



THE
POWER
TO KNOW.

**SAS[®] Enterprise Miner[™] and
SAS[®] Text Miner Procedures
Reference for SAS[®] 9.1.3
The NEURAL Procedure
(Book Excerpt)**

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., SAS® *Enterprise Miner™* and SAS® *Text Miner Procedures: Reference for SAS® 9.1.3*, Cary, NC: SAS Institute Inc.

SAS® Enterprise Miner™ and SAS® Text Miner Procedures: Reference for SAS 9.1.3

Copyright © 2008 by SAS Institute Inc., Cary, NC, USA.

All rights reserved. Produced in the United States of America.

For a Web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

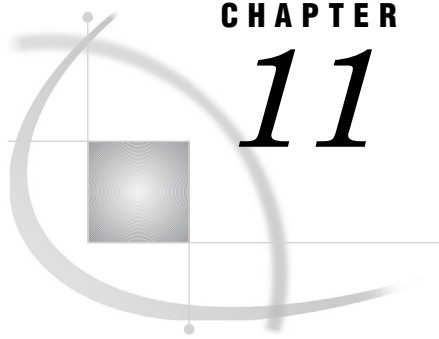
SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, October 2008

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/publishing or call 1-800-727-3228.

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 registered trademarks or trademarks of their respective companies.



CHAPTER

11**The NEURAL Procedure**

<i>Overview: NEURAL Procedure</i>	232
<i>Terminology and Architectural Description</i>	232
<i>Running the Neural Procedure</i>	232
<i>Syntax: NEURAL Procedure</i>	233
<i>PROC NEURAL Statement</i>	235
<i>ARCHITECTURE Statement</i>	237
<i>CODE Statement</i>	239
<i>CONNECT Statement</i>	241
<i>CUT Statement</i>	242
<i>DECISION Statement</i>	242
<i>DELETE Statement</i>	244
<i>FREEZE Statement</i>	244
<i>FREQ Statement</i>	245
<i>HIDDEN Statement</i>	245
<i>INITIAL Statement</i>	246
<i>INPUT Statement</i>	248
<i>NETOPTIONS Statement</i>	249
<i>NLOPTIONS Statement</i>	251
<i>PERTURB Statement</i>	256
<i>PRELIM Statement</i>	257
<i>QUIT Statement</i>	259
<i>RANOPTIONS Statement</i>	259
<i>SAVE Statement</i>	260
<i>SCORE Statement</i>	261
<i>SET Statement</i>	262
<i>SHOW Statement</i>	262
<i>TARGET Statement</i>	263
<i>THAW Statement</i>	265
<i>TRAIN Statement</i>	266
<i>USE Statement</i>	268
<i>ACTIVATION FUNCTIONS</i>	269
<i>COMBINATION FUNCTIONS</i>	269
<i>Details: NEURAL Procedure</i>	271
<i>The BPROP, RPROP, and QPROP Algorithms Used in PROC NEURAL</i>	271
<i>Introduction</i>	271
<i>Mathematical Notation</i>	272
<i>Standard Printing for the PROP Algorithms</i>	272
<i>The Standard Backprop Algorithm</i>	273
<i>The RPROP Algorithm</i>	274
<i>The Quickprop Algorithm</i>	275
<i>Examples: NEURAL Procedure</i>	276

Example 1: Developing a Simple Multilayer Perceptron (Rings Data) 276

Example 2: Developing a Neural Network for a Continuous Target 285

Example 3: Neural Network Hill-and-Plateau Example (Surf Data) 292

References 299

Overview: NEURAL Procedure

The NEURAL procedure trains a wide variety of feedforward neural networks using proven statistical methods and numerical algorithms. Please read the chapters “Introduction to Predictive Modeling” and “Neural Network Node: Reference” before reading this chapter. These chapters provide background information and details needed to understand the current chapter.

Terminology and Architectural Description

Each INPUT, HIDDEN and TARGET statement defines a “layer”. For the INPUT statement, a layer is a convenient grouping of variables, serving as inputs to the network, having common values for LEVEL and STD. Similarly, for the TARGET statement, a layer is a convenient grouping of variables, serving as outputs of the network, having common values for LEVEL, STD, ACTIVATION, COMBINATION, and other characteristics. Each layer consists of “units”. A unit is synonymous with the term “neuron” in the literature. It is the smallest computational entity in the network.

The INPUT and TARGET statements require a list of variables. If the variables are interval type, there is one unit corresponding to each variable. If the variables are nominal or ordinal, there is a unit for each level of each variable. The HIDDEN statement requires a number which determines the number of units in the associated layer. This layer is a grouping of units having common values for ACTIVATION, COMBINATION, and other characteristics.

Each INPUT statement produces an input layer. Because multiple INPUT statements are allowed, a network can have multiple input layers. However, connections from multiple input layers must be parallel; there cannot be serial connections between input layers. Similarly, multiple TARGET statements generate multiple output layers. The connections to multiple output layers must be in parallel; there cannot be serial connections between output layers. Hidden layers can be connected serially or in parallel.

Running the Neural Procedure

Before running the NEURAL procedure, you must run the DMDB procedure to create a DMDB catalog entry.

A typical application of the NEURAL procedure uses the following statements:

- A PROC NEURAL statement to specify the training set, DMDB catalog, and random number seed. If you specify a 0 or negative seed, the system clock is used to obtain a seed. This seed will be unavailable if it is ever necessary to reproduce the initial weights.
- One or more INPUT statements to specify an input layer, including the input variables and other layer characteristics.
- One or more HIDDEN statements to specify a hidden layer, including the number of hidden units and other characteristics.
- One or more TARGET statements to specify a target layer, including the target variables and other characteristics.

- An ID statement within the INPUT, HIDDEN, or TARGET statements. If not specified then a default ID is used. This default is I1, I2, .. for input layers, H1, H2, .. for hidden layers, and O1, O2, ... for output layers.
- One or more CONNECT statements to connect layers in the network if the default serial connections are not appropriate.
- A PRELIM statement to do preliminary training to avoid bad local optima.
- A TRAIN statement to train the network.
- One or more SCORE statements to create output data sets.

Two types of statements are used with the NEURAL procedure. All of the statements in the list above are action statements, which directly affect the network or directly produce output. There are also option statements (NETOPTIONS, NLOPTIONS, and RANOPTIONS) that set options for future use. Options specified in an action statement apply only to that statement and do not affect subsequent statements. For example, the default technique for least squares training is Levenberg-Marquardt (TECH=LEVLMAR). If you execute a TRAIN statement with the option TECH=CONGRA, conjugate gradient training will be used for that particular training run. If you then execute another TRAIN statement without a TECH= option, the technique will revert to the default value of TECH=LEVLMAR. But if you submit an NLOPTIONS statement with TECH=CONGRA, conjugate gradient training will be used for all subsequent TRAIN statements until you explicitly specify a different technique.

Each layer in the network has an identifier specified by the ID= option in the INPUT, HIDDEN, or TARGET statements. An identifier can be any SAS name, but to avoid confusion, you should not use the name of a variable in the training set. Layer identifiers are used in various statements, such as CONNECT, to specify previously defined layers.

Each unit in the network has a name. For units corresponding to interval variables in input or output layers, the name of the unit is the same as the name of the variable. For units corresponding to dummy variables for categorical (nominal or ordinal) inputs or targets, the name of each unit is constructed by concatenating the name of the input or target variable with the value of the category, truncating as necessary to make the length of the name eight characters or less. For hidden units, the names are constructed by concatenating the layer ID with an integer.

Syntax: NEURAL Procedure

```

PROC NEURAL <option-list>;
  ARCHITECTURE architecture-name
    <HIDDEN=n>
    <DIRECT>;
  CODE FILE=file-name
    <FORMAT=format>
    <RESIDUAL|NORESIDUAL>
    <ERROR|NOERROR>
    <GROUP=name>;
  CONNECT id-list/
    <RANDIST=name>
    <RANLOC=number>
    <RANSCALE=number>;
  CUT id-list | ALL;
  DECISION DECDATA=<libref.>SAS-data-set
    <DECVAR=decision-variable(s)> <option(s)>;

```

```

DELETE id-list | ALL;
FREEZE weight-list /
    <VALUE=number>
    <EST=<libref.>SAS-data-set>;
FREQ variable(s);
HIDDEN integer /
    ID=name
    <ACT=keyword>
    <BIAS | NOBIAS>
    < COMBINE=keyword>;
INITIAL INEST=<libref.>SAS-data-set
    OUTEST=<libref.>SAS-data-set
    <BIADJUST=adjustment-value>
    <INFAN=number>
    <RANDBIAS | NORANDBIAS >
    <RANDOUT | NORANDOUT >
    <RANDOM=integer>
    <RANSCALE | NORANSCALE>;
INPUT variable-list /
    ID=name
    <LEVEL=value>
    < STD=method>;
NETOPTIONS network-option(s);
NLOPTIONS <nonlinear-options>;
PERTURB weight-list /
    OUTEST=<libref.>SAS-data-set
    DF=number
    <RANDIST=name>
    <RANDOM=integer>
    <RANLOC=number>
    <RANSCALE=number>;
PRELIM integer
    INEST=<libref.>SAS-data-set
    OUTEST=<libref.>SAS-data-set
    <ACCELERATE=number>
    <DECELERATE=number>
    <LEARN=number>
    <MAXLEARN=number>
    <MAX | MAXMOMENTUM=number>
    <MINLEARN=number >
    <MOM | MOMENTUM=number>
    <PREITER=integer>
    <PRETECH=name>
    <PRETIME=number>
    <RANDBIAS | NORANDBIAS >
    <RANDOUT | NORANDOUT>
    <RANDOM=integer>;
QUIT;
RANOPTIONSconnection-list /
    <RANDIST=name>
    <RANDOM=integer>
    <RANLOC=number>
    <RANSCALE=number>;

```

```

SAVE OUTEST=<libref.>SAS-data-set
      NETWORK=screen-specification;
SCORE DATA=<libref.>SAS-data-set
      OUT=<libref.>SAS-data-set
      OUTFIT=<libref.>SAS-data-set
      <DUMMIES | NODUMMIES>
      <ROLE=role-option>;
SET weight-list number;
SHOW weights;
TARGET variable-list /
      <ACT=keyword>
      <BIAS | NOBIAS>
      <COMBINE=keyword>
      <ERROR=keyword>
      <ID=name>
      <LEVEL=value>
      <MESTA=number>
      <MESTCON=number>
      <SIGMA=number>
      <STD=method>
      <WEIGHT=number>;
THAW weight-list;
TRAIN OUT=<libref.>SAS-data-set
      OUTEST=<libref.>SAS-data-set
      OUTFIT=<libref.> SAS-data-set
      <ACCEL | ACCELERATE=number>
      <DECEL | DECELERATE=number>
      <DECAY=number>
      <DUMMIES | NODUMMIES>
      <ESTITER=i>
      <LEARN=number>
      <MAX | MAXMOMENTUM=number>
      <MAXITER=integer>
      <MAXLEARN=number>
      <MAXTIME =number>
      <MINLEARN=number>
      <MOM | MOMENTUM=number >
      <TECHNIQUE=name>;
USE <libref.>SAS-data-set;

```

PROC NEURAL Statement

Invokes the NEURAL procedure.

```
PROC NEURAL <option-list>;
```

Required Arguments

DATA=<libref.>SAS-data-set

Specifies the input SAS data set containing the training data.

DMDBCAT=<libref.>SAS-catalog

Specifies the DMDB catalog.

Options

NETWORK=screen-specification

Constructs a network according to a description that was saved by using a SAVE statement during a previous execution of the NEURAL procedure. *screen-specification* is the catalog entry that was specified in the SAVE statement.

Default: None

RANDOM=integer

Specifies the random number seed used in network weight initialization.

Default: 12345

CAUTION:

The weights and predicted outputs from the network cannot be reproduced when a 0 or negative RANDOM= value is specified. When a 0 or negative value is specified, the system clock is used to generate the seed. The actual value of this seed will be unavailable, and you lose control over the initialization of weights. Different initializations will result in different final weights and predicted outputs for repeated runs of the same set of NEURAL statements and same input data sets. Δ

STOPFILE='file pathname'

This option enables you to stop the NEURAL training when you are running a large job. Before you invoke the NEURAL procedure, specify the file pathname in the STOPFILE option. For example, STOPFILE= "c:\mydir\haltneural". Initially, this file should not exist. The NEURAL procedure checks for the existence of this file between iterations in the training process. When you want to stop the job, create the specified file, and the NEURAL procedure will halt the training at the current iteration. The file does not have to contain any contents.

TESTDATA=<libref.>SAS-data-set

Specifies a data set used to compute the test average error during training. At selected iterations (controlled by ESTITER=), each observation in the TESTDATA= data set is read in, scored using the current network weight values, and the error computed. The average test error is then output to the OUTEST= data set.

Note: This requires the TESTDATA= data set to contain the inputs and target variables. Δ

VALIDATA=<libref.>SAS-data-set

Specifies a data set used to compute the validation average error during training. At selected iterations (controlled by ESTITER=), each observation in the VALIDATA= data set is read in, scored using the current network weight values, and the error computed. The average validation error is then output to the OUTEST= data set.

Note: This requires the VALIDATA= data set to contain the inputs and target variables. Δ

REMOTE remote-option(s)

See the DMREG procedure chapter for further explanation of the REMOTE statement.

ARCHITECTURE Statement

Constructs a network with 0 or 1 hidden layers, sets the hidden-unit ACT= and COMBINE= options, and sets default values for various other options as described below.

Interaction: You *cannot* override the hidden-unit ACT= and COMBINE= options implied by the ARCHITECTURE statement, because these are what define the architecture. You *can* override all the other values set by ARCHITECTURE by using an INPUT, a HIDDEN, a TARGET, or a RANOPTIONS statement.

Alias: ARCH

ARCHITECTURE *architecture-name*
 <HIDDEN=*n*>
 <DIRECT>;

Required Arguments

architecture-name

Names the architecture you want to use to construct the network. Only one *architecture-name* from the following list can be specified:

GLIM	Requests a Generalized Linear Model.
MLP	Requests a Multilayer Perceptron.
ORBFEQ	Requests an Ordinary Radial Basis Function Network with Equal Widths.
ORBFUN	Requests an Ordinary Radial Basis Function Network with Unequal Widths.
NRBFEQ	Requests a Normalized Radial Basis Function Network with Equal Widths and Heights.
NRBFEH	Requests a Normalized Radial Basis Function Network with Equal Heights and Unequal Widths.
NRBFEW	Requests a Normalized Radial Basis Function Network with Unequal Heights and Equal Widths.
NRBFEV	Requests a Normalized Radial Basis Function Network with Equal Volume.
NRBFUN	Requests a Normalized Radial Basis Function Network with Unequal Heights and Unequal Widths.

Note: See the following two tables for INPUT, TARGET, and HIDDEN options implied by architecture name and RANOPTIONS implied by architecture name. Δ

Table 11.1 INPUT, TARGET, and HIDDEN Options Implied by Architecture Name

ARCHITECTURE NAME	INPUT Options	TARGET Options	HIDDEN Options
GLIM	STD=NONE	STD=NONE	No hidden layers
MLP			ACT=TANH COMBINE=LINEAR
ORBFEQ			ACT=EXP COMBINE=EQRADIAL
ORBFUN			ACT=EXP COMBINE=EHRADIAL
NRBFEQ		NOBIAS	ACT=SOFTMAX COMBINE=EQRADIAL
NRBFEH		NOBIAS	ACT=SOFTMAX COMBINE=EHRADIAL
NRBFEW		NOBIAS	ACT=SOFTMAX COMBINE=EWRADIAL
NRBFEV		NOBIAS	ACT=SOFTMAX COMBINE=EV RADIAL
NRBFUN		NOBIAS	ACT=SOFTMAX COMBINE=XRADIAL

The following definitions apply to the table below:

fan_in specifies the fan_in of a hidden unit, that is, the number of non-bias and non-altitude weights feeding into the unit.

n_hidden_units is the number of hidden units.

defloc is $2 * \max(.01, (n_hidden_units ** (1/fan_in)))$.

ranloc is the value of the RANLOC= option.

