27, 2015. Accessed March 7, 2016.

Pao, Y. 1989. *Adaptive Pattern Recognition and Neural Networks*. Reading, MA: Addison-Wesley.

Parker, D. B. 1985. *Learning Logic: Technical Report TR-47*. Center for Computational Research in Economics and Management Science. MIT. Cambridge, MA.

Pearlmutter, B. 1995. “Gradient calculations for dynamic recurrent neural networks: A survey.” *Neural Networks*. IEEE Transactions. 6(5):1212–1218.

Poh, H., J. Yao, and T. Jasic. 1998. “Neural networks for the Analysis and Forecasting of Advertising and Promotion Impact.” *International Journal of Intelligent Systems in Accounting, Finance, and Management* 7:253–268.

Potts, W. J. E. 1999. “Generalized Additive Neural Networks.” in *KDD-99 Proceedings.* (Chaudhuri, Madigan, eds.). ACM.

Prechelt, L. 1996. “A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice.” *Neural Networks* 9:457–462.

Prechelt, L. 1997. “Investigation of the CasCor Family of Learning Algorithms.” *Neural Networks* 10:885–896.

Prechelt, L. 1998. “Automatic Early Stopping Using Cross Validation: Quantifying the Criteria.” *Neural Networks* 11:761–767.

Principe, J.C., N. R. Euliano, and W. C. Lefebvre. 2000. *Neural and Adaptive Systems*. New York: Wiley.

Raftery, A. E. 1995. “Bayesian Model Selection in Social Research (with discussion).” in *Sociological Methodology 1995* (Marsden ed.) New York: Blackwell.

Riedmiller, M. and H. Braun. 1993. “A direct adaptive method for faster back propagation learning: The RPROP algorithm.” *Proceedings of the International Conference on Neural Networks*. San Francisco, CA. pp. 586–591.

Riedmiller, M. 1994. “Supervised Learning in Multi-layer Perceptrons – From Backpropagation to Adaptive Learning Algorithms.” *Int. Journal of Computer Standards and Interfaces* 16.

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks*. Cambridge University Press.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. “Learning Representations by Back-Propagating Errors.” *Nature* 323:533–536.

Runkle, D. E. 1987. “Vector Autoregressions and Reality (with discussion).” *Journal of Business and Economic Statistics* 5:437–454.

Sarle, W. S. 1994. “Neural Networks and Statistical Models.” *Proceedings of the Nineteenth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc.

Sarle, W. S. 1995. “Stopped Training and Other Remedies for Overfitting.” *Proceedings of the 27th Symposium on the Interface*.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Sarle, W. S., ed. 1997. Neural Network FAQ. URL: ftp://ftp.sas.com/pub/neural/FAQ.html.

Sarle, W. S. 1999. "How to Measure the Importance of Inputs?" URL: ftp://ftp.sas.com/pub/neural/importance.html.

Sarle, W. S. 2000. (revised June 23) "How to measure importance of inputs?" Cary, N.C. [http://mu.dmt.ibaraki.ac.jp/yanai/neu/faq/importance.html]

SAS Institute Inc. 1989. *SAS® Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 1990a. *SAS/GRAPH® Software: Reference, Version 6, First Edition, Volume 1*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 1990b. *SAS/GRAPH® Software: Reference, Version 6, First Edition, Volume 2*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 1990c. *SAS/STAT® User's Guide, Version 6, Fourth Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 1990d. *SAS® Procedures Guide, Version 6, Third Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 1993. *SAS/ETS® User's Guide, Version 6, Second Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc 1996. *SAS/OR® Technical Report: The NLP Procedure*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2008. *Applied Analytics Using SAS® Enterprise Miner™ 5.3 Course Notes*. Cary, NC: SAS Institute Inc.

Schwarz, G. 1978. "Estimating the Dimension of a Model." *Annals of Statistics* 6:461–464.

Seber, G. A. F. and C. J. Wild. 1989. *Nonlinear Regression*. New York: Wiley.

Shanno, D. F. 1970. "Conditioning of Quasi-Newton Methods for Function Estimation." *Mathematics of Computation* 24:647–656.

Suykens, J., B. D. Moor, and J. Vandewalle. 2008. "Toward optical signal processing using photonic reservoir computing." *Optics Express* 16(15):11182–11192.

Tibshirani, R. and G. E. Hinton. 1995 "Coaching variables for regression and classification." *Statistics and Computing* 8:25–33.

Tsukimoto, H. 2000. "Extracting Rules from Trained Neural Networks." *IEEE Transactions on Neural Networks*. Vol. II, No. 2 (March).

Vracko, M. 2005. "Kohonen Artificial Neural Network and Counterpropagation Neural Network in Molecular Toxicity Studies." *Current Computer-Aided Drug Design* 1:73–78.

Wasserman, P. D. 1993. *Advanced Methods in Neural Computing*. New York: Van Nostrand Reinhold.

Weigend, A. S. and N. A. Gershenfeld. 1994. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Reading, MA: Addison-Wesley.

Werbos, P. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. Thesis. Harvard University.

Zahavi, J. and N. Levin. 1997. "Applying Neural Computing to Target Marketing." *Journal of Direct Marketing* 11:5–22.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.