Table 11.2 RANOPTIONS Implied by Architecture Name

ARCHITECTURE NAME	RANOPTIONS for BIAS -> HIDDEN Weights	RANOPTIONS for INPUT -> HIDDEN Weights (not affected by early stopping)	RANOPTIONS for ALTITUDE -> HIDDEN Weights
GLIM			
MLP			
ORBFEQ	RANLOC=defloc RANSSCALE=ranloc*.1	RANSSCALE=1	
ORBFUN	RANLOC=defloc RANSSCALE=ranloc*5	RANSSCALE=1	

ARCHITECTURE NAME	RANOPTIONS for BIAS -> HIDDEN Weights	RANOPTIONS for INPUT -> HIDDEN Weights (not affected by early stopping)	RANOPTIONS for ALTITUDE -> HIDDEN Weights
NRBFEQ	RANLOC=defloc RANSSCALE=ranloc*.1	RANSSCALE=1	
NRBFEH	RANLOC=defloc RANSSCALE=ranloc*.5	RANSSCALE=1	
NRBFEW	RANLOC=defloc RANSSCALE=ranloc*.1	RANSSCALE=1	RANLOC=1 RANSSCALE=ranloc*.5
NRBFEV	RANLOC=.5*defloc RANSSCALE=ranloc*.1	RANSSCALE=1	
NRBFUN	RANLOC=defloc RANSSCALE=ranloc*.5	RANSSCALE=1	RANLOC=1 RANSSCALE=ranloc*.5

Options

HIDDEN= *n*

Specifies the number of hidden units for all architectures other than GLIM.

Default: None

DIRECT

Requests direct connections from inputs to outputs.

CODE Statement

If you want to score a data set based on a previously trained neural network outside of PROC NEURAL, you can use a CODE statement. The CODE statement writes SAS DATA step code to a file or catalog entry. This code can then be included into a DATA step that reads (using a SET statement) the data set to be scored.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

CODE FILE=*file-name*

```

<ERROR | NOERROR>
<CATALOG | CAT | C=>
<FORMAT=format>
<GROUP=name>
<LINESIZE | LS=integer value>
<RESIDUAL | NORESIDUAL>
<DUMMIES | NODUMMIES>
<LOOKUP=lookup algorithm>
<PMML | XML>;

```

See the DMINE procedure chapter for more information on the CODE statement.

Required Arguments

FILE=*file-name*

Specifies where to write the code.

When enclosed in a quoted string, FILE= specifies the path for writing the code to an external file. For example, FILE="c:\mydir\scorecode.sas".

FILE= can also use unquoted SAS filenames of no more than eight characters. If the filename is assigned as a fileref in a FILENAME statement, the file specified in the FILENAME statement is opened. The special filerefs LOG and PRING are always assigned. If the specified name is not an assigned fileref, the specified value is concatenated with a .txt extension before opening. For example, if FOO is not an assigned fileref, FILE=FOO would cause FOO.txt to be opened. If the name has more than eight characters, an error message is printed.

Options

ERROR | NOERROR

Specifies whether the error function E_* is to be computed.

Default: NOERROR

CATALOG | CAT | C=*library.catalog.entry.type*

Where to write the code in the form of library.catalog.entry.type. The compound name can have from one to four levels. The default library is determined by the SAS system option USER=, usually WORK. The default entry is SASCODE, and the default type is SOURCE.

Default: WORK.CATALOGNAME.SASCODE.SOURCE

FORMAT=*format*

Use FORMAT= to format weights or other numeric values that don't have a format from the input data set.

Default: BEST20

GROUP=*group identifier*

Use GROUP= to specify the group identifier for group processing. This should be a valid SAS name of no more than 16 characters, which is used to construct array names and statement labels in the generated code.

LINESIZE | LS=*integer-value*

Use LINESIZE= to specify the line size for generated code. The permissible range is 64 to 254.

Default: 72

RESIDUAL | NORESIDUAL

Use RESIDUAL to generate residual values for the variables R_*, F_*, CL_*, CP_*, BL_*, BP_*, and ROI_*. If you request code for residuals and then score a data set that does not contain target values, the residuals will have missing values.

Default: NORESIDUAL

DUMMIES | NODUMMIES

Use DUMMIES | NODUMMIES to specify whether to keep dummy variables, standardized variables, or other transformed variables in the data set.

Default: NODUMMIES

LOOKUP=SELECT | LINEAR | LINFREQ | BINARY | AUTO

Use LOOKUP= to specify the algorithm that you want to use to look up CLASS levels: SELECT uses a SELECT statement. SELECT is slow under SAS Version 8 if there are many categories, but might be faster in SAS Version 9.

LINEAR uses a linear search with IF statements with categories in the order specified in the DMDDB catalog. This is slow if there are many categories.

LINFREQ uses a linear search with IF statements with categories in descending order of frequency as given in the DMDDB catalog. This is fast if the distribution of class frequencies is highly uneven, but is slow if there are many categories, all with approximately equal frequencies.

BINARY uses a binary search. This is fast, but might produce incorrect results if you generate the code on an ASCII machine and execute the code on an EBCDIC machine or vice versa, and the normalized category values contain characters that collate in different orders on ASCII and EBCDIC. PROC CSCORE is unable to translate this code.

AUTO selects the fastest method for each variable that is portable across ASCII/EBCDIC.

You cannot specify both FILE= and CATALOG=. If you specify neither, the code is written to the SAS log.

Default: AUTO

PMML | XML

Produces 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.

CONNECT Statement

A network can be specified without any CONNECT statements. However, such a network will be connected by default as follows. First all input layers 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 layers. If this particular architecture is not appropriate, use one or more CONNECT statements to explicitly define the network connections.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

CONNECT *id-list*;

Required Arguments

id-list

Lists the identifiers of two or more layers to connect. The identifiers must have been previously defined by the ID= option in an INPUT, a HIDDEN, or a TARGET statement. Each layer except the last is connected to the next layer in the list.

Connections must be feedforward. Loops are not allowed.

For example, the following PROC NEURAL step connects the input layers to the output layer, the input layers to the hidden units, and the hidden units to the output layer.

```
title 'Fully Connected Network';
proc neural data=mydata dmdbcat=mycat;
```

```

input a b / level= nominal id=nom;
input x z / level= interval id=int;
hidden 2 / id= hu;
target y / level=interval id=tar;
connect int tar;
connect nom tar;
connect int hu;
connect nom hu;
connect hu tar;
train;
run;

```

CUT Statement

If the weights corresponding to the connection between two layers are not contributing the predictive ability of the network, you can remove that connection and the corresponding weights by using a CUT statement.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

CUT *id-list* | ALL;

Options

You must specify either:

id-list

Specifies the identifiers of the layers to disconnect.

ALL

Disconnects all layers.

DECISION Statement

Specifies information used for decision processing in the DECIDE, DMREG, NEURAL, and SPLIT procedures. *This documentation applies to all four procedures.*

Tip: The DECISION statement is required for PROC DECIDE. It is optional for DMREG, NEURAL and SPLIT procedures.

DECISION DECDATA=<libref.> SAS-data-set
<DECVARS=decision-variable(s)><option(s)>;

DECDATA= <libref.> SAS-data-set

Specifies the input data set that contains the decision matrix. The DECDATA= data set must contain the target variable.

Note: The DECDATA= data set can also contain decision variables specified by means of the DECVARS= option, and prior probability variable(s) specified by means of the PRIORVAR= option or the OLDPRIORVAR= option, or both.

The target variable is specified by means of the TARGET statement in the DECIDE, NEURAL, and SPLIT procedures or by using the MODEL statement in the DMREG procedure. If the target variable in the DATA= data set is categorical, then the target variable of the DECDATA= data set should contain the category values, and the decision variables will contain the common consequences of making those decisions for the corresponding target level. If the target variable is interval, then each decision variable will contain the value of the consequence for that decision at a point specified in the target variable. The unspecified regions of the decision function are interpolated by a piecewise linear spline. Δ

Tip: The DECDATA= data set can be of TYPE=LOSS, PROFIT, OR REVENUE. If unspecified, TYPE=PROFIT is assumed by default. TYPE= is a data set option that should be specified when the data set is created.

DECVARS=decision-variable(s)

Specifies the decision variables in the DECDATA= data set that contain the target-specific consequences for each decision.

Default: None

COST=cost-option(s)

Specifies numeric constants that give the cost of a decision, or variables in the DATA= data set that contain the case-specific costs, or any combination of constants and variables. There must be the same number of cost constants and variables as there are decision variables in the DECVARS= option. In the COST= option, you cannot use abbreviated variable lists such as D1-D3, ABC-XYZ, or PQR:.

Default: All costs are assumed to be 0.

CAUTION:

The COST= option can be specified only when the DECDATA= data set is of TYPE=REVENUE. Δ

PRIORVAR=variable

Specifies the variable in the DECDATA= data set that contains the prior probabilities to use for making decisions.

Tip: In the DECIDE procedure, if PRIORVAR= is specified, OLDPRIORVAR= must also be specified.

Default: None

OLDPRIORVAR=variable

Specifies the variable in the DECDATA= data set that contains the prior probabilities that were used when originally fitting the model.

Tip: If OLDPRIORVAR= is specified, PRIORVAR= must also be specified.

CAUTION:

OLDPRIORVAR= is not allowed in PROC SPLIT. Δ

Default: None

DELETE Statement

If an input or hidden layer is not contributing the predictive ability of the network, you can remove that layer with a DELETE statement. The DELETE statement also removes all associated weights.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

DELETE *id-list*;

Required Argument

id-list

Specifies the identifiers of layers to delete.

FREEZE Statement

Normally during training, all weights are updated. If you freeze one or more weights, those weights will retain their frozen value until a corresponding THAW statement is executed. Freezing weights causes training to proceed faster and require less memory.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

FREEZE *weight-list* / **<VALUE=number>**
<EST=<libref.>SAS-data-set>;

Required Argument

weight-list

List of weights to freeze.

Weight-list consists of 0 or more repetitions of:

wname \rightarrow *wname-2* where:

wname

is a unit name, a layer identifier, BIAS, or ALTITUDE

wname-2

is a unit name or a layer identifier

Options

You can specify either VALUE= or EST= but not both. If neither option is specified, the weights are frozen to their current values.

VALUE=number

Specifies the numeric value to which weights are to be frozen.

EST=<libref.>SAS-data-set

Specifies the SAS data set containing the values to which weights are to be frozen.

FREQ Statement

Specifies the frequency variable for training.

Category Variable Statement - specifies variables.

FREQ *variable*;

Options

variable

Specifies the frequency variable. (The frequency variable can contain integer and non-integer values.)

Note: The FREQ variable is not required in the DATA= data set. The NEURAL procedure searches for the name of the FREQ variable in the DATA=, VALIDATA=, and TESTDATA= data sets. If the FREQ variable does not appear in any of these data sets, then the procedure issues a warning but continues processing. For any data set that does not contain the FREQ variable, a FREQ value of 1 is used for all observations. △

HIDDEN Statement

You can specify as many HIDDEN statements as you want up to the limits imposed by computer memory, time, and disk space. The hidden layers can be connected in any feedforward pattern using CONNECT statements.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

HIDDEN *integer* / **ID**=*name*
 <**ACT**=*activation-function*>
 <**BIAS**|**NOBIAS**>
 <**COMBINE**=*combination-function*>;

Required Arguments

integer

Specifies the number of units in the hidden layer.

ID=*name*

Specifies the identifier for the layer.

Options**ACT=*activation-function***

Specifies the activation function. See “ACTIVATION FUNCTIONS” on page 269.

Default: For hidden units, the default activation function depends on the combination function and on the number of units in the layer.

For COMBINE=ADD, the default is ACT=IDENTITY.

For COMBINE=LINEAR or EQSLOPES, the default is ACT=TANH.

For 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 bias.

Default: BIAS

COMBINE=*combination-function*

Specifies the combination function. See “COMBINATION FUNCTIONS” on page 269.

INITIAL Statement

After a network has been defined in terms of input, hidden and output layers, all weights and biases in the network must be given initial values before any training is performed. PROC NEURAL will by default supply appropriate random or computed values for these quantities. If you train a network without supplying an INITIAL or USE statement, the network will be initialized using the default specifications.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that 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>;
```

Options

BIADJUST= *adjustment-value*

Specifies how to adjust the random biases for units with a LINEAR combination function. A random bias is adjusted by multiplying by the function of the weights indicated by *adjustment-value*, and dividing by the scale (RANSCALE=) of the distribution from which the random bias was drawn. *adjustment value* can be one of the following:

SUM

Adjusts random initial biases for the sum of the absolute connection weights leading into the unit. This value 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 into the unit. This value is typically used with STD=STD for inputs and RANDIST=NORMAL.

NONE | NO

No bias adjustment.

Default: BIADJUST=NONE

INEST=<libref.>SAS-data-set

Specifies an input data set that contains some or all of the weights. Any weights in the INEST= data set that have missing values are assigned 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. An INEST= data set will typically have been created by using the OUTEST= option in a SAVE or a TRAIN statement from a previous execution of the NEURAL procedure.

INFAN=number

Divide 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.

Default: 0 for radial combination functions, otherwise .5

Range: between 0 and 1

OUTEST=<libref.>SAS-data-set

Specifies the output data set that contains all the initial weights.

RANDBIAS | NORANDBIAS

Specifies whether to randomize output biases.

Note: NORANDBIAS overrides whatever you specify in the RANOPTIONS statement. Δ

Default: NORANDBIAS, which sets bias to the inverse activation function of the target mean.

RANDOM=integer

Specifies the random number seed.

Default: 12345

RANDOUT | NORANDOUT

Specifies whether to randomize the output connection weights.

Note: NORANDOUT overrides whatever you specify in the RANOPTIONS statement. Δ

Default: NORANDOUT, which sets weights to 0.

RANSCALE | NORANSCALE

Specifies whether to randomize target scale estimates.

Default: NORANSCALE, which sets each scale estimate to the standard deviation of the corresponding target variable.

Note: NORANSCALE overrides whatever is specified in the RANOPTIONS statement. Δ

INPUT Statement

The INPUT statement allows you to group together input variables having common levels and standardizations. You can specify as many INPUT statements as you want up to the limits imposed by computer memory, time, and disk space. The input layers can be connected to hidden or output layers using CONNECT statements.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

```
INPUT variable-list / ID=name
      <LEVEL=value>
      <STD=method>;
```

Required Arguments

variable-list

Specifies the input variables.

ID=*name*

Specifies the identifier for the layer.

Options

LEVEL=*value*

Specifies the measurement level, where *value* can be:

NOMINAL | NOM
Nominal

ORDINAL | ORD
Ordinal

INTERVAL | INT
Interval

Default: Interval (for variables specified by a VAR statement in the DMDB procedure) or nominal (for variables specified by a CLASS statement in the DMDB procedure).

STD=*method*

Specifies the standardization method, where *method* is:

NONE | NO

Variables are not altered.

STD

Variables are rescaled to have a mean of 0 and a standard deviation of 1.

RANGE | RAN

Variables are rescaled to have a minimum of 0 and a range of 1. This standardization is not recommended for input variables.

MIDRANGE | MID

Variables are rescaled to have a midrange of 0 and a half-range of 1 (that is, a minimum of -1 and a maximum of 1).

Default: STD

NETOPTIONS Statement

Identifies the network options to set.

Category Option Statement – does not directly affect the network, but sets options for use in subsequent action statements. The options persist until reset at a later stage in the processing.

Alias: NETOPTS | NETOPT

NETOPTIONS *network-option(s)*;

Network Options

DECAY=number

Specifies the weight decay.

Range: *number* ≥ 0

Default: For the QPROP optimization technique: .0001; for all others: 0

The decay option does not affect the Softmax output weights, because the adjustment to the error is very small. The function AdjustForDecay() explicitly excludes target biases from the adjustment term.

INVALIDTARGET=action

Specifies the action taken during training if an out-of-range target value is found, where *action* can be:

OMITCASE OMIT	If INVALIDTARGET = OMITCASE is specified, and an invalid target value is found in the training data set, a warning is given, the observation is not used, but the training will continue.
--------------------	---

STOP	If INVALIDTARGET = STOP is specified, an error is issued, and training is terminated.
------	---

Example: If ERROR = GAMMA is specified in a target statement, the target values should be positive. If a zero or negative value is found for the variable(s) listed in the target statement, the observation containing that value cannot be used for training. The training either continue with the remaining valid observations or stops depending on the INVALIDTARGET= specification. Note, however, if the $\{\backslash$ it mean $\}$, over the training data set, of the target variable(s) is zero or negative, an error is issued and the training is stopped, regardless of the INVALIDTARGET= specification.

Default: The default is INVALIDTARGET = STOP. INVALIDTARGET can be abbreviated as INVALIDTARG or INVTARG.

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.

Default: Depends on the error functions that the network uses. To determine the default, examine the table below named Errors by Objective Functions. Scan the table from left to right. The default is the first column that contains a “yes” in every row corresponding to an error function used in the network. If no such column exists in the table, an error message is issued and the network cannot be trained.

Table 11.3 Errors by Objective Functions

ERRORS	DEV	LIKE	MEST
Normal	Yes	Yes	No
Cauchy	No	Yes	No
Logistic	No	Yes	No
Huber	No	No	Yes
Biweight	No	No	Yes
Wave	No	No	Yes
Gamma	Yes	Yes	No
Poisson	Yes	Yes	No
Bernoulli	Yes	Yes	No
Binomial	Yes	Yes	No
Entropy	Yes	Yes	No
Mbernoulli	Yes	Yes	No
Multinomial	Yes	Yes	No
Mentropy	Yes	Yes	No

RANDF=number

Specifies the degrees of freedom parameter for random numbers. See the Table 11.4 on page 251 table.

RANDIST=*name*

Specifies the type of distribution to be used for random initial weights and perturbations. The distributions and default parameter values are as follows:

Table 11.4 Randomization Options and Default Parameters

RANDIST	RANLOC	RANSSCALE	DF
NORMAL	mean=0	std=1	-
UNIFORM	mean=0	halfrange=1	-
CAUCHY	median=0	scale=1	-
CHIINV	-	scale=1	df=1

Default: NORMAL

RANDOM=*integer*

Specifies the random number seed.

Default: 12345

RANLOC=*number*

Specifies the location parameter for random numbers. See Table 11.4 on page 251

RANSSCALE=*number*

Specifies the scale parameter for random numbers. See Table 11.4 on page 251

NLOPTIONS Statement

Identifies the nonlinear optimization options to set.

Category Option Statement – does not directly affect the network, but sets options for use in subsequent action statements. The options persist until reset at a later stage in the processing.

NLOPTIONS <*nonlinear-options*>;

Nonlinear Options

ABSCONV= *number*

Specifies an absolute function convergence criterion. ABSCONV= is a function of the log-likelihood for the intercept-only model.

Default: The default value is the negative square root of the largest double precision value.

Range: *number* > 0

ABSFCNV= *number*

Specifies an absolute function convergence criterion.

Default: 0

Range: $number > 0$

ABSGCONV= *number*

Specifies the absolute gradient convergence criterion.

Default: 1E-5

Range: $number > 0$

ABSXCONV= *number*

Specifies the absolute parameter convergence criterion.

Default: 0

Range: $number > 0$

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.

Default: 2

Range: $number > 0$

DIAHES

Forces the optimization algorithm (TRUREG, NEWRAP, or NRRIDG) to take advantage of the diagonality.

FCONV= *number*

Specifies a function convergence criterion.

Default: 1E-4

Range: $number > 0$

FSIZE= *number*

Specifies the parameter of the relative function and relative gradient termination criteria.

Default: Not applicable.

Range: $number \geq 0$

GCONV= *number*

Specifies the relative gradient convergence criterion.

Default: 1E-8

Range: $number > 0$

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.

Default:

1 - for LEVMAR minimization technique

0 - for all others

INHESSIAN= *number*

Specifies how to define the initial estimate of the approximate Hessian for the quasi-Newton techniques QUANEW and DBLDOG.

Default: The default is to use a Hessian based on the initial weights as the initial estimate of the approximate Hessian. When $r=0$, the initial estimate of the approximate Hessian is computed from the magnitude of the initial gradient.

Range: $number > 0$

INSTEP= *number*

Specifies the initial radius of the trust region used in the TRUREG, DBLDOG, and LEVMAR algorithms.

Default: 1

Range: $number > 0$

LCEPS | LCEPSILON= *number*

Specifies the range for active constraints.

Range: $number > 0$

LCSINGULAR= *number*

Specifies the tolerance for dependent constraints

Range: $number > 0$

LINESEARCH= *number*

Specifies the line-search method for the CONGRA, QUANEW, and NEWRAP optimization techniques.

Default: 2

Range: $1 \leq number \leq 8$

LSPRECISION= *number*

Specifies the degree of accuracy that should be obtained by the second and third line-search algorithms.

Default:

Table 11.5 Line-Search Precision Values

TECHNIQUE=	UPDATE=	LSPRECISION VALUE
QUANEW	DBFGS, BFGS	0.4
QUANEW	DDFP, DFP	0.06
CONGRA	all	0.1
NEWRAP	no update	0.9

Range: $number > 0$

MAXFUNC= *number*

Specifies the maximum number of function calls in the optimization process.

Default: 2147483647 for all techniques

Range: $number > 0$

MAXITER= *number*

Specifies the maximum number of iterations in the optimization process.

Default:

100 for TRUREG, NRRIDG, NEWRAP, and LEVMAR

200 for QUANEW and DBLDOG

400 for CONGRA

Range: $number > 0$

MAXSTEP= *number*

Specifies the upper bound for the step length of the line-search algorithms.

Default: The largest double precision value

Range: *number* > 0

MAXTIME= *number*

Specifies the upper limit of CPU time for the optimization process. It is measured in seconds.

Default: 7 days, that is, MAXTIME=604800 seconds

Range: *number* > 0

MINITER= *number*

Specifies the minimum number of iterations in the optimization process.

Default: 0

Range: *number* \geq 0

NOPRINT

Suppresses all output. Only ERRORS, WARNINGS, and NOTES are printed on the log file.

PALL

Prints all optional output except the output generated by the PSTDERR , LIST, or LISTCODE option.

PSUMMARY

Restricts the amount of default printed output to a short form of iteration history and NOTES, WARNINGS, and ERRORS.

PHISTORY

Prints the optimization history. If PSUMMARY or NOPRINT are 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/ascent search direction after the *number* of iterations has been completed.

Range: *number* \geq 1

Default:

For TECHNIQUE=CONGRA, and UPDATE= PB, restart is done automatically, so *number* is not used;

For TECHNIQUE=CONGRA, and UPDATE not = PB, *number* is the number of parameters.

For TECHNIQUE=QUANEW, *number* is the largest integer.

SINGULAR= *number*

Specifies an absolute singularity criterion for the computation of the inertia of Hessian and cross-product Jacobian and their projected forms.

Default: 1E-8

Range: *number* > 0

TECHNIQUE= *method*

Specifies an optimization technique, where *method* is one of the following:

CONGRA

Specifies the Conjugate Gradient optimization technique. This is the default when the number of parameters to be estimated is \geq 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 when the number of parameters to be estimated is ≤ 40 .

QUANEW

Specifies the quasi-Newton optimization technique. This is the default when the number of parameters to be estimated is in the range $40 < n \leq 400$.

TRUREG

Specifies the Trust-Region optimization technique.

UPDATE=update-type

Specifies an update method, where *update-type* 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 (Broyden-Fletcher-Goldfarb-Shanno) update of the Cholesky factor of the Hessian matrix. This is the default for **TECHNIQUE=QUANEW** and **DBLDOG**.

DDFP

For **TECHNIQUE= DBLDOG** or **QUANEW**, performs the dual DFP (Davidson-Fletcher-Powell) update of the Cholesky factor of the Hessian matrix.

DFP

For **TECHNIQUE= QUANEW**, performs the original DFP (Davidson-Fletcher-Powell) update of the inverse Hessian matrix.

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**, performs the Polak-Ribiere 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.

Default: 2

XCONV= number

Specifies the relative parameter convergence criterion.

Default: 0

Range: *number* > 0

XSIZE= number

Specifies the number of successive iterations for which the criterion must be satisfied before the optimization process can be terminated.

Default: 0

Range: *number* > 0

PERTURB Statement

Perturbs weights. Perturbing weights can sometimes allow you to escape a local minimum.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

```
PERTURB weight-list / <RANDF=number >
      <OUTEST=<libref>SAS-data-set>
      <RANDIST=name>
      <RANDOM=integer>
      <RANLOC=number>
      <RANSCALE=number>;
```

Required Argument

weight-list

List of weights to freeze.

Weight-list consists of 0 or more repetitions of:

wname \rightarrow *wname-2* where:

wname

is the unit name, the layer ID, BIAS, or ALTITUDE

wname-2

is the unit name or layer ID

Options

RANDF=*number*

Specifies the degrees of freedom parameter for random numbers. See Table 11.4 on page 251 for values.

OUTEST=<*libref*>*SAS-data-set*

Specifies the output data set containing all the weights.

Default: none

RANDIST=*name*

Specifies the type of distribution for random numbers. See Table 11.4 on page 251 for values.

RANDOM=*integer*

Specifies the random number seed.

Default: 0

RANLOC=*number*

Specifies the location parameter for random numbers. See Table 11.4 on page 251 for values.

RANSCALE=number

Specifies the scale parameter for random numbers. See Table 11.4 on page 251 for values.

PRELIM Statement

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 (default 10) using different initializations. The final weights of the best trained network are then used to initialize a subsequent TRAIN statement.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

PRELIM *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>;

```

Required Argument*integer*

Specifies the number of preliminary optimizations.

Options**ACCEL | ACCELERATE=number**

Specifies the rate of increase of learning for the RPROP optimization technique.

Range: *number* > 1

Default: 1.2

DECEL | DECELERATE=number

Specifies the rate of decrease of learning for the RPROP optimization technique.

Range: 0 < *number* < 1

Default: 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 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.

Range: *number* > 0

Default: 0.1

MAXLEARN=number

Specifies the maximum learning rate for RPROP.

Range: *number* > 0

Default: Reciprocal of the square root of the machine epsilon

MAXMOM | MAXMOMENTUM=number

Specifies the maximum momentum for BPROP.

Range: *number* > 0

Default: 1.75

MINLEARN=number

Specifies the minimum learning rate for RPROP.

Range: *number* > 0

Default: Square root of the machine epsilon

MOM | MOMENTUM=number

Specifies the momentum for BPROP.

Range: $0 \leq \textit{number} < 1$

Default: For BPROP: 0.9; for RPROP: 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.

Default: 10

PRETECH | TECHNIQUE=name

Specifies the optimization technique. See “TRAIN Statement” on page 266.

Default: Same as TECH= in the TRAIN statement.

PRETIME=number

Specifies the amount of time after which training stops.

RANDBIAS | NORANDBIAS

Specifies whether to randomize output biases.

Default: NORANDBIAS, which sets bias to the inverse activation function of the target mean.

[NORANDBIAS overrides whatever you specify in the RANDOM statement.]

RANDOM=integer

Specifies the random number seed.

Default: 0

RANDOUT | NORANDOUT

Specifies whether to randomize the output connection weights.

Default: NORANDOUT, which sets weights to 0.

[NORANDOUT overrides whatever you specify in the RANDOM statement.]

QUIT Statement

Stops the procedure.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

QUIT;

RANOPTIONS Statement

Specifies distribution of random initial weights.

Category Option Statement – does not directly affect the network, but sets options for use in subsequent action statements. The options persist until reset at a later stage in the processing.

RANOPTIONS *connection-list* /

<RANDF=*number*>
 <RANDIST=*name*>
 <RANDOM=*integer*>
 <RANLOC=*number*>
 <RANSCALE=*number*>;

Required Arguments

Note: When a RANOPTIONS statement is executed, the specified options are stored in all the connections listed before the slash. These options are used whenever an INITIAL or PRELIM statement is executed. If you submit two RANOPTIONS statements for the same connection, the second statement overrides all options in the first. In other words, one RANOPTIONS statement does not remember what options were specified in previous RANOPTIONS statements. To have options persist over multiple statements, use the NETOPTIONS statement. Δ

connection-list

List of connections to randomize.

connection-list consists of 0 or more repetitions of:

wname \rightarrow *wname-2* where:

wname
is the layer ID, BIAS, or ALTITUDE

wname-2
is the layer ID

Options

RANDF=number

Specifies the degrees of freedom parameter for random numbers. See Table 11.4 on page 251 for values.

Default: 1

RANDIST=name

Specifies the type of distribution for random numbers. See Table 11.4 on page 251 for values.

Default: NORMAL

RANDOM=integer

Specifies the random number seed.

Default: 0

RANLOC=number

Specifies the location parameter for random numbers. See Table 11.4 on page 251 for values.

RANSCALE=number

Specifies the scale parameter for random numbers. See Table 11.4 on page 251 for values.

SAVE Statement

Writes weights to data set or a description of the network to a catalog entry.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

NOTE: At least one option must be specified, but there is no single argument that is required.

```
SAVE OUTEST=<libref.>SAS-data-set
    NETWORK=screen-specification;
```

Options

Specify at least one:

NETWORK=screen-specification

Saves the definition of the entire network. *screen-specification* is the name of a catalog entry.

OUTEST=<libref.> SAS-data-set

Saves the network weights in an output data set.

SCORE Statement

Creates an output data set containing predicted values and possibly other results such as residuals, classifications, decisions, and assessment values.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

```
SCORE DATA=<libref.>SAS-data-set
      OUT=<libref.>SAS-data-set
      OUTFIT=<libref.>SAS-data-set
      <DUMMIES | NODUMMIES>
      <ROLE=role-option>;
```

Required Arguments

OUT=<libref.>SAS-data-set

Specifies the output data set that contains the outputs.

Options

DATA=<libref.>SAS-data-set

Specifies the input data to be scored that contains inputs. The data can contain targets as well but this is optional.

Default: Defaults to the training data (DATA= in the PROC statement).

DUMMIES | NODUMMIES

Specifies whether to write dummy variables to the OUT= data set.

Default: NODUMMIES

OUTFIT=<libref.>SAS-data-set

Specifies the output data set that contains the fit statistics.

ROLE=role-option

Specifies the role of the DATA= data set. *ROLE=role-option* primarily affects which fit statistics are computed and what their names and labels are. *Role-option* is one of the following:

TRAIN

Specifies that the DATA= data set is the training set. The data set must contain the target variable.

VALID | VALIDATION

Specifies that the DATA= data set is a validation set. The data set must contain the target variable.

TEST

Specifies that the DATA= data set is a test set. The data set must contain the target variable.

SCORE

Specifies that residuals, error functions, and fit statistics are not produced. The data set does not have to contain the target variable.

Default: TEST, except as follows:

TRAIN	when the DATA= data set in the PROC statement is the same as the DATA= data set in the SCORE statement. Specifying TRAIN with any data set other than the actual training set is an error.
VALID	when the DATA= data set in the SCORE statement is the same as the VALIDATA= data set in the PROC statement.

SET Statement

Sets the value of the weight-list to number. The SET statement does not freeze weights, so subsequent training might change the values of the weights specified in a SET statement.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

SET *weight-list number*;

Required Arguments

weight-list

Specifies the list of weights to be affected or changed.

number

Specifies the number to which the weight-list is set.

SHOW Statement

Prints information about the network.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

NOTE: At least one option must be specified, but there is no single argument that is required.

**SHOWWEIGHTS
STATEMENTS**

Options

Specify at least one:

STATEMENTS

Prints statements that can be used with the NEURAL procedure to reproduce the network.

WEIGHTS

Prints the network weights.

TARGET Statement

Defines an output layer.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

Alias: OUTPUT

```
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>;
```

Required Arguments***variable-list***

Specifies the target variables.

ID=*name*

Specifies the identifier for the layer.

Options**ACT=*activation-function***

Specifies the activation function. See Table 11.8 on page 269.

Default: Depends on the measurement level, as follows:

If LEVEL=INTERVAL, then the default is IDENTITY.

If LEVEL=ORDINAL then the default is LOGISTIC.

If LEVEL=NOMINAL, then the default is MLOGISTIC

(For Error=MBERNOULLI, MENTROPY, or MULTINOMIAL, the only activation function allowed is MLOGISTIC.)

BIAS | NOBIAS

Specifies whether to use bias (or not to use bias).

Default: BIAS

COMBINE=*combination-function*

Specifies the combination function. See Table 11.9 on page 270.

ERROR=keyword

Specifies the Error function. Default is NORMAL for LEVEL=INTERVAL; otherwise, default is MBERNOULLI. For more information, see the Error Functions table that follows.

Table 11.6 Error Functions

KEYWORD	TARGET	DESCRIPTION
Functions with scale parameters :		
NORmal	any	Normal distribution
CAUchy	any	Cauchy distribution
LOGistic	any	Logistic distribution
HUBer	any	Huber M estimator
BIWeight	any	Biweight M estimator
WAVE	any	Wave M estimator
GAMma	>0	Gamma distribution
POIsson	≥ 0	Poisson distribution
Functions with no scale parameter		
BERnoulli	0,1	Bernoulli distribution (binomial with one trial)
BINomial	≥ 0	Binomial distribution
ENTropy	0-1	Cross or relative entropy for independent targets
MBErnoulli	0,1	Multiple Bernoulli (multinomial with one trial)
MULTinomial	≥ 0	Multinomial distribution
MENtropy	0-1	Cross or relative entropy for targets that sum to 1 (Kullback-Leibler divergence)

LEVEL=value

Specifies the measurement level, where *value* can be:

NOMINAL | NOM

Nominal.

ORDINAL | ORD

Ordinal.

INTERVAL | INT

Interval.

Default:

NOMINAL for character variables.

INTERVAL for numeric variables.

MESTA=number

Specifies the scale constant for M estimation.

Default: Default value is computed from MESTCON to give consistent scale estimates for normal noise.

MESTCON=number

Specifies the tuning constant for M estimation.

Default:

Huber: 1.5;

Biweight: 9;

Wave: $2.1 * \pi$

SIGMA=number

Specifies the fixed value of the scale parameter.

Default: By default, SIGMA is not used; but with OBJECT=LIKE, the scale parameter is estimated.

STD=method

Specifies the standardization method, where *method* is

NONE | NO

Variables are not altered.

STD

Variables are rescaled to have a mean of 0 and a standard deviation of 1.

RANGE | RAN

Variables are rescaled to have a minimum of 0 and a range of 1.

MIDRANGE | MID

Variables are rescaled to have a midrange of 0 and a half-range of 1 (that is, a minimum of -1 and a maximum of 1).

Default: NO

THAW Statement

Thaws frozen weights.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that statement.

THAW *weight-list*;

Required Argument

weight-list

List of weights to thaw.

Weight-list consists of 0 or more repetitions of:

wname \rightarrow *wname-2* where:

wname

is the unit name, the layer ID, BIAS, or ALTITUDE

wname-2

is the unit name or the layer ID

Default: All weights are thawed.

TRAIN Statement

Trains the network.

Category Action Statement – affects the network or the data sets. Options set in an action statement affect only that 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>;

```

Options

ACCEL | **ACCELERATE**=*number*

Specifies the rate of increase of learning for the RPROP optimization technique.

Range: *number* > 1

Default: 1.2

DECEL | **DECELERATE**=*number*

Specifies the rate of decrease of learning for the RPROP optimization technique.

Range: 0 < *number* < 1

Default: 0.5

DUMMIES | **NODUMMIES**

Specifies whether to write dummy variables to the OUT= data set.

Default: NODUMMIES

ESTITER=*i*

i = 0

Writes only initial and final weights to the OUTEST=data set.

i > 0

Writes weights after every *i* iterations, as well as the final weights, to the OUTEST= data set.

Default: 0

LEARN=*number*

Specifies the learning rate for BPROP or the initial learning rate for QPROP and RPROP.

Range: *number* > 0

Default: 0.1

MAXITER=*integer*

Maximum number of iterations.

Default:

100 for TRUREG and LEVMAR

200 for QUANEW and DBLDOG

400 for CONGRA

1000 for BPROP, RPROP, and QPROP

MAXLEARN=*number*

Specifies the maximum learning rate for RPROP.

Range: *number* > 0

Default: 1/MACSQRTEPS

MAXMOM | MAXMOMENTUM=*number*

Specifies the maximum momentum for BPROP.

Range: *number* > 0

Default: 1.75

MAXTIME=*number*

Specifies the amount of time after which training stops.

Default: 7 days, that is, 604800 seconds

MINLEARN=*number*

Specifies the minimum learning rate for RPROP.

Range: *number* > 0

Default: MACSQRTEPS

MOM | MOMENTUM=*number*

Specifies the momentum for BPROP.

Range: $0 \leq \textit{number} < 1$

Default:

For BPROP: 0.9

For RPROP: 0.1

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 where *name* is one of the following:

TRUREG

Requests Trust region.

LEVMAR

Requests Levenberg-Marquardt.

DBLDOG

Requests Double dogleg.

QUANEW

Requests quasi-Newton.

CONGRA

Requests Conjugate gradient.

BPROP

Requests standard backprop (backpropagation), that is, a variation on an algorithm called the generalized delta rule. In backpropagation, the difference (delta) between the output value and the target value is the error.

RPROP

Requests the RPROP algorithm.

QPROP

Requests Quickprop.

See the following table for the defaults for weight-based optimization techniques for a given value of the OBJECT= option.

Table 11.7 Defaults for Weight-based Optimization Techniques

OBJECTIVE FUNCTION	OPTIMIZATION TECHNIQUE	NUMBER OF WEIGHTS
OBJECT=DEV	LEVMAR	0 to 100 weights
OBJECT=DEV	QUANEW	101 - 501 weights
OBJECT=DEV	CONGRA	501 or more weights
(All other objective functions)	QUANEW	up to 500 weights
(All other objective functions)	CONGRA	501 or more weights

USE Statement

Sets all weights to values from a data set.

Category Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

USE <libref.>SAS-data-set;

Required Arguments

<libref.>SAS-data-set

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.

ACTIVATION FUNCTIONS

Table 11.8 Activation Functions

FUNCTION	RANGE	FUNCTION (OF NET INPUT t)
IDEntity	$(-\infty, +\infty)$	t
LINear	$(-\infty, +\infty)$	t
EXPonential	$(0, \infty)$	exp^t
RECiprocal	$(0, \infty)$	$\frac{1}{t}$
SQUare	$[0, +\infty)$	t^2
LOGistic	$(0, 1)$	$\frac{1}{1+e^{-t}}$
MLOGistic	$(0, 1)$	$\frac{e^t}{\sum exponentials}$
SOFTmax	$(0, 1)$	$\frac{e^t}{\sum exponentials}$
GAUss	$(0, 1]$	e^{-t^2}
SINe	$[0, 1]$	$\sin(t)$
COSine	$[0, 1]$	$\cos(t)$
ELLIott	$(-1, 1)$	$\frac{t}{(1+abs(t))}$
TANh	$(-1, 1)$	$\tanh(t) = 1 - \frac{2}{(1+e^{(2t)})}$
ARCTan	$(-1, 1)$	$\arctan(t) * \frac{2}{\pi}$

COMBINATION FUNCTIONS

A combination function combines the values received from preceding nodes into a single number called the net input. Both output and hidden layers are assigned combination functions.

The following combination functions are available. Formulas are given in Table 11.9 on page 270.

Add

Adds all the incoming values without using any weights or biases.

Linear

Is a linear combination of the incoming values and weights.

EQSlopes

Is identical to the Linear combination function, except that the same connection weights are used for each unit in the layer, although different units have different biases. EQSlopes is mainly used for ordinal targets.

EQRadial

Is a radial basis function with equal heights and widths for all units in the layer.

EHRadial

Is a radial basis function with equal heights but unequal widths for all units in the layer.

EWRadial

Is a radial basis function with equal widths but unequal heights for all units in the layer.

EVRadial

Is a radial basis function with equal volumes for all units in the layer.

XRadial

Is a radial basis function with unequal heights and widths for all units in the layer. The following definitions apply to the Table of Combination Functions:

All summations

Are divided by the net inputs indexed by i .

alt_j

The altitude of the j th unit

$bias_j$

The width (bias) of the j th unit

$bias$

A common bias shared by all units in the layer

f

The fan-in of the j th unit

w_{ij}

The weight connecting the i th incoming value to the j th unit

w_i

The common weight for the i th input shared by all units in the layer

x_i

The i th incoming value

Table 11.9 Combination Functions

FUNCTION	DEFINITION
ADD	$\sum x_i$
LINear	$bias_j + \sum w_{ij}x_i$
EQSlopes	$bias_j + \sum_i w_i x_i$
XRAdial	$f \log (alt_j) - bias_j^2 \sum_i [(w_{ij} - x_i)^2]$
EHRAdial	$bias_j^2 [(w_{ij} - x_i)^2]$
EVRAdial	$f \log (bias_j) - bias_j^2 [(w_{ij} - x_i)^2]$

FUNCTION	DEFINITION
EWRadial	$f \log (altb_j) - bias^2 [(w_{ij} - x_i)^2]$
EQRadial	$-bias^2 [(w_{ij} - x_i)^2]$
RADial	defaults to EHRadial

Details: NEURAL Procedure

For details about neural network architecture and training, see the online Neural Network Node: Reference documentation. For an introduction to predictive modeling, see the online Predictive Modeling document. Both of these documents can be accessed by using the Help menu to select the “Open the Enterprise Miner Nodes Help” item.

The BPROP, RPROP, and QPROP Algorithms Used in PROC NEURAL

Introduction

While the standard backprop algorithm has been a very popular algorithm for training feedforward networks, performance problems have motivated numerous attempts at finding faster algorithms.

The following discussion of the implementation of the backprop (BPROP), RPROP, and QPROP algorithms in PROC NEURAL relates the details of these algorithms with the printed output resulting from the use of the PDETAIL option. The discussion uses the algorithmic description and notation in Schiffmann, Joost, and Werner (1994) as well as the Neural Net FAQ available at: <ftp://ftp.sas.com/pub/neural/FAQ.html>.

There is an important distinction between “backprop” (or “back propagation of errors”) and the “backpropagation algorithm”.

The “back propagation of errors” is an efficient computational technique for computing the derivative of the error function with respect to the weights and biases of the network. This derivative, more commonly known as the error gradient, is needed for any first order nonlinear optimization method. The standard backpropagation algorithm is a method for updating with weights based on the gradient. It is a variation of the simple “delta rule”. See “What is backprop?” in part 2 of the FAQ for more details and references on standard backprop, RPROP, and Quickprop.

With any of the “prop” algorithms, PROC NEURAL allows detailed printing of the iterations. The PDETAIL option in the TRAIN statement prints, for each iteration, all quantities involved in the algorithm for each weight in the network. This option should be used with caution as it can result in voluminous output. However, by restricting the number of iterations and number of non-frozen weights, a manageable amount of information is produced. The purpose of the PDETAIL option is to allow you to follow the nonlinear optimization of the error function for each of the network weights. For any particular propagation method, as described below, all quantities used to compute an updated weight are printed.

In standard backprop, too low a learning rate makes the network learn very slowly. Too high a learning rate makes the weights and error function diverge, so there is no learning at all. If the error function is quadratic, as in linear models, good learning rates can be computed from the Hessian matrix. If the error function has many local and global optima, as in typical feedforward neural networks with hidden units, the optimal learning rate often changes dramatically during the training process, because

the Hessian also changes dramatically. Trying to train a neural network using a constant learning rate is usually a tedious process requiring much trial and error.

With batch training, there is no need to use a constant learning rate. In fact, there is no reason to use standard backprop at all, because vastly more efficient, reliable, and convenient batch training algorithms exist such as Quickprop and RPROP.

Many other variants of backprop have been invented. Most suffer from the same theoretical flaw as standard backprop: the magnitude of the change in the weights (the step size) should not be a function of the magnitude of the gradient. In some regions of the weight space, the gradient is small and you need a large step size; this happens when you initialize a network with small random weights. In other regions of the weight space, the gradient is small and you need a small step size; this happens when you are close to a local minimum. Likewise, a large gradient might call for either a small step or a large step. Many algorithms try to adapt the learning rate, but any algorithm that multiplies the learning rate by the gradient to compute the change in the weights is likely to produce erratic behavior when the gradient changes abruptly. The great advantage of Quickprop and RPROP is that they do not have this excessive dependence on the magnitude of the gradient. Conventional optimization algorithms use not only the gradient but also second-order derivatives or a line search (or some combination thereof) to obtain a good step size.

Mathematical Notation

It is helpful to establish notation so that we can relate quantities and describe algorithms.

- 1 $\omega_{ij}(n)$ is the weight associated with the connection between the i th unit in the current layer and the j th unit from the previous layer. The argument n refers to iteration.
- 2 $\Delta\omega_{ij}(n)$ is the update or change for weight $\omega_{ij}(n)$. This update results in the $n + 1$ iteration value for ω_{ij} .
- 3 $\frac{\partial E(n)}{\partial \omega_{ij}}$ is the partial derivative of the error function $E(\omega)$ with respect to the weight ω_{ij} at the n th iteration.
- 4 $y_k^m(\mathbf{x}^m; \omega)$ is the k th component of the output vector for the m th case as a function of the inputs \mathbf{x}^m and network weights w .
- 5 $t_k^m(\mathbf{x}^m)$ is the k th component of the target vector for the m th case as a function of the inputs \mathbf{x}^m .

The basic algorithm in all methods is a generic update given by

$$\omega_{ij}(n+1) = \omega_{ij}(n) + \Delta\omega_{ij}(n)$$

The BPROP, RPROP, and QPROP algorithms differ in how $\Delta\omega_{ij}$ is computed.

Standard Printing for the PROP Algorithms

When the PDETAIL option is not specified, a standard table is produced displaying the iteration number, the value of the objective function at that iteration, and l_∞ norm of the gradient vector $\nabla E(\omega)$.

This table is useful for overall convergence behavior. However, unlike the table produced by the PDETAIL option, no information about individual network weights is given.

In the case of sum of squared error, which results from specifying OBJECTIVE=DEV in the NETOPTS statement and ERROR=NORMAL in the TARGET statement, the error function serves as the objective function, and is given by

$$E(\omega) = \frac{1}{2} \sum_{m=1}^M \sum_{k=1}^K (y_k^m(\mathbf{x}; \omega) - t_k^m)^2$$

Candidate network weight values ω^* that minimize the objective function $E(\omega)$ satisfy the first order condition

$$\nabla E(\omega^*) = 0$$

Hence, a natural convergence criteria is

$$\|\nabla E(\omega)\|_{\infty} < \epsilon$$

for some small value ϵ . This is, in fact, convergence criteria for all prop methods. The value of ϵ is set by the ABSGCONV= option in the NLOPTIONS statement, with a default value of $\epsilon = 10^{-8}$. Note that the l_{∞} norm $\|z\|_{\infty}$ for some vector z is simply the maximum of the absolute value of the components of z .

The standard table prints the following quantities:

- \square Iteration n
- \square Objective, $E(\omega)$, using current network weight ω
- \square Max Abs Gradient Element, $\|\nabla E(\omega)\|_{\infty}$

When the PDETAIL option is specified, this standard table is still printed. Each line of the standard tables follows the detail lines for each of the weights at each iteration.

The Standard Backprop Algorithm

The standard backprop algorithm is a gradient descent with momentum. At the n th iteration, the update is computed as

$$\Delta\omega_{ij}(n) = -\epsilon \frac{\partial E(n)}{\partial \omega_{ij}} + \alpha \Delta\omega_{ij}(n-1)$$

For TECH=BPROP, the PDETAIL option in the TRAIN statement results in the following quantities being printed:

$\Delta\omega_{ij}(n-1)$ is labeled "Previous Change"

$\frac{\partial E(n)}{\partial \omega_{ij}}$ is labeled "Gradient"

$\Delta\omega_{ij}(n)$ is labeled "Current Change"

$\omega_{ij}(n)$ is labeled "Previous Weight"

$\omega_{ij}(n+1)$ is labeled "Current Weight"

The learning rate ϵ and the momentum α are printed at the beginning of the table. These quantities are set by the LEARN= and MOMENTUM= options respectively, with the default values of $\epsilon = 0.1$ and $\alpha = 0.9$.

The RPROP Algorithm

The RPROP algorithm (Riedmiller and Braun 1993) is unusual as a descent algorithm in that it does not use the magnitude of the gradient in calculating the weight update. Instead, the signs of the current and previous gradient are used to determine a step size $\Delta_{ij}(n)$ at each iteration.

To prevent oscillations and underflows, the step size $\Delta_{ij}(n)$ is bounded by

$$\Delta_{\min} \leq \Delta_{ij}(n) \leq \Delta_{\max}$$

The value of Δ_{\max} is set by the MAXLEARN= option with a default value of 10^7 . The value of Δ_{\min} is set by the MINLEARN= option with a default value of 10^{-7} . Note that these values are substantially different from the recommendations given in Schiffmann, Joost and Werner, (1994). These new values improved stability and convergence over a wide range of problems.

For each connection weight, an initial stepsize $\Delta_{ij}(0)$ is given a small value. According to Schiffmann, Joost and Werner, (1994), results are not typically dependent on the exact value given $\Delta_{ij}(0)$. PROC NEURAL uses a default initial step size of 0.1 for all weights and is set by the LEARN= option in the TRAIN statement.

At the n th iteration, adjust the step size by

$$\Delta_{ij}(n) = \begin{cases} \Delta_{ij}(n-1)u, & \text{if } \frac{\partial E(n)}{\partial \omega_{ij}} \frac{\partial E(n-1)}{\partial \omega_{ij}} > 0 \\ \Delta_{ij}(n-1)d, & \text{if } \frac{\partial E(n)}{\partial \omega_{ij}} \frac{\partial E(n-1)}{\partial \omega_{ij}} < 0 \\ \Delta_{\max}, & \text{if } \Delta_{ij}(n) > \Delta_{\max} \\ \Delta_{\min}, & \text{if } \Delta_{ij}(n) < \Delta_{\min} \end{cases}$$

The factors u and d are the acceleration and deceleration respectively. The values of these factors are set by the ACCELERATE= and DECELERATE= options in the TRAIN statement. The default value for ACCELERATE= 1.2; for DECELERATE= the default value is 0.5.

For TECH=RPROP, the PDETAIL option in the TRAIN statement results in the following quantities being printed:

$\Delta_{ij}(n-1)$ is labeled "Previous Step Size"

$\frac{\partial E(n-1)}{\partial \omega_{ij}}$ is labeled "Previous Gradient"

$\frac{\partial E(n)}{\partial \omega_{ij}}$ is labeled "Current Gradient"

$\Delta_{ij}(n)$ is labeled "Current Step Size"

$\Delta\omega_{ij}(n)$ is labeled "Current Change"

$\omega_{ij}(n)$ is labeled "Previous Weight"

$\omega_{ij}(n+1)$ is labeled "Current Weight"

The Quickprop Algorithm

The Quickprop algorithm (Fahlman 1989) assumes that the error function behaves locally like a parabola, and uses the method of false position to find the minimum of the approximating quadratic. Variations are required to ensure that the change is not uphill and to handle cases where the gradient does not change between iterations (causing the false position method to fail).

The quickprop algorithm uses a modified gradient $\frac{\partial E^*(n)}{\partial \omega_{ij}}$ related to the regular gradient by

$$\frac{\partial E^*(n)}{\partial \omega_{ij}} = \frac{\partial E(n)}{\partial \omega_{ij}} + \text{decay} * \omega_{ij}(n)$$

At the n th iteration, the weight update is given by

$$\Delta \omega_{ij}(n) = -\epsilon(n) \frac{\partial E^*(n)}{\partial \omega_{ij}} + \alpha_{ij}(n) \Delta \omega_{ij}(n-1)$$

For initialization at $n=1$, we set $\Delta \omega_{ij}(0) = 0$, so the update step becomes a gradient descent:

$$\Delta \omega_{ij}(1) = -\epsilon(1) \frac{\partial E^*(1)}{\partial \omega_{ij}}$$

At the second and subsequent iterations, $\epsilon(n)$ and $\alpha_{ij}(n)$ are computed as follows:

$$\epsilon(n) = \begin{cases} \epsilon_0, & \text{if } \frac{\partial E^*(n)}{\partial \omega_{ij}} \Delta \omega_{ij}(n-1) > 0 \\ \epsilon_0, & \text{if } \Delta \omega_{ij}(n-1) = 0 \\ 0, & \text{otherwise} \end{cases}$$

Calculation of $\alpha_{ij}(n)$ first involves evaluation $\hat{\alpha}_{ij}(n)$, the numerical estimate of the second derivative:

$$\hat{\alpha}_{ij}(n) = \frac{\frac{\partial E^*(n)}{\partial \omega_{ij}}}{\frac{\partial E^*(n-1)}{\partial \omega_{ij}} - \frac{\partial E^*(n)}{\partial \omega_{ij}}}$$

This second derivative can become large in absolute value or can signal a move “up” the gradient away from a minimum. The following modifications are applied to account for these situations.

$$\alpha(n) = \begin{cases} \alpha_{\max}, & \text{if } |\hat{\alpha}_{ij}(n)| > \alpha_{\max} \\ \alpha_{\max}, & \text{if } \left(\frac{\partial E^*(n-1)}{\partial \omega_{ij}} - \frac{\partial E^*(n)}{\partial \omega_{ij}} \right) \Delta \omega_{ij}(n-1) > 0 \\ \hat{\alpha}_{ij}(n), & \text{otherwise} \end{cases}$$

The value of ϵ_o is set by the LEARN= option in the TRAIN statement, with a default value of $\epsilon_o = 0.1$. The bound α_{\max} is set by the MAXMOMENTUM= option in the TRAIN statement, with a default value of $\alpha_{\max} = 1.75$.

For TECH=QPROP, the PDETAIL option in the TRAIN statement results in the following quantities being printed:

$\omega_{ij}(n-1)$ is labeled "Previous Weight"

$\frac{\partial E(n)}{\partial \omega_{ij}}$ is labeled "Gradient"

$\frac{\partial E^*(n)}{\partial \omega_{ij}}$ is labeled "Modified Gradient"

$\frac{\partial E^*(n-1)}{\partial \omega_{ij}}$ is labeled "Previous Modified Gradient"

$\Delta \omega_{ij}(n-1)$ is labeled "Previous Change"

$\alpha(n)$ is labeled "Alpha"

$\epsilon(n)$ is labeled "Epsilon"

$\Delta \omega_{ij}(n)$ is labeled "Current Change"

$\omega_{ij}(n+1)$ is labeled "Current Weight"

Examples: NEURAL Procedure

The following examples were executed using the Windows Professional operating system and the SAS software release 9.1.3.

Example 1: Developing a Simple Multilayer Perceptron (Rings Data)

Features

- Specifying Input, Hidden, and Output layers
- Scoring a Test Data Set
- Outputting Fit Statistics
- Creating a Classification Table
- Creating Contour Plots of the Posterior Probabilities

This example demonstrates how to develop a multilayer perceptron network with three hidden units. The example training data set is named SAMPSIO.DMDRING (rings data). It contains a categorical target (C= 0, 1, or 2) plus two continuous inputs (X and Y). There are 180 cases in the data set. The SAMPSIO.DMSRING data set is scored using the scoring formula from the trained models.

Both data sets and the SAMPSIO.DMDRING catalog are stored in the sample library.

Program

PROC GPLOT creates a scatter plot of the Rings training data.

```
proc gplot data=sampsio.dmlring;
  plot y*x=c /haxis=axis1 vaxis=axis2;
  symbol c=black i=none v=dot;
  symbol2 c=red i=none v=square;
  symbol3 c=green i=none v=triangle;
  axis1 c=black width=2.5 order=(0 to 30 by 5);
  axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
  title 'Plot of the Rings Training Data';
run;
```

The PROC DMDB statement is used to create a DMDB catalog entry to be used in the example.

```
proc dmdb batch data=sampsio.dmlring out=dmdring dmdbcat=catring;
var x y ;
class c;
```

The PROC NEURAL statement invokes the procedure. The DATA= option identifies the training data set that is used to fit the model. The DMDBCAT= option identifies the DMDB catalog. The RANDOM= option specifies the random number seed.

```
proc neural data=sampsio.dmlring
           dmdbcat=catring
           random=789;
```

The INPUT statement specifies an interval input layer. The LEVEL= option specifies the measurement level. The ID= option specifies an identifier for the interval input layer.

```
input x y / level=interval id=i;
```

The TARGET statement defines an output layer. The output layer computes predicted values and compares those predicted values with the value of the target variable. The ID= option specifies an identifier for the output layer. The LEVEL= option specifies the target measurement level. By default, for nominal targets the combination function is set to linear, the activation function is set to mlogistic, and the error function is set to mbernoulli.

```
target c / id=o level=nominal;
```

The HIDDEN statement defines the number of hidden units that are used to perform the internal computations. By default, the input units are connected to each hidden unit and each hidden unit is connected to the output unit. The ID= option specifies an identifier for the hidden unit.

```
hidden 3 / id=h;
```

The PRELIM statement causes the procedure to search for the best starting weights for subsequent training. The integer value of 5 specifies to use 5 preliminary runs. The weights from the seed with the smallest objective function among all runs is chosen. Preliminary training can help prevent the network from converging in a local minima.

```
prelim 5;
```

The TRAIN statement trains the network in order to find the best weights (parameter estimates) that accurately reflect the training data.

```
train;
```

The first SCORE statement scores the training data. The OUT= option identifies the output data set that contains outputs. The OUTFIT= option identifies the output data set that contains fit statistics.

```
score out=out outfit=fit;
```

The second SCORE statement specifies the score data set that you want to score in conjunction with training.

```
score data=sampsio.dmsring out=gridout;
title 'MLP with 3 Hidden Units';
run;
```

PROC PRINT lists selected training fit statistics.

```
proc print data=fit noobs label;
var _aic_ _ase_ _max_ _rfpe_ _misc_ _wrong_;
where _name_ = 'OVERALL';
title2 'Fits Statistics for the Training Data Set';
run;
```

PROC FREQ creates a misclassification table for the training data. The F_C variable is the actual target value for each case and the I_C variable is the target value into which the case is classified.

```
proc freq data=out;
tables f_c*i_c;
title2 'Misclassification Table';
run;
```

PROC GPLOT plots the classification results for the training data.

```
proc gplot data=out;
plot y*x=i_c /haxis=axis1 vaxis=axis2;
symbol c=black i=none v=dot;
symbol2 c=black i=none v=square;
symbol3 c=black i=none v=triangle;
```

```

axis1 c=black width=2.5 order=(0 to 30 by 5);
axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
title2 'Classification Results';
run;

```

PROC GCONTOUR produces a contour plot of the posterior probabilities for the scored data set.

```

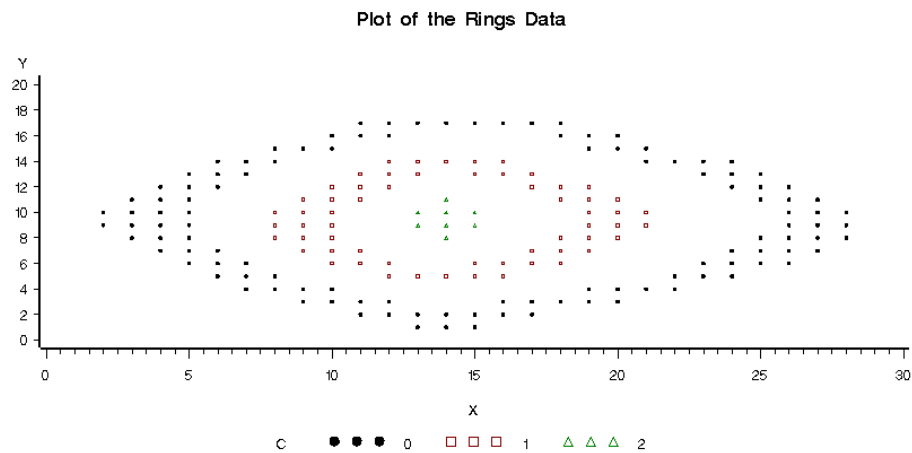
proc gcontour data=gridout;
  plot y*x=p_c1 / pattern ctext=black outline=gray;
  plot y*x=p_c2 / pattern ctext=black outline=gray;;
  plot y*x=p_c3 / pattern ctext=black outline=gray;;
  pattern v=msolid;
  legend frame;
  title2 'Posterior Probabilities';
run;

proc gcontour data=gridout;
  plot y*x=h1 / pattern ctext=black outline=gray;
  plot y*x=h2 / pattern ctext=black outline=gray;;
  plot y*x=h3 / pattern ctext=black outline=gray;;
  pattern v=msolid;
  legend frame;
  title2 'Hidden Unit Values';
run;

```

Output

PROC GPLOT of the Rings Training Data



Notice that the target classes are not linearly separable.

PROC NEURAL: Preliminary Training Output

This section lists the objective function for each preliminary iteration. The parameter estimates (weights) from the iteration number that has the smallest objective function are passed as input for final training. Because the target is nominal, the error function is set to multiple Bernoulli. Therefore, the objective function that is being minimized is the negative log-likelihood. Iteration number 5 has the smallest objective function.

The 17 initial parameter estimates are also listed in this section of the output.

The NEURAL Procedure

Preliminary Training Run	Starting Random Seed	Objective Function Value	Number of Iterations	Terminating Criteria
1	789	0.351915867291	20	
2	761237432	0.200672118175	20	
3	1092694980	0.186015332483	20	
4	577625332	0.291946794763	20	
5	261548896	0.153123214751	20	

Optimization Start
Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function
1	x_h1	-0.399940	-0.005956
2	y_h1	-2.215319	0.016700
3	x_h2	2.570511	-0.038575
4	y_h2	0.672317	-0.020869
5	x_h3	2.589547	0.019067
6	y_h3	-1.945493	0.001492
7	BIAS_h1	2.153111	0.015863
8	BIAS_h2	2.276595	0.046348
9	BIAS_h3	-2.243021	0.009793
10	h1_c1	5.688000	0.002081
11	h2_c1	5.828867	0.000806
12	h3_c1	-5.975478	-0.002290
13	h1_c2	4.632142	-0.002675
14	h2_c2	4.809949	-0.009964
15	h3_c2	-4.803055	-0.006949
16	BIAS_c1	-11.870050	0.008693
17	BIAS_c2	-7.664741	0.003524

Value of Objective Function = 0.1531232148

Levenberg–Marquardt Termination Criteria

This section lists the termination criteria for the Levenberg–Marquardt optimization.

Levenberg–Marquardt Optimization

Minimum Iterations	0
Maximum Iterations	100
Maximum Function Calls	2147483647
Maximum CPU Time	604800
ABSGCONV Gradient Criterion	0.00001
GCONV Gradient Criterion	1E-8
GCONV2 Gradient Criterion	0
ABSFCONV Function Criterion	0
FCONV Function Criterion	0.0001
FCONV2 Function Criterion	0

FSIZE Parameter	0
ABSXCONV Parameter Change Criterion	0
XCONV Parameter Change Criterion	0
XSIZE Parameter	0
ABSCONV Function Criterion	0.0031296692
Trust Region Initial Radius Factor	1
Singularity Tolerance (SINGULAR)	1E-8

Levenberg-Marquardt Iteration Log

At the start of optimization, the procedure lists the number of active constraints, the current value of the objective function, the maximum gradient element, and the radius. The iteration history includes the following:

- the iteration number (iter)
- the number of iteration restarts (rest)
- the number of active constraints (act)
- the value of the objective function (optcrit)
- the difference between the adjacent objective functions (difcrit)
- the maximum of the absolute (projected) gradient components (maxgrad)
- the value of lambda
- the value of rho

The optimization results section lists information specific for the optimization technique.

Levenberg-Marquardt Optimization

Scaling Update of More (1978)

Iter	Function Calls	Objective Function	Function Change	Gradient Element	Lambda	Predicted Change
1	5	0.10070	0.0524	0.0708	0.0138	0.582
2	9	0.08826	0.0124	0.0589	0.417	0.330
3	15	0.05464	0.0336	0.0493	0.00993	0.674
4	19	0.02555	0.0291	0.0726	0.00154	0.667
5	21	0.02328	0.00227	0.1363	0	0.0960
6	23	0.00332	0.0200	0.0154	0	0.897
7	24	0.0007420	0.00258	0.00889	5.65E-7	0.816

Parameter Estimates (MLP Weights)

Optimization Results Parameter Estimates

N Parameter	Estimate	Gradient Objective Function
1 x_h1	-0.511387	0.001713
2 y_h1	-0.882599	-0.000808
3 x_h2	1.778776	0.002143
4 y_h2	-0.027908	0.000091773
5 x_h3	0.548944	-0.001945
6 y_h3	-0.958498	-0.000262

7	BIAS_h1	0.885838	-0.008794
8	BIAS_h2	1.434464	-0.007087
9	BIAS_h3	-0.710813	0.008894
10	h1_c1	80.255869	-0.000302
11	h2_c1	77.816530	-0.000365
12	h3_c1	-73.007083	0.000297
13	h1_c2	42.085041	0.000250
14	h2_c2	36.128046	0.000386
15	h3_c2	-39.517819	-0.000256
16	BIAS_c1	-133.606275	-0.000434
17	BIAS_c2	-54.275823	0.000394

Value of Objective Function = 0.0007420024

List Report of Selected Variables in the Score OUTFIT= Data Set

The example PROC PRINT report of the OUTFIT = data set lists selected fit statistics for the training data. By default, the OUTFIT data set contains two observations. These observations are distinguished by the value of the _NAME_ variable:

- _NAME_ = 'Target Variable Name'
- _NAME_ = 'OVERALL'

Because a WHERE statement was used to select only values of _NAME_ = 'OVERALL', this report contains a single observation.

Notice that the MLP network with 3 hidden units correctly classifies all cases in the training data set.

Fits Statistics for the Training Data Set

Train: Akaike's Information Criterion.	Train: Average Squared Error.	Train: Maximum Absolute Error.	Train: Root Final Prediction Error.	Train: Number of Wrong Classifications.
34.4007	.000018153	0.050798	.004466767	0

PROC FREQ Misclassification Table for the Training Data

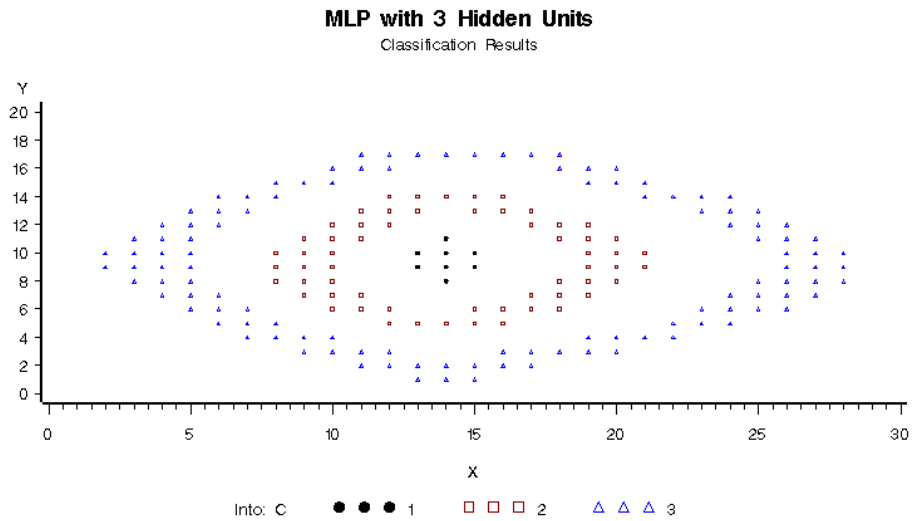
MLP with 3 Hidden Units
Misclassification Table

TABLE OF F_C BY I_C

F_C(From: C)	I_C(Into: C)			Total
Frequency Percent Row Pct Col Pct	1	2	3	
1	8	0	0	8
	4.44	0.00	0.00	4.44
	100.00	0.00	0.00	
	100.00	0.00	0.00	

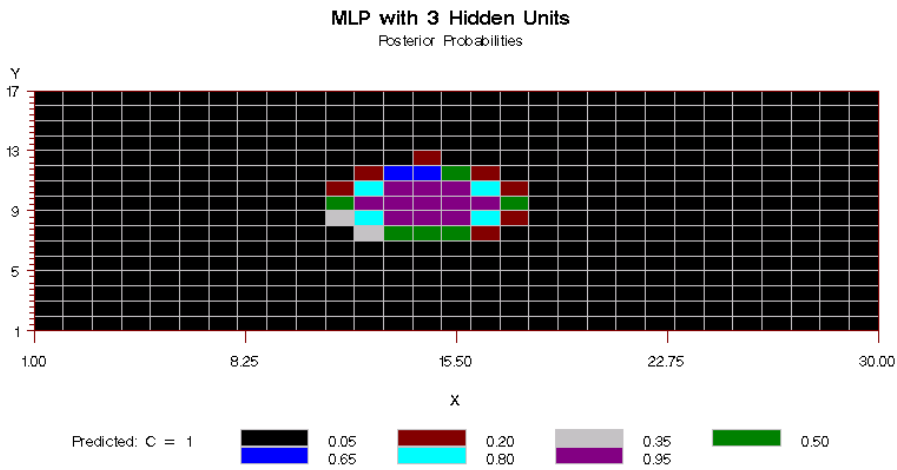
2	0	59	0	59
	0.00	32.78	0.00	32.78
	0.00	100.00	0.00	
	0.00	100.00	0.00	
3	0	0	113	113
	0.00	0.00	62.78	62.78
	0.00	0.00	100.00	
	0.00	0.00	100.00	
Total	8	59	113	180
	4.44	32.78	62.78	100.00

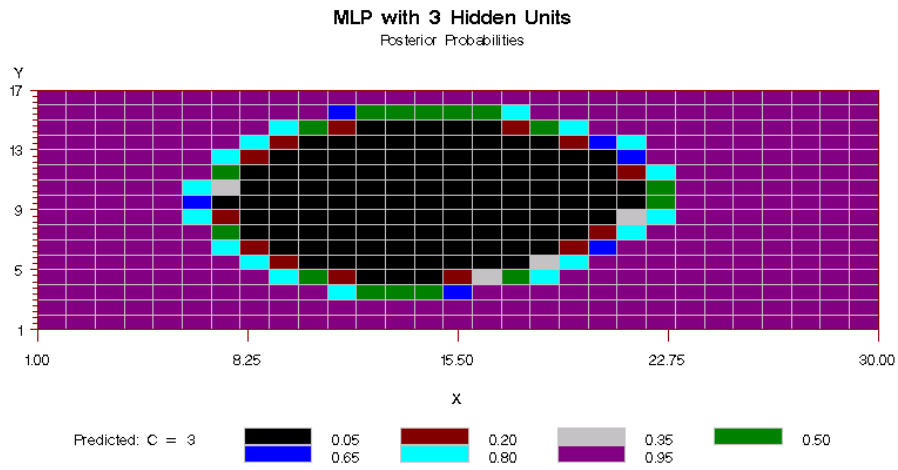
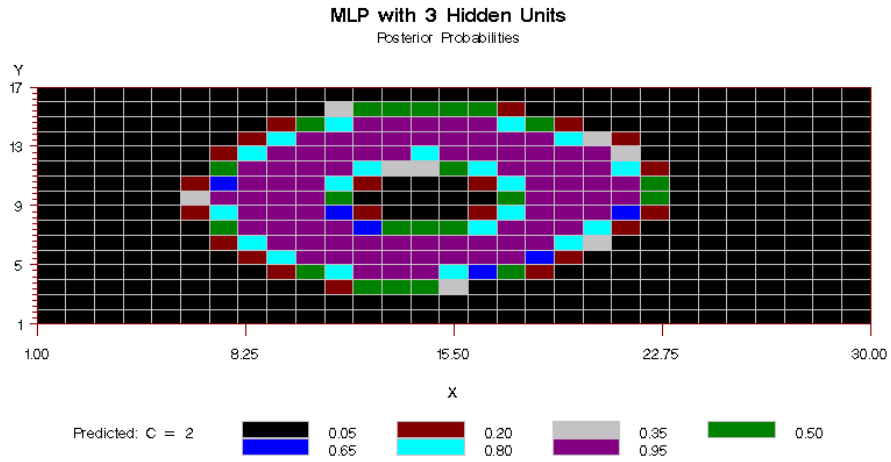
PROC GPLOT Plot of the Classification Results



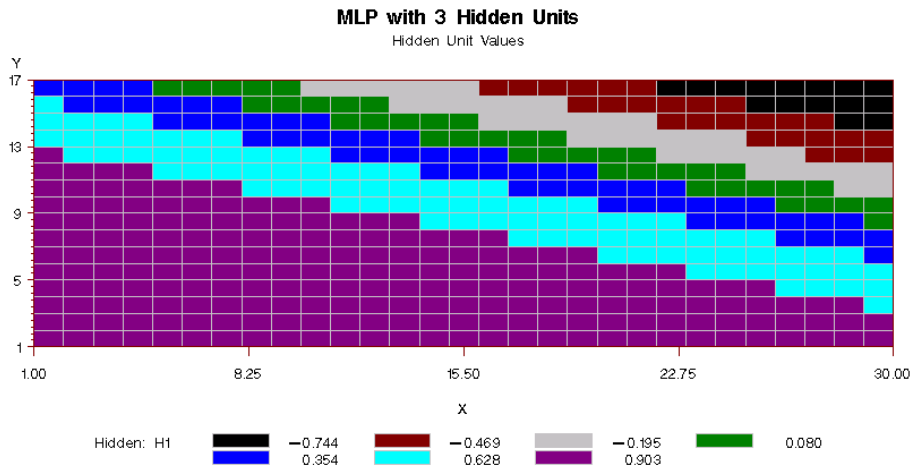
PROC GGONTOUR Plots of the Posterior Probabilities

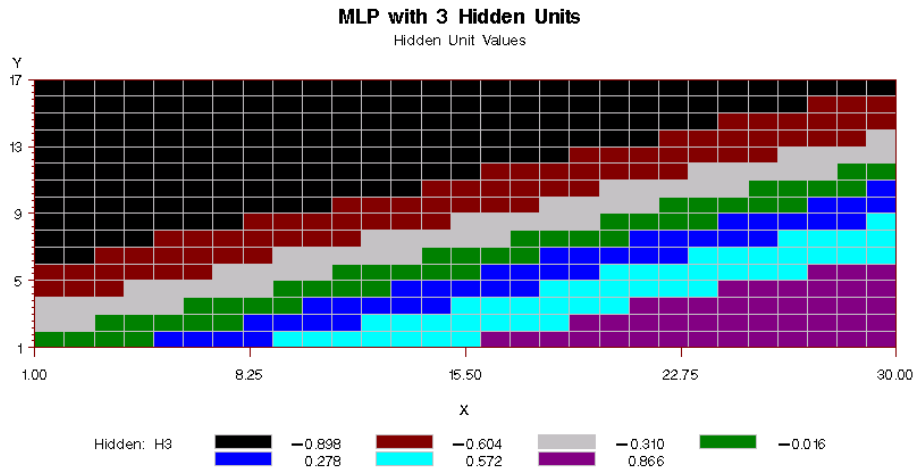
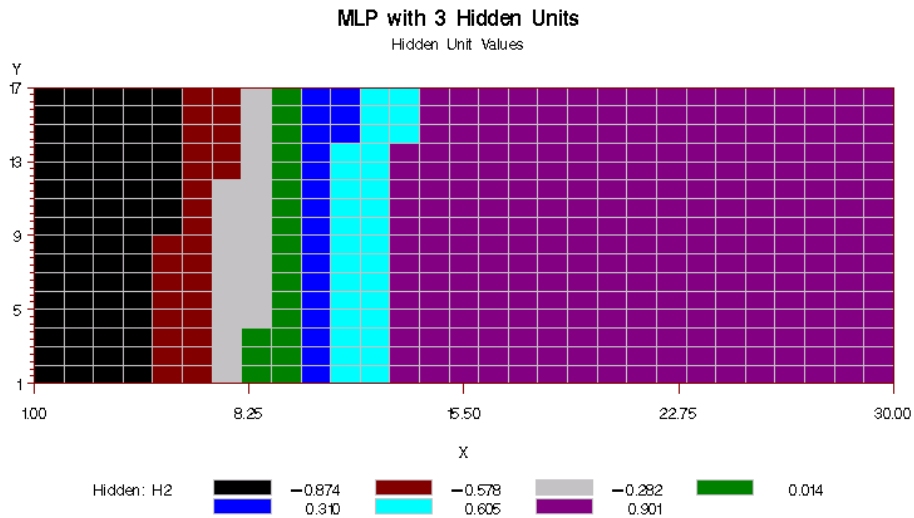
The legend at the bottom of the chart identifies the target level.





PROC GCONTOUR Plots of the Hidden Units





Example 2: Developing a Neural Network for a Continuous Target

Features

- Specifying Input, Hidden, and Output Layers
 - Defining Direct Connections
 - Scoring Data with the Score Statement
 - Outputting Fit Statistics
-

This example demonstrates how to develop a neural network model for a continuous target. A simple multilayer perceptron architecture is used with one hidden unit and direct connections. The example DMBASE training data set SAMPSIO.DMBASE (baseball data set) contains performance measures and salary levels for regular hitters and leading substitute hitters in major league baseball for the year 1986 (Collier 1987). There is one observation per hitter. The continuous target variable is log of salary (logsalar).

Before fitting the neural network model, the number of original model inputs was reduced based on a preliminary stepwise PROC DMREG run. The input set from the model with the smallest SBC (Schwarz's Bayesian Criterion) is used as input to the network. The output from the PROC DMREG analysis can be found in the PROC DMREG chapter, "Example 2. Performing a Stepwise OLS Regression".

The SAMPSIO.DMTBASE data set is a test data set that is scored using the scoring formula from the trained model. The SAMPSIO.DMBASE and SAMPSIO.DMTBASE data sets and the SAMPIO.DMDBASE catalog are stored in the sample library.

Program

The PROC DMDB statement creates a DMDB catalog entry to be used in the example.

```
proc dmdb batch data=sampsio.dmlbase out=dmdbase dmdbcat=catbase;
var no_atbat no_hits no_home no_runs no_rbi no_bb yr_major
    cr_atbat cr_hits cr_home cr_runs cr_rbi cr_bb no_outs
    no_assts no_error salary logsalar;
class name team league division position;
run;
```

The preliminary PROC DMREG run selects the reduced input set.

```
proc dmreg data=dmdbase dmdbcat=catbase
  testdata=sampsio.dmtbase outest=regest;
class league division position;
model logsalar = no_atbat no_hits no_home no_runs no_rbi no_bb
  yr_major cr_atbat cr_hits cr_home cr_runs
  cr_rbi cr_bb league division position no_outs
  no_assts no_error /
  error=normal selection=stepwise
  slentry=0.25 slstay=0.25 choose=sbc;

  title1 'Preliminary DMREG Stepwise Selection';
run;
```

The PROC NEURAL statement invokes the procedure. The DATA= option identifies the training data set that is used to fit the model. The DMDBCAT= option identifies the training catalog. The RANDOM= option specifies the seed that is used to set the random initial weights.

```
proc neural data=sampsio.dmlbase
  dmdbcat=catbase
  random=12345;
```

The INPUT statements specifies the input layers. There are separate input layers for the interval and nominal inputs. The LEVEL= option specifies the measurement level. The ID= option specifies an identifier for each input layer.

```
input cr_hits no_hits no_outs no_error no_bb
  / level=interval id=int;
input division / level=nominal id=nom;
```

The **HIDDEN** statement sets the number of hidden units. The **ID=** option specifies an identifier for the hidden layer. By default, the combination function is set to linear and the activation function is set to hyperbolic tangent.

```
hidden 1 / id=hu;
```

The **TARGET** statement defines an output layer. The output layer computes predicted values and compares those predicted values with the value of the target variable (**LOGSALAR**). The **LEVEL=** option specifies the target measurement level. The **ID=** option specifies an identifier for the output layer. By default, the combination function is set to linear, the activation function is set to the identity, and the error function is set to normal for continuous targets.

```
target logsalar /
    level=interval
    id=tar ;
```

The **CONNECT** statements specify how to connect the layers. The **id-list** specifies the identifier of two or more layers to connect. In this example, each input unit is connected to the hidden unit and to the output unit, and the hidden unit is connected to the output unit.

```
connect int tar;
connect nom tar;
connect int hu;
connect nom hu;
connect hu tar;
```

The **PRELIM** statement does preliminary training using 10 different sets of initial weights. The weights from the preliminary run with the smallest objective function among all runs are retained for subsequent training when using the **TRAIN** statement. Preliminary training can help prevent the network from converging to a bad local minima.

```
prelim 10;
```

The **TRAIN** statement trains the network in order to find the best weights (parameter estimates) to fit the training data. By default, the Levenberg-Marquardt optimization technique is used for small least squares networks, such as the one in this example.

```
train;
```

The **SCORE** statement creates output data sets. The **DATA=** option specifies the data set you want to score. The **OUTFIT=** option creates a data set containing fit statistics.

```
score data=sampsio.dmtbase outfit=netfit
```

The **OUT=** option identifies the output data for predicted values and residuals. The **RENAME=** option renames the variables in the **OUT=** data set containing predicted values and residuals.

```
out=netout(rename=(p_logsalar=predict r_logsalar=residual));
title 'NN:1 Hidden Unit, Direct Connections,
```

```

        and Reduced Input Set';
run;

```

PROC PRINT lists selected variables from the OUTFIT= data set.

```

proc print data=netfit noobs label;
  where _name_ = 'logsalar';
  var _iter_ _pname_ _tmse_ _trmse_ _tmax_;
  title 'Partial Listing of the Score OUTFIT= Data Set';
run;

```

PROC GPLOT plots diagnostic plots for the scored data set.

```

proc gplot data=netout;
  plot logsalar*predict / haxis=axis1 vaxis=axis2;
  symbol c=black i=none v=dot h=3 pct;
  axis1 c=black width=2.5;
  axis2 c=black width=2.5;
  title 'Diagnostic Plots for the Scored Test Baseball Data';
  plot residual*predict / haxis=axis1 vaxis=axis2;

run;
quit;

```

Output

Preliminary Training Output

This section lists the objective function for each preliminary iteration run. The weights from the iteration number that has the smallest objective function are passed as input for final training. Because the target is continuous, the error function is set to normal. Therefore, the objective function that is being minimized is the least squares error. Iteration number 1 has the smallest objective function. The parameter estimates for iteration number 1 are also listed in this section of the output.

NN:1 Hidden Unit, Direct Connections, and Reduced Input Set

The NEURAL Procedure

Preliminary Training Run	Starting Random Seed	Objective Function Value	Number of Iterations	Terminating Criteria
1	12345	0.145357465444	20	FCONV
2	845250737	0.145357464615	19	FCONV
3	111329849	0.282103361977	20	
4	1696138964	0.184771559944	20	
5	1038363354	0.145357466647	19	FCONV
6	1071492826	0.189420230282	20	
7	117568856	0.145359113041	20	
8	1792608669	0.304940595488	20	
9	1691324682	0.183308229556	20	
10	2114796956	0.286050483847	20	

NN:1 Hidden Unit, Direct Connections, and Reduced Input Set

Optimization Start
Parameter Estimates

N Parameter	Estimate	Gradient Objective Function
1 cr_hits_hul	5.240279	-0.000000984
2 no_hits_hul	-0.575284	0.000011756
3 no_outs_hul	-0.298484	0.000001912
4 no_error_hul	0.019049	0.000012696
5 no_bb_hul	-0.097201	0.000008015
6 divisionEast_hul	-0.159479	0.000003441
7 BIAS_hul	4.099012	0.000001798
8 cr_hits_logsalar	0.114451	-8.569367E-8
9 no_hits_logsalar	0.186707	1.1811749E-8
10 no_outs_logsalar	0.156401	2.6624957E-9
11 no_error_logsalar	-0.042491	8.9691117E-8
12 no_bb_logsalar	0.151510	4.8487992E-9
13 divisionEast_logsalar	0.055166	3.48459E-8
14 hul_logsalar	0.839297	-9.836257E-8
15 BIAS_logsalar	5.490837	7.8226732E-8

Value of Objective Function = 0.1453574646

Levenberg-Marquardt Termination Criteria

NN:1 Hidden Unit, Direct Connections, and Reduced Input Set

Levenberg-Marquardt Optimization

Minimum Iterations	0
Maximum Iterations	100
Maximum Function Calls	2147483647
Maximum CPU Time	604800
ABSGCONV Gradient Criterion	0.00001
GCONV Gradient Criterion	1E-8
GCONV2 Gradient Criterion	0
ABSFCNV Function Criterion	0
FCONV Function Criterion	0.0001
FCONV2 Function Criterion	0
FSIZE Parameter	0
ABSXCONV Parameter Change Criterion	0
XCONV Parameter Change Criterion	0
XSIZE Parameter	0
ABSCNV Function Criterion	0.0007747696
Trust Region Initial Radius Factor	1

Singularity Tolerance (SINGULAR) 1E-8

Levenberg-Marquardt Iteration Log

Levenberg-Marquardt Optimization

Scaling Update of More (1978)

Parameter Estimates 15

Optimization Start

Active Constraints 0 Objective Function 0.1453574646
 Max Abs Gradient Element 0.0000126958 Radius 1

Iter	Rest arts	Func Calls	Act Con	Objective Function	Obj Fun Change	Max Abs Gradient Element	Lambda	Actual Over Pred Change
1	0	2	0	0.14536	4.131E-9	8.81E-6	0	1.694

Optimization Results

Iterations 1 Function Calls 4
 Jacobian Calls 3 Active Constraints 0
 Objective Function 0.1453574605 Max Abs Gradient Element 8.810331E-6
 Lambda 0 Actual Over Pred Change 1.6942286514
 Radius 0.0023750664

Convergence criterion (ABSGCONV=0.00001) satisfied.

Parameter Estimates

Optimization Results
 Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function
1	cr_hits_hul	5.242352	-0.000000672
2	no_hits_hul	-0.575574	0.000008165
3	no_outs_hul	-0.298618	0.000001332
4	no_error_hul	0.018999	0.000008810
5	no_bb_hul	-0.097288	0.000005568
6	divisionEast_hul	-0.159583	0.000002389
7	BIAS_hul	4.100399	0.000001236
8	cr_hits_logsalar	0.114473	-4.131105E-8
9	no_hits_logsalar	0.186717	5.6881619E-9
10	no_outs_logsalar	0.156385	1.2635206E-9
11	no_error_logsalar	-0.042475	4.3267007E-8
12	no_bb_logsalar	0.151513	2.3343948E-9
13	divisionEast_logsalar	0.055178	1.6827961E-8
14	hul_logsalar	0.839144	-4.746079E-8

15 BIAS_logsalary 5.490961 3.7702813E-8

Value of Objective Function = 0.1453574605

List Report of Selected Variables in the Score OUTFIT= Data Set

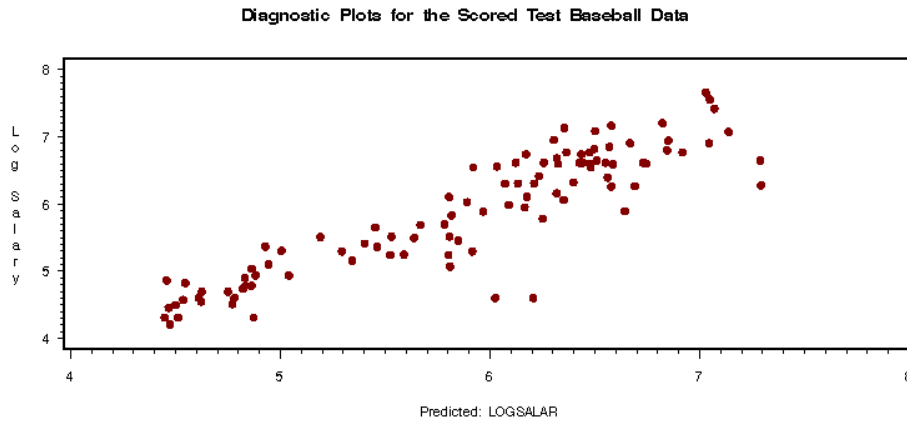
The example PROC PRINT report of the OUTFIT= data set contains selected summary statistics from the scored training data set.

Partial Listing of the Score OUTFIT= Data Set

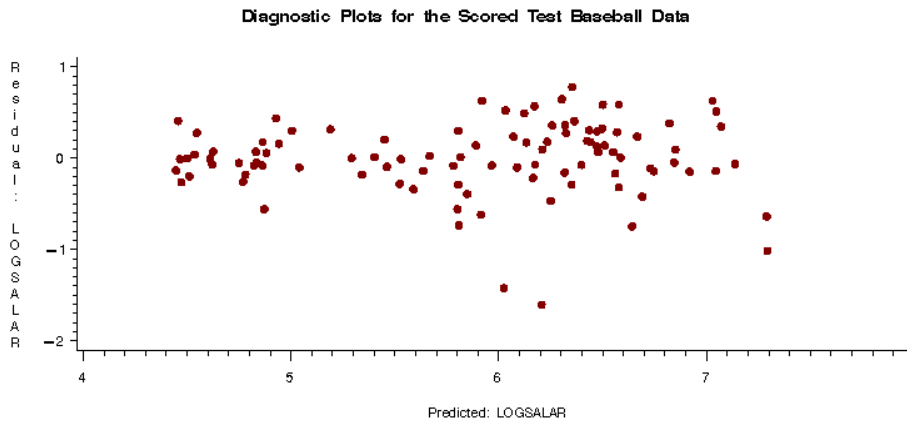
ITER	Test: Mean	Test: Root	Test:
	Squared	Mean Squared	Maximum
	Error.	Error.	Absolute
			Error.
0	0.15595	0.39491	1.60237

Diagnostic Plots for the Scored Test Baseball Data

Plot of the log of salary versus the predicted log of salary.



Plot of the residual values versus the predicted log of salary.



Example 3: Neural Network Hill-and-Plateau Example (Surf Data)

Features

- Specifying input, hidden, and output layers
- Scoring Data with the Score Statement
- Outputting Fit Statistics
- Plotting the Posterior Probabilities

This example demonstrates how to develop a neural network model for a continuous target. A multilayer perceptron architecture is used with 3 and then 30 hidden units. The example test data set is named SAMPSIO.DMDSURF (Surf Data). It contains the interval target HIPL, and two interval inputs X1 and X2. The data set was artificially generated as a surface containing a hill and a plateau. The hill is easily learned by an RBF architecture. The plateau is easily learned by an MLP architecture.

The SAMPSIO.DMTSURF data set is a test data set that is scored using the scoring formula from the trained model. The SAMPSIO.DMDSURF and SAMPSIO.DMTSURF data sets and the SAMPIO.DMDSURF catalog are stored in the sample library.

Program: 3 Hidden Units

The PROC DMDB statement creates a DMDB catalog entry to be used in the example.

```
proc dmdb batch data=sampsio.dmlsurf out=dmdsurf dmdbcat=catsurf;
var x1 x2 d2 d addd adds ces hat hill hipl hiva mult plane ridge spiral
splash steps tanh tanh3 flat;
run;

proc g3d data=dmdsurf;plot x2*x1=hipl
      / grid side ctop=blue caxis=green
      ctext=black zmin=-1.5 zmax=1.5;
      title 'Plot of the Surf Training Data';
      footnote 'Hill Plateau Response Surface';
run;
```

The %LET statement sets the macro variable HIDDEN to 3.

```
title 'Hill & Plateau Data';
%let hidden=3;

proc neural data=sampsio.dmlsurf
      dmdbcat=catsurf
      random=789;
input x1 x2 / id=i;
target hipl / id=o;
```

The MAXITER = option specifies the maximum number of iterations.

```
hidden &hidden / id=h;
```

```

prelim 10;
train maxiter=1000 outest=mlpest;
score data=sampsio.dmtsurf out=mlpout outfit=mlpfit;
title2 'MLP with &hidden Hidden Units';
run;

```

PROC PRINT creates a report of selected fit statistics.

```

proc print data=mlpfit noobs label;
  var _tase_ ;
  where _name_ ='hipl';
  title3 'Fit Statistics for the Test Data';
run;

```

PROC GCONTOUR creates a plot of the predicted values.

```

proc gcontour data=mlpout;
  plot x2*x1=p_hipl / pattern ctext=black outline=gray;
  pattern v=msolid;
  legend frame;
  title3 'Predicted Values';
  footnote;
run;

```

PROC G3D creates a plot of the predicted values. Note that this network underfits badly.

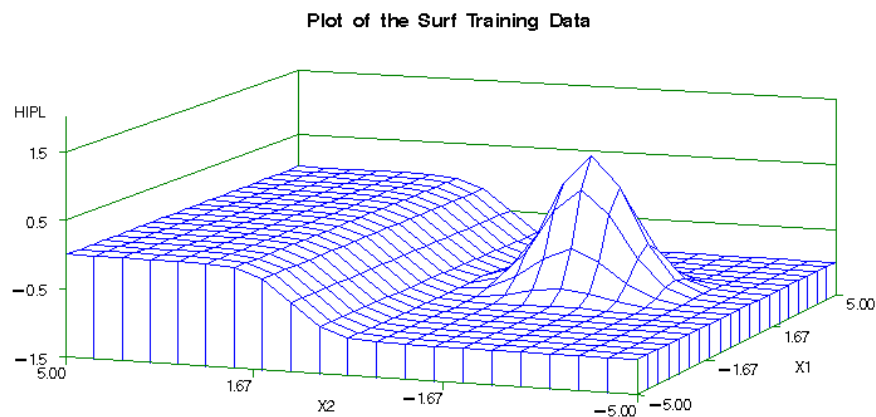
```

proc g3d data=mlpout;
  plot x2*x1=p_hipl / grid side ctop=blue
                    caxis=green ctext=black
                    zmin=-1.5 zmax=1.5;
run;

```

Output: 3 Hidden Units

PROC GCONTOUR Plot of the Surf Training Data



Hill Plateau Response Surface

PROC Neural Output

The NEURAL Procedure

Preliminary Training Run	Starting Random Seed	Objective Function Value	Number of Iterations	Terminating Criteria
1	789	0.040226670802	20	
2	761237432	0.036731388712	20	
3	1092694980	0.041868838662	20	
4	577625332	0.052159528829	20	
5	261548896	0.035928397351	20	
6	616485149	0.040297830797	20	
7	692687363	0.039729773755	20	
8	1510804008	0.041407025962	20	
9	1385020003	0.035827176635	20	
10	1070679467	0.04166691836	20	

Optimization Start
Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function
1	x1_h1	6.064004	-0.000002674
2	x2_h1	0.880274	0.000005284
3	x1_h2	0.048809	0.000000537
4	x2_h2	-4.988958	0.000000604
5	x1_h3	-5.916343	-0.000006049
6	x2_h3	0.730854	-0.000020662
7	BIAS_h1	-3.004936	-0.000010488
8	BIAS_h2	1.791982	0.000002171
9	BIAS_h3	0.864474	-0.000012608
10	h1_hipl	-0.261095	0.000017865
11	h2_hipl	-0.484358	-0.000010698
12	h3_hipl	-0.265490	0.000012340
13	BIAS_hipl	-0.490112	-0.000018205

Value of Objective Function = 0.0358271766

Levenberg-Marquardt Optimization

Minimum Iterations	0
Maximum Iterations	1000
Maximum Function Calls	2147483647
Maximum CPU Time	604800
ABSGCONV Gradient Criterion	0.00001
GCONV Gradient Criterion	1E-8
GCONV2 Gradient Criterion	0

```

ABSFCNV Function Criterion          0
FCNV Function Criterion             0.0001
FCNV2 Function Criterion           0
FSIZE Parameter                    0
ABSXCONV Parameter Change Criterion 0
XCONV Parameter Change Criterion   0
XSIZE Parameter                    0
ABSCNV Function Criterion           0.0002174074
Trust Region Initial Radius Factor  1
Singularity Tolerance (SINGULAR)   1E-8
    
```

Levenberg-Marquardt Optimization

Scaling Update of More (1978)

Parameter Estimates 13

Optimization Start

```

Active Constraints          0   Objective Function   0.0358271766
Max Abs Gradient Element  0.0000206616   Radius           1
    
```

Iter	Rest arts	Func Calls	Act Con	Objective Function	Obj Fun Change	Max Abs Gradient Element	Lambda	Actual Over
								Pred Change
1	0	2	0	0.03583	1.989E-7	0.000019	0	0.568
2	0	3	0	0.03583	1.764E-7	0.000017	0	0.698
3	0	4	0	0.03583	1.553E-7	0.000016	0	0.869
4	0	5	0	0.03583	1.44E-7	0.000015	0	1.043
5	0	6	0	0.03583	1.351E-7	0.000015	0	1.247
6	0	7	0	0.03583	1.297E-7	0.000014	0	1.415
7	0	8	0	0.03583	1.259E-7	0.000013	0	1.585
8	0	9	0	0.03583	1.235E-7	0.000013	0	1.700
9	0	10	0	0.03583	1.221E-7	0.000013	0	1.805
10	0	11	0	0.03583	1.213E-7	0.000013	0	1.864

Optimization Results

```

Iterations          10   Function Calls          13
Jacobian Calls      12   Active Constraints      0
    
```

Optimization Results

```

Objective Function   0.0358257445   Max Abs Gradient Element  0.0000127206
Lambda              0   Actual Over Pred Change  1.864321939
Radius              0.0043652733
    
```

Convergence criterion (FCNV=0.0001) satisfied.

Optimization Results
Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function
1	x1_h1	6.020084	-0.000000489
2	x2_h1	0.823365	0.000011128
3	x1_h2	0.049663	-3.227714E-8
4	x2_h2	-4.986906	2.1975494E-8
5	x1_h3	-5.848915	-0.000001897
6	x2_h3	0.767294	-0.000012721
7	BIAS_h1	-3.013469	0.000000560
8	BIAS_h2	1.791192	0.000000132
9	BIAS_h3	0.889276	-0.000003004
10	h1_hi1	-0.262306	0.000000451
11	h2_hi1	-0.484458	1.7993028E-8
12	h3_hi1	-0.266660	0.000000336
13	BIAS_hi1	-0.490183	-0.000000312

Value of Objective Function = 0.0358257445

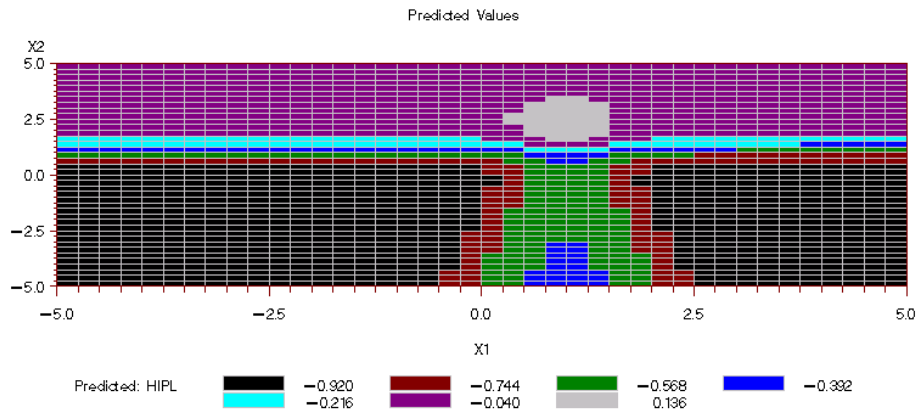
PROC PRINT Report of Selected Fit Statistics for the Scored Test Data Set

Hill & Plateau Data
MLP with 3 Hidden Units
Fit Statistics for the Test Data

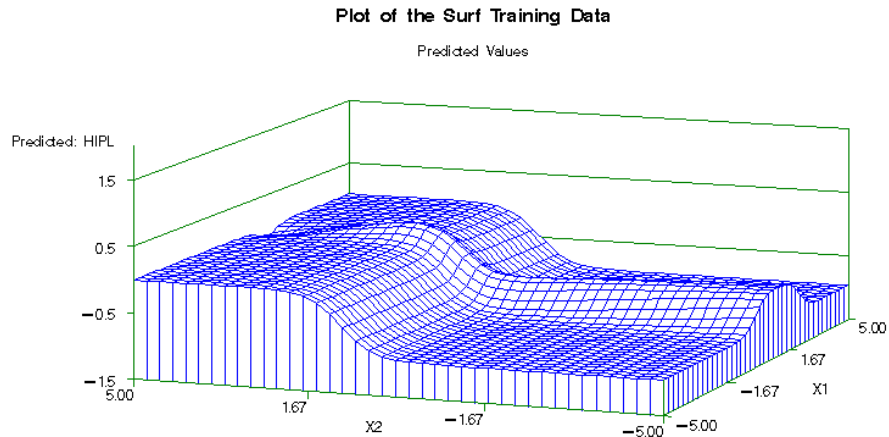
Test:	Test: Lower	Test: Upper
Average	95% Conf.	95% Conf.
Squared	Limit for	Limit for
Error.	TASE.	TASE.
0.036830	0.028999	0.045583

GCONTOUR Plot of the Predicted Values

Plot of the Surf Training Data



G3D Plot of the Predicted Values



Program: 30 Hidden Units

The PROC DMDB statement creates a DMDB catalog entry to be used in the example.

```
proc dmdb batch data=sampsio.dmlsurf out=dmdsurf dmdbcat=catsurf;
var x1 x2 d2 d addd adds ces hat hill hipl hiva mult plane ridge spiral
splash steps tanh tanh3 flat;
run;
```

The %LET statement sets the macro variable HIDDEN to 30.

```
title 'Hill & Plateau Data';
%let hidden=30;

proc neural data=sampsio.dmlsurf
    dmdbcat=dmdsurf
    random=789;
input x1 x2 / id=i;
target hipl / id=o;
```

The MAXITER= option specifies the maximum number of iterations. Note that, despite the simple form of the function being learned in this example, attempting to interpret the network weights is futile.

```
hidden &hidden / id=h;
prelim 10;
train maxiter=1000 outest=mlpest2;
score data=sampsio.dmts surf out=mlpout2 outfit=mlpfit2;
title2 'MLP with &hidden Hidden Units';
run;
```

PROC PRINT creates a report of selected fit statistics

```
proc print data=mlpfit2 noobs label;
var _tase_ _tase1_ _taseu_;
```

```

    where _name_ ='hipl';
    title3 'Fit Statistics for the Test Data';
run;

```

PROC GCONTOUR creates a plot of the predicted values.

```

proc gcontour data=mlpout2;
  plot x2*x1=p_hipl / pattern ctext=black coutline=gray;
  pattern v=msolid;
  legend frame;
  title3 'Predicted Values';
  footnote;
run;

```

PROC G3D creates a plot of the predicted values. Note that even with 30 hidden units and 121 weights in the network, there are visible discrepancies between the actual target function and the network output. It would take about 50 hidden units to reduce the discrepancies to the point they could not be seen in the plot.

```

proc g3d data=mlpout2;
  plot x2*x1=p_hipl / grid side ctop=blue
                    caxis=green ctext=black
                    zmin=-1.5 zmax=1.5;
run;

```

Output: 30 Hidden Units

PROC PRINT Report of the Average Squared Error for the Scored Test Data Set

```

          Hill & Plateau Data
          MLP with 30 Hidden Units
          Fit Statistics for the Test Data

```

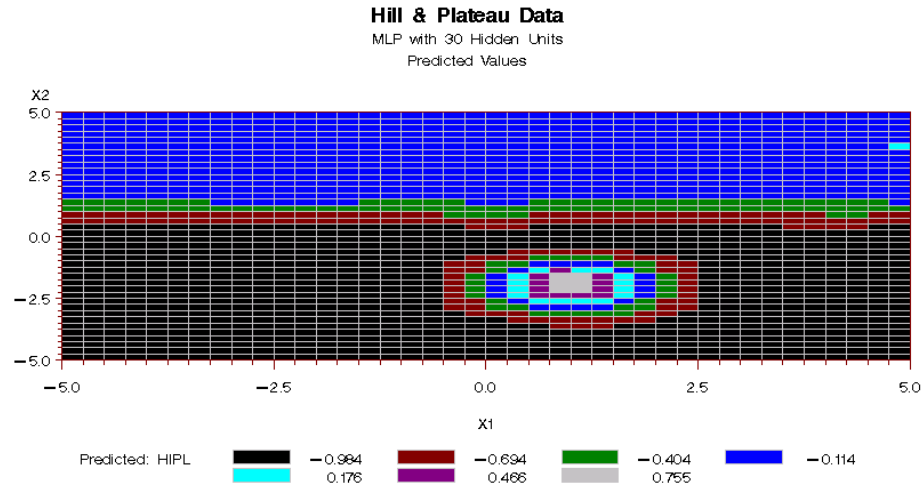
```

          Average
          Squared
          Error.

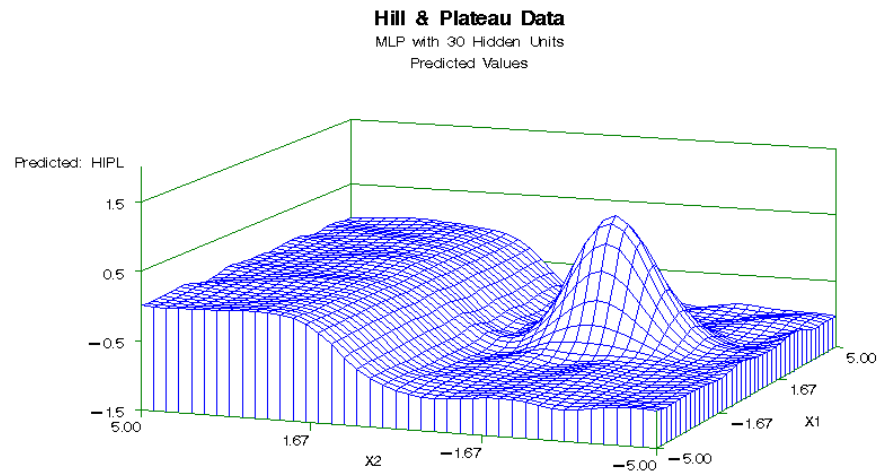
          .000657317

```

GCONTOUR Plot of the Predicted Values



G3D Plot of the Predicted Values



References

- Berry, M. J. A. and Linoff, G. (1997), *Data Mining Techniques for Marketing, Sales, and Customer Support*, New York: John Wiley and Sons, Inc.
- Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, New York: Oxford University Press.
- Bigus, J. P. (1996), *Data Mining with Neural Networks: Solving Business Problems - from Application Development to Decision Support*, New York: McGraw-Hill.
- Collier Books (1987), *The 1987 Baseball Encyclopedia Update*, New York: Macmillan Publishing Company.
- Fahlman, S.E. (1989), "Faster-Learning Variations on Back-Propagation: An Empirical Study", in Touretzky, D., Hinton, G., and Sejnowski, T., eds., *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufman, 38-51.

- Michie, D., Spiegelhalter, D. J. and Taylor, C. C. (1994), *Machine Learning, Neural and Statistical Classification*, New York: Ellis Horwood.
- Riedmiller, M. and Braun, H. (1993), "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *Proceedings of the IEEE International Conference on Neural networks 1993*, San Francisco: IEEE.
- Ripley, B. D. (1996), *Pattern Recognition and Neural Networks*, New York: Cambridge University Press.
- Sarle, W. S. (1994a), "Neural Networks and Statistical Models," *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc., 1538–1550.
- Sarle, W. S. (1994b), "Neural Network Implementation in SAS Software," *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc., 1550–1573.
- Sarle, W. S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface*, Cary, NC: SAS Institute Inc.
- Schiffmann, W., Joost, M., and Werner, R. (1994), "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons", ftp://archive.cis.ohiostate.edu/pub/neuroprose/schiff.bp_speedup.ps.Z.
- Smith, M. (1993), *Neural Networks for Statistical Modeling*, New York: Van Nostrand Reinhold.
- Weiss, S. M. and Kulikowski, C. A., (1991), *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, San Mateo, CA: Morgan Kaufmann